Oracle® Cloud

Using Oracle Developer Cloud Service

E37145-38
October 2019

Oracle Cloud Using Oracle Developer Cloud Service,

E37145-38

Primary Author: Himanshu Marathe

Contributing Authors: Eric Jendrock

Contributors: Oracle Developer Cloud Service development team

Contents

3 Plan Your Project

4 Use Project's Repositories

5 Collaborate with Your Team

6 Build Applications and Deploy to Oracle Cloud

9 Organization and Project Management

10 Use IDEs

Preface

Using Oracle Developer Cloud Service describes how to use Oracle Developer Cloud Service to commit source code files to the Git repositories, track issues, review the source code and merge repository branches, share information through wikis, build applications, and deploy them.

Topics:

Audience

Using Oracle Developer Cloud Service is intended for:

- Customers developing applications that deploy to Oracle Cloud or to an on-premise environment
- Developers using NetBeans IDE, Eclipse IDE or Oracle Enterprise Pack for Eclipse (OEPE), or Oracle JDeveloper to access and use Oracle Developer Cloud Service
- Project Administrators or Project Managers that support the development team

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc`.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info` or visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs` if you are hearing impaired.

Related Resources

For more information, see these Oracle resources:

- Oracle Cloud

`http://cloud.oracle.com`

- About Oracle Cloud in *Getting Started with Oracle Cloud*
- About Oracle Java Cloud Service - SaaS Extension in *Using Oracle Java Cloud Service - SaaS Extension*
- About Oracle Java Cloud Service in *Administering Oracle Java Cloud Service*
- About Oracle Application Container Cloud Service in *Using Oracle Application Container Cloud Service*

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

1 Overview

Learn about Oracle Developer Cloud Service, its projects, components, roles, and how to access the service.

See Oracle Cloud Terminology in *Getting Started with Oracle Cloud* for definitions of terms found in this and other documents in the Oracle Cloud library.

What is Oracle Developer Cloud Service?

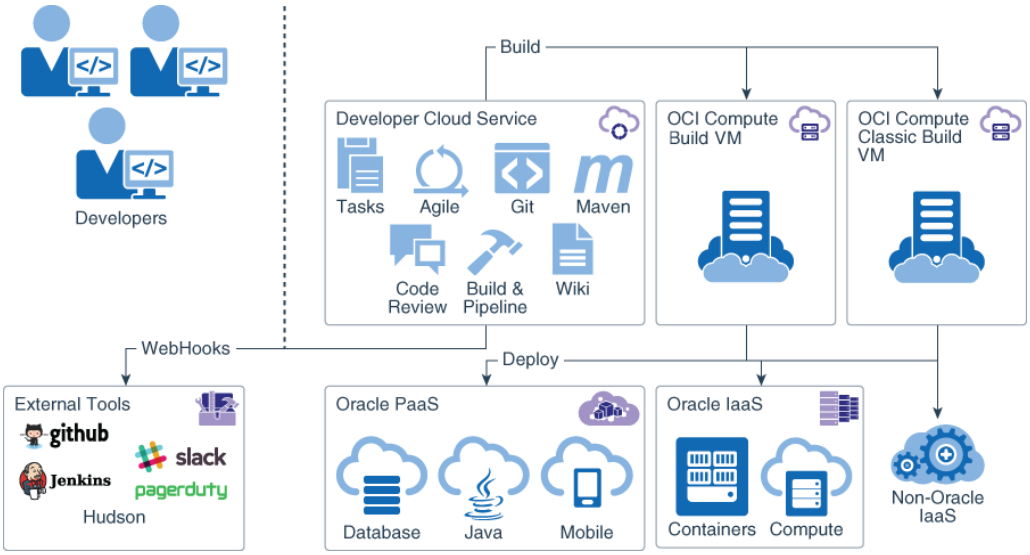
Oracle Developer Cloud Service (DevCS) is a cloud-based software development Platform as a Service (PaaS) and a hosted environment for your application development infrastructure. It provides an open source standards-based solution to plan, develop, collaborate, build, and deploy applications in Oracle Cloud.

DevCS can make your software development experience easier by providing:

- Git repositories, so you can host your application source code files on Oracle Cloud and track their versions
- Maven repositories, so you can host library and binary dependencies of your Maven applications
- Build system, so you can automate builds with continuous integration and continuous testing using Oracle Cloud Infrastructure Compute or Oracle Cloud Infrastructure Compute Classic Virtual Machines
- Deploy system, so you can automate deployment to Oracle Java Cloud Service, Oracle Application Container Cloud Service, and Oracle Java Cloud Service - SaaS Extension
- Code review, so you can peer review the code updates
- Issue tracking system, so you can track tasks, defects, and features
- Wikis, so you can collaborate with your team
- Integration with Eclipse IDE, Oracle JDeveloper, and NetBeans IDE, so you can access and update source code files from your favorite IDEs
- Integration with external software using webhooks, so you can send DevCS event notifications to the external software

As soon as your DevCS account is active, you can access its features immediately.

This diagram shows a basic workflow in DevCS.



The diagram shows how a team of developers work together and use DevCS to write code, track issues, build, and deploy applications.

- The team updates code in the hosted Git source code management repositories and uploads dependencies to the hosted Maven repository.
- As the software development progresses, the team reviews the code updates using the built-in code review system.
- Using the built-in issue tracking system, the team tracks tasks and defects and then, using Agile boards, they track their development progress.
- To run builds and tests of the software application, the team sets up build jobs and pipelines that run builds on Oracle Cloud Infrastructure Compute or Oracle Cloud Infrastructure Compute Classic Virtual Machines.
- The team deploys the built artifacts to Oracle PaaS, Oracle IaaS, or on-premise non-Oracle IaaS instances.
- Using webhooks, the team sends out DevCS event notifications to external tools and software, such as Slack and Jenkins.

Watch a short video to learn more about DevCS.

Video

A Word About Oracle Cloud Infrastructure

If you're an Oracle Cloud Infrastructure (OCI) user, you can create a **Developer** instance. If you're an Oracle Cloud Infrastructure Classic (OCI Classic) user, you can create a **Developer Classic** instance. These instances have no functional differences.

On OCI, DevCS builds run on Oracle Cloud Infrastructure Compute (OCI Compute) virtual machines (VMs). Project artifacts are stored in an Oracle Cloud Infrastructure Object Storage (OCI Object Storage) bucket.

On OCI Classic, DevCS builds run on Oracle Cloud Infrastructure Compute Classic (OCI Compute Classic) virtual machines (VMs). Project artifacts are stored in an

Oracle Cloud Infrastructure Object Storage Classic (OCI Object Storage Classic) container.

Before you can use DevCS on OCI or OCI Classic, you must configure a connection to Compute and Object Storage instances. See Connect to OCI or OCI Classic.

What is a Project?

A project is a collection of DevCS features.

You can use a project to host source code files, track issues, collaborate on code, build, and deploy your applications. A project can host multiple Git repositories. Each Git repository can have multiple branches and hundreds of code files. You can create a merge request for each branch of the Git repository and ask reviewers to review the code. You can create and configure multiple build jobs to generate different project artifacts that you can deploy to Oracle Cloud or your on-premise web server.

Watch a short video to learn more about DevCS projects.

Video

Project Visibility

A project can be a private project or a shared project. There are no public projects. You can define a project's visibility when you create it, or from its properties page later.

Private projects are accessible to invited users only. Shared projects are accessible to all users of the organization. Any user can view the source code, create or update issues, edit wiki pages, and interact with project builds. However, only invited users can make updates to the source code in Git repositories, create and run build jobs, and perform deployment operations.

Key Concepts, Components, and Terms

Before you use Oracle Developer Cloud Service (DevCS), make yourself familiar with these key concepts, components, and terms.

See Oracle Cloud Terminology in *Getting Started with Oracle Cloud* to understand the basic terminology used to describe the Oracle Cloud features and services.

Term	Description
Organization	The top-most entity in the project structure of DevCS. Think of an organization as the umbrella for all the projects in a given identity domain.
Git repository	A Source Code Management (SCM) and distributed version control tool to host source code files.
Maven repository	A hosted repository to store build artifacts, library files, and dependencies for Maven applications.
Issue tracker	A built-in issue management system to create and track tasks, defects, and features.
Merge Request and Code review	A method to merge a Git repository branch with another branch. Before merging the branches, team members can review differences between files of both branches and provide their feedback.

Term	Description
Build System	A built-in system to define and automate builds of your applications.
Build VM	A Virtual Machine (VM) in Oracle Cloud Infrastructure Compute Classic, which run project builds. Only one build can run on a Build VM at a time.
Build VM Template	A template that defines the operating system and the software installed on the Build VM.
Build executor	Software that runs a build.
Wiki	Built-in wiki system to help your team author and manage wiki pages.
Deployment configuration	A configuration to deploy a build artifact to an Oracle Java Cloud Service, Oracle Application Container Cloud Service, or Oracle Java Cloud Service - SaaS Extension instance.
Oracle Cloud Infrastructure	Oracle Cloud Infrastructure is a set of cloud services that enable you to build and run a wide range of applications and services in a highly available hosted environment. Oracle Cloud Infrastructure offers high-performance compute capabilities (as physical hardware instances) and storage capacity in a flexible overlay virtual network that is securely accessible from your on-premises network. If you're new to OCI, see Key Concepts and Terminology to understand OCI concepts and terminologies.
Oracle Cloud Infrastructure Compute (OCI Compute) and Oracle Cloud Infrastructure Compute Classic (OCI Compute Classic)	Service that hosts virtual machines (VMs) on Oracle Cloud with all the necessary storage and networking resources. DevCS uses the VMs to run project builds. For more information, see Overview of the Compute Service in *Oracle Cloud Infrastructure Documentation* and About Compute Classic in *Using Oracle Cloud Infrastructure Compute Classic*.
Oracle Cloud Infrastructure Object Storage (OCI Object Storage) and Oracle Cloud Infrastructure Object Storage Classic (OCI Object Storage Classic)	Oracle Cloud service that hosts containers on Oracle Cloud to store project data. DevCS uses the containers to archive build artifacts and Maven artifacts, and export project data. For more information, see Overview of Object Storage in *Oracle Cloud Infrastructure Documentation* and About Oracle Cloud Infrastructure Object Storage Classic in *Using Oracle Cloud Infrastructure Object Storage Classic*.
Oracle Java Cloud Service	Oracle Cloud service to deploy web applications to a public Oracle WebLogic Server domain on Oracle Cloud. For more information about the service, see About Oracle Java Cloud Service in *Administering Oracle Java Cloud Service*.
Oracle Application Container Cloud Service	Oracle Cloud service to deploy Java SE, Node.js, PHP, Python, Ruby, Go, Java EE 7 (or later), and .NET applications to Oracle Cloud. For more information about the service, see About Oracle Application Container Cloud Service in *Using Oracle Application Container Cloud Service* and About Your Application and Oracle Application Container Cloud Service in *Developing for Oracle Application Container Cloud Service*.
Oracle Java Cloud Service - SaaS Extension	Oracle Cloud service to deploy Java web applications that are extensions to existing Oracle SaaS products, such as CRM, HCM, and so on. For more information about the service, see About Oracle Java Cloud Service - SaaS Extension in *Using Oracle Java Cloud Service - SaaS Extension*.

Roles

Oracle Developer Cloud Service (DevCS) has two types of roles, Identity Domain and Project. These roles allow their grantees to control aspects of the service or projects, or both.

Identity Domain Roles

Those who have an Oracle Cloud identity domain role control which users can sign into an Oracle Cloud service and what features they can access.

To find out how to grant an identity domain role to someone, see Learn About Cloud Account Roles and Add Users and Assign Roles in *Getting Started with Oracle Cloud*.

There're several types of identity domain roles, each of which has different privileges:

This role...	Enables a user to:
`TenantAdminGroup` (Identity Domain Administrator)	Add and manage users and their roles in the identity domain. By default, the role is assigned to the service subscriber who can assign the role to other users.
`DEVCS_APP_ENTITLEMENT_ADMINISTRATOR` (Administrator Role for Developer Cloud Service Provisioning)	Create a DevCS instance in an identity domain.
`DEVELOPER_ADMINISTRATOR` (Developer Service Administrator)	Update the organization details of DevCS. The user with this role is also called as the *Organization Administrator*. The role is available after a DevCS instance is created. By default, the role is assigned to the service subscriber who can assign the role to other users.
`DEVELOPER_USER` (Developer Service User)	Create and access DevCS projects. All users of DevCS must be assigned this role. The role is available after a DevCS instance is created. Note that this role doesn't allow the user to update the organization details.

Additional Roles

To access other Oracle Cloud services from DevCS, these roles should be assigned to the appropriate users:

This role...	Enables a user to:
`JaaS_Administrator` (Java Administrators)	Connect to Oracle Java Cloud Service and Oracle Java Cloud Service - SaaS Extension and deploy build artifacts.
`APaaS_Administrator` (APaaS Administrator)	Connect to Oracle Application Container Cloud Service and deploy build artifacts.
`OCI_Administrator` (OCI Administrator)	Create and manage OCI compartments and its resources.

This role...	Enables a user to:
`Compute.Compute_Operations` (Service Administrator)	Create and manage DevCS VMs on Oracle Cloud Infrastructure Compute Classic.
`Storage.Storage_Administrator` (Service Administrator)	Create and manage containers, and export to and import project data from Oracle Cloud Infrastructure Object Storage Classic.

Project Roles

To take part in a project, the user must be a project member.

This role...	Enables a user to:
Organization Administrator	Access and manage all projects of the organization, and set connections to OCI and OCI Classic. In this documentation, the  icon indicates the organization administrator. Usually, organization heads and members of the IT department are assigned this role. The creator of the service instance is automatically assigned this role.
Owner	Access all components of the project. An owner can perform project management and administrative tasks such as add or remove Git repositories, manage project users, assign default reviewers, configure Webhooks, and manage ATOM/RSS Feeds handlers. In this documentation, the  icon indicates a project owner. Usually, project managers and team leaders are assigned this role. The creator of the project is automatically assigned this role.
Member	Access most components of the project, but restricts project management or administrative actions. In this documentation, the  icon indicates a project member. Usually, developers, QA, technical writers, and other members are assigned this role.
Non-member	View components of a shared project, but can't update source code files, can't create or configure build jobs and deployment configurations, and can't manage the project. The user can enter comments, update issues, view wikis, and download build artifacts.

2
Get Started

Where you start in Oracle Developer Cloud Service (DevCS) depends on the type of role you are in, identity domain role or project role.

If you're ...	You can:
The service subscriber	• Create the DevCS Instance • Add Users to the Identity Domain
An organization administrator	• Connect to OCI or OCI Classic • Update the Organization's Display Name and Description
A DevCS user	• Access Oracle Developer Cloud Service • Set Your User Preferences • Set Up IDEs and Git Clients • Create a Project or Open a Project

Know Yourself

Before you use DevCS, know your identity domain roles and project roles.

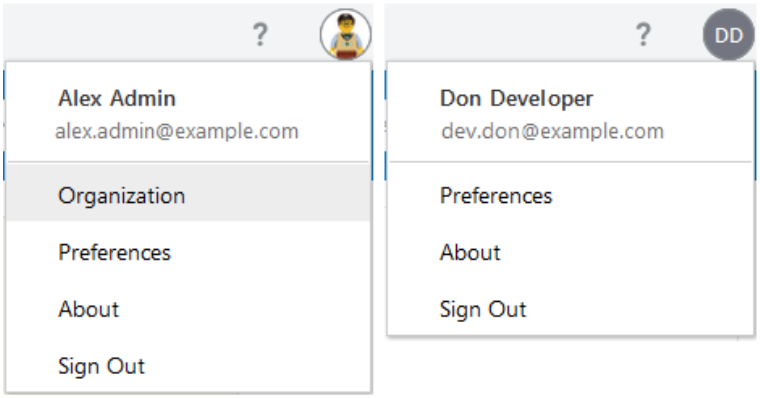
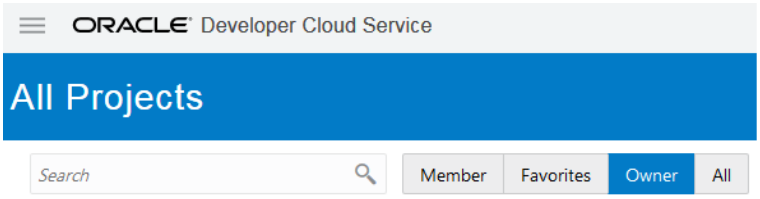
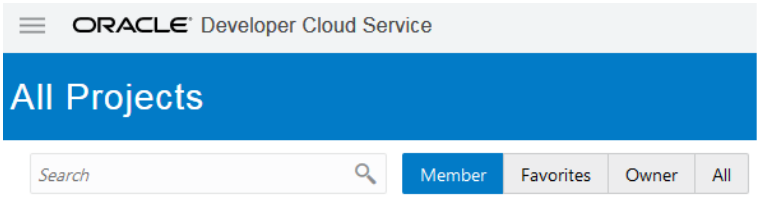
Know Your Identity Domain Roles

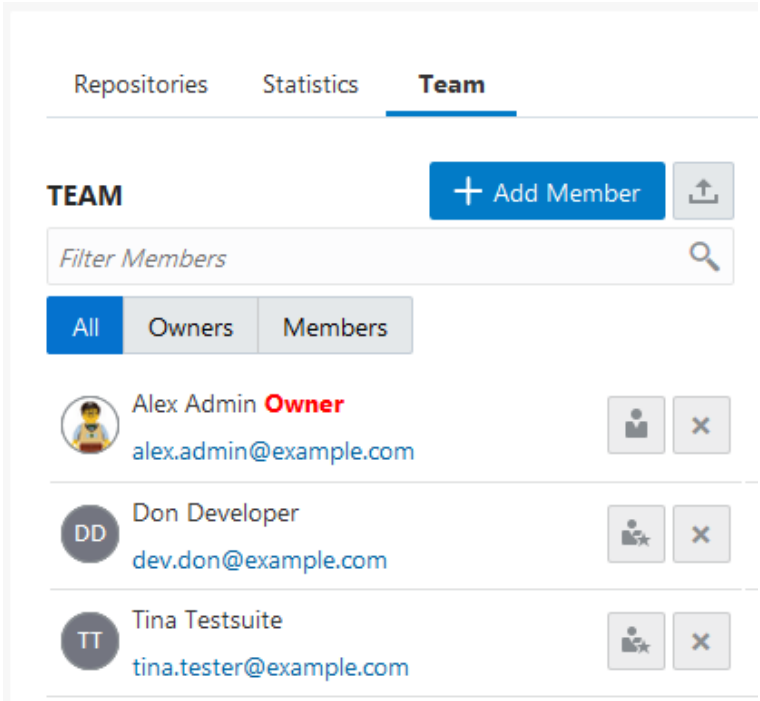
You can check your identity domain roles from the Oracle Cloud My Services page.

1. In a web browser, go to `https://cloud.oracle.com`, and click **Sign In**.
2. On the Sign-In page, in **Account**, enter your account or tenant name, and click **Next**.
3. On the Oracle Cloud Account sign-in page, enter your Oracle Cloud account credentials, and click **Sign In**.
4. If you land on the My Oracle Services page, click **Infrastructure Dashboard**.
5. On the OCI Console, click  in the top-left corner.
6. Under **Governance and Administration**, select **Identity**, and then select **Federation**.
7. Select the identity service provider.
8. In the **IDCS Username** column, click your name.
9. Click **Manage Service Roles**.
10. In the **Service** column, find the Developer service.
11. Click the three vertical dots on the right, and select **Manage instance access**.
12. In the **Instance Role** column, note your roles.
13. Click **Cancel** to return to the last page.

Know Your Project Role

To find out your project role in a project, sign in to DevCS and click the project's name.

Action	Do this:
Know if you're assigned the Organization Administrator role	In the branding bar, click the user avatar. If you see the **Organization** option in the menu, you're assigned the Organization Administrator role. For example, in this graphic, Alex Admin's user menu shows the **Organization** option, but Don Developer's user menu doesn't. This indicates Alex is assigned the Organization Administrator role, but Don isn't. 
Find projects where you're a project owner	On the Welcome page, click the **Owner** toggle button. 
Find projects where you're a project member	On the Welcome page, click the **Member** toggle button. 

Action	Do this:
Find your role in a project you can access	1. In the navigation bar, click **Project Home** . 2. To the right side of the page, click the **Team** tab. If you see the **Owner** tag next to your name, you're a project owner. If you don't see the **Owner** tag next to your name, you're a project member. If you can't find your name, you're a project non-member. For example, in this graphic, Alex Admin is a project owner, and Don Developer and Tina Testsuite are project members. 

What Actions Can Each Role Perform in a Project?

Depending upon the project role assigned to you, you can perform various actions on the pages of DevCS. Non-members can perform actions in a shared project only.

The following actions can be performed across all pages of the DevCS web interface.

Action	Organization Administrator	Owner	Member	Non-Member
Collapse or expand the left navigation bar	✓	✓	✓	✓

Action	Organization Administrator	Owner	Member	Non-Member
Switch to another project	✓	✓	✓	✓
Open the help menu	✓	✓	✓	✓
Use the user menu	✓	✓	✓	✓

Project Home Page

Recent Activities Feed

Action	Organization Administrator	Owner	Member	Non-Member
View the recent activities feed	✓	✓	✓	✓
Filter the recent activities feed	✓	✓	✓	✓
Search activities	✓	✓	✓	✓

Repositories Tab

Action	Organization Administrator	Owner	Member	Non-Member
Create a Git repository	✓	✓	✗	✗
View Git repositories	✓	✓	✓	✓
Mark a Git repository as your favorite	✓	✓	✓	✓
Copy a Git repository's URL	✓	✓	✓	✓
Browse the Maven repository	✓	✓	✓	✓
Copy the Maven repository's URL	✓	✓	✓	✓
Browse a Docker registry and copy its URL	✓	✓	✓	✓

Graphs and Statistics Tab

Action	Organization Administrator	Owner	Member	Non-Member
View graphs and statistics	✓	✓	✓	✓

Team Tab

Action	Organization Administrator	Owner	Member	Non-Member
View project users	✓	✓	✓	✓
Export the users list	✓	✓	✓	✓
Add or remove a user	✓	✓	✗	✗
Change a user's project role	✓	✓	✗	✗

Git Actions

Action	Organization Administrator	Owner	Member	Non-Member
Create a hosted Git repository, add an external Git repository, or import a Git repository	✓	✓	✗	✗
Clone the Git repository	✓	✓	✓	✓
Push commits to the Git repository	✓	✓	✓	✗
Set the default branch of a Git repository	✓	✓	✗	✗
Set Git repository branch restrictions	✓	✓	✗	✗
View file contents and commits	✓	✓	✓	✓

Action	Organization Administrator	Owner	Member	Non-Member
Create or delete branches and tags	✓	✓	✓	✗
Compare files and revisions	✓	✓	✓	✓
Lock or protect a branch	✓	✓	✗	✗
Download archive of a branch or a tag	✓	✓	✓	✓
Add comments to commits	✓	✓	✓	✓
View graphical history of commits	✓	✓	✓	✓
Index a Git repository	✓	✓	✗	✗
Delete a Git repository	✓	✓	✗	✗

Note that a non-member can clone a Git repository and make commits to it, but can't push the commits to the remote Git repository.

Merge Requests Actions

Action	Organization Administrator	Owner	Member	Non-Member
Create a merge request	✓	✓	✓	✗
Add comments or reply to a comment	✓	✓	✓	✓
Subscribe to merge request email notifications	✓	✓	✓	✓

Note that all reviewers are automatically subscribed to merge request email notifications. Non-Members can also subscribe to email notifications. Open the merge request and click the **CC Me** button to subscribe.

When a merge request is created, all reviewers are assigned the REVIEWER role. The submitter of the request is assigned the REQUESTOR role. This table lists additional actions a REVIEWER or a REQUESTOR can perform.

Action	REQUESTOR	REVIEWER	Other Users
Add or remove reviewers	✓	✓	✗
Approve or Reject a merge request	✓	✓	✗
Merge branches or close a merge request	✓	✓	✗

A project Owner can always approve or reject a merge request, merge branches, or close a merge request, even if he or she is not assigned the REVIEWER role.

Maven Actions

Action	Organization Administrator	Owner	Member	Non-Member
Browse the Maven repository	✓	✓	✓	✗
Download artifacts from the Maven repository	✓	✓	✓	✗
Upload artifacts to the Maven repository	✓	✓	✓	✗
Search artifacts in the Maven repository	✓	✓	✓	✗
Configure the auto-cleanup of the Maven repository	✓	✓	✗	✗

Docker Actions

Action	Organization Administrator	Owner	Member	Non-Member
Link an external Docker registry	✓	✓	✗	✗
View external Docker registries, their repositories, and images	✓	✓	✓	✓
Download an image manifest of an external Docker registry repository	✓	✓	✓	✓

Action	Organization Administrator	Owner	Member	Non-Member
Delete an image manifest of an external Docker registry repository	✓	✓	✓	✓

Builds Actions

Action	Organization Administrator	Owner	Member	Non-Member
Set up connection to OCI Compute and OCI Object Storage	✓	✗	✗	✗
Create, configure, and manage Build VM templates	✓	✗	✗	✗
Add and manage Build VMs	✓	✗	✗	✗
Create a job	✓	✓	✓	✗
View job details	✓	✓	✓	✓
View or edit the job configuration	✓	✓	✓	✗
Run a build	✓	✓	✓	✗
Download artifacts	✓	✓	✓	✓
View logs, such as build console, audit logs, and Git polling log	✓	✓	✓	✗
Disable or delete a job	✓	✓	✓	✗
Create a pipeline	✓	✓	✓	✗
Configure a pipeline	✓	✓	✓	✗
View a pipeline's instances	✓	✓	✓	✓
Delete a pipeline	✓	✓	✓	✗

Releases Actions

Action	Organization Administrator	Owner	Member	Non-Member
Create a Release	✓	✓	✓	✗
Clone or Edit a Release	✓	✓	✓	✗
Delete a Release	✓	✓	✓	✗

Deployments Actions

Action	Organization Administrator	Owner	Member	Non-Member
Create a deployment configuration	✓	✓	✓	✗
Deploy or redeploy a configuration	✓	✓	✓	✗
View deploy logs	✓	✓	✓	✓

Environments Actions

Action	Organization Administrator	Owner	Member	Non-Member
Create and manage an environment	✓	✓	✓	✓
Add and manage an environment's instances	✓	✓	✓	✓

Issues Actions

Action	Organization Administrator	Owner	Member	Non-Member
Create an issue	✓	✓	✓	✓
Update an issue	✓	✓	✓	✓

Action	Organization Administrator	Owner	Member	Non-Member
Create and configure issue products	✓	✓	✗	✗
Create and configure issue tags	✓	✓	✗	✗
Create and configure issue custom fields	✓	✓	✗	✗

Boards Actions

Action	Organization Administrator	Owner	Member	Non-Member
Create a board	✓	✓	✓	✗
Use Scrum board	✓	✓	✓	✓
Use Kanban board	✓	✓	✓	✓
View burndown charts and sprint reports	✓	✓	✓	✓

When you create a board, you become the owner of the board. As the owner, you can perform various board and sprint related actions.

Action	Board Owner	Other Project Members	Non-Member
Add issues to a sprint	✓	✗	✗
Start a sprint	✓	✗	✗
Delete a sprint	✓	✗	✗
Configure the board	✓	✗	✗

Wiki Actions

Action	Organization Administrator	Owner	Member	Non-Member
Set the organization's default wiki markup language	✓	✗	✗	✗
Set the project's wiki markup language	✓	✓	✗	✗
Create a wiki	✓	✓	✓	✓
View a wiki page	✓	✓	✓	✓
Edit a wiki page	✓	✓	✓ By default	✗ By default
Delete a wiki page	✓	✓	✓ By default	✗ By default

The project Owner (or Member, if allowed) can grant the edit and delete rights of a wiki page to all users; or restrict the edit access to Members or Owners only.

Snippets Actions

Action	Organization Administrator	Owner	Member	Non-Member
Create a snippet	✓	✓	✓	✗
View snippet files	✓	✓	✓	✓
Insert a snippet file or copy a snippet file's text	✓	✓	✓	✓
Clone the snippet Git repository	✓	✓	✓	✓
Push the commits to the snippet Git repository	✓	✓	✓	✗
Download the archive of the snippet Git repository	✓	✓	✓	✓
Like a snippet	✓	✓	✓	✓
Add comments	✓	✓	✓	✓

Note that a non-Member can clone the snippet's repository and make commits, but can't push the commits to the Git repository.

The following table lists additional actions a snippet owner can perform.

Action	Snippet Owner	Other Project Members	Non-Member
Add, update, or remove snippet files	✓	✗	✗
Create snippet from selection	✓	✗	✗
Delete a snippet	✓	✗	✗

Set Up the Service

Like other Oracle Cloud services, you must create an instance of DevCS before you can start using it. You can create only one instance in an identity domain. After creating the instance, set up connections to Compute and Object Storage services on Oracle Cloud Infrastructure (OCI) or Oracle Cloud Infrastructure Classic (OCI Classic).

Create the DevCS Instance

You can create only one DevCS instance in an Oracle Cloud account. Before you attempt to create an instance make sure there's no existing DevCS instance in your account.

1. In a web browser, go to `https://cloud.oracle.com/sign-in`.

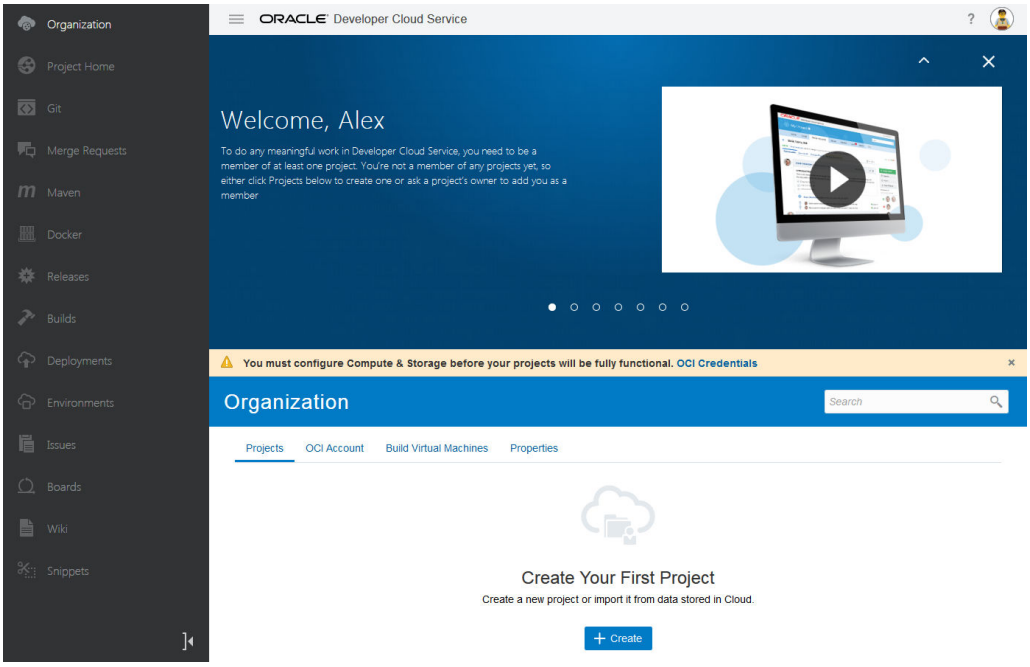
   To view the list of supported browsers, see `https://docs.cloud.oracle.com/iaas/Content/GSG/Tasks/signingin.htm#supported_browsers`.
2. On the Sign-In page, in **Account**, enter your account name, and click **Next**.
3. On the Oracle Cloud Account sign-in page, enter your Oracle Cloud account credentials, and click **Sign In**.

   The Oracle Cloud Console, also called as the OCI Console, opens.
4. On the OCI Console, click **Navigation Bar** in the top-left corner.
5. Under **More Oracle Cloud Services**, select **Platform Services**, and then select **Developer**.
6. In the **Instances** tab, click **Create Instance**.
7. On the Create New Instance page, in **Instance** enter a unique name. In **Description**, enter a description.

   The name helps you to identify the service instance.
8. Click **Next**.
9. On the Service Details page, click **Next**.
10. On the Confirmation page, click **Create**.

Once the service instance is created, you can open the service console by clicking **Action**, and then selecting **Access Service Instance**.

The DevCS **Organization** page opens. Click the **OCI Credentials** link or the **OCI Account** tab to configure OCI connections before you create a project.



Connect to OCI or OCI Classic

You need to connect to the Oracle Cloud Infrastructure Compute (OCI Compute) or Oracle Cloud Infrastructure Compute Classic (OCI Compute Classic) because they provide the virtual machines (VMs) on which DevCS runs its builds.

You need to connect to Oracle Cloud Infrastructure Object Storage (OCI Object Storage) or Oracle Cloud Infrastructure Object Storage Classic (OCI Object Storage Classic) because they are used to store build and Maven artifacts for DevCS projects. Note that this feature is not available to DevCS users in the Traditional identity domain, who run builds using shared build executors, not virtual machines (VMs).

If you're an OCI user, set up connections to OCI Compute and OCI Object Storage. DevCS runs its builds on OCI Compute VMs, and stores build and Maven artifacts on the OCI Object Storage buckets.

If you're an OCI Classic user, set up connections to OCI Compute Classic and OCI Object Storage Classic. DevCS runs its builds on OCI Compute Classic VMs, and stores build and Maven artifacts on the OCI Object Storage Classic containers.

Set Up the OCI Connection

Before you set up the connections, set up your OCI account to host DevCS resources. They allow DevCS to manage necessary resources, such as VMs for your builds and storage buckets for your project data.

To set up the OCI account, open the OCI console and create a compartment, a group and a user to access the compartment, and a policy that defines access to the compartment.

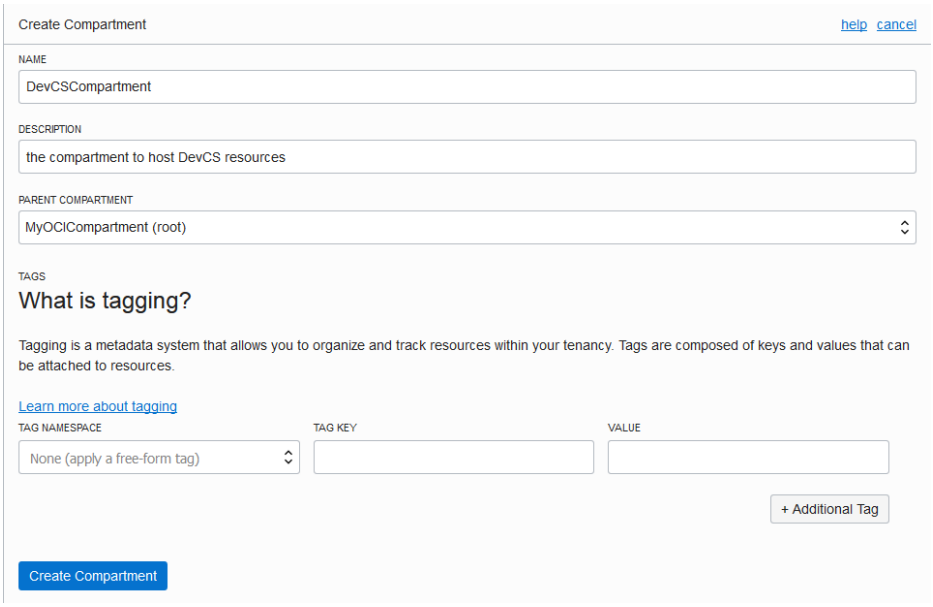
You can use the root compartment and the tenancy user that was created when the OCI account was created, but it's recommended to create a dedicated compartment to host DevCS resources. This allows you to organize DevCS resources better as they aren't mixed with the other resources of your tenancy. You can also restrict users and control read-write access to the compartment without affecting other resources. To learn more about compartments, see Understanding Compartments.

After setting up your OCI account, share the compartment's and the created user's details with the DevCS Organization Administrator to set up the OCI connection in DevCS.

Set Up the OCI Account

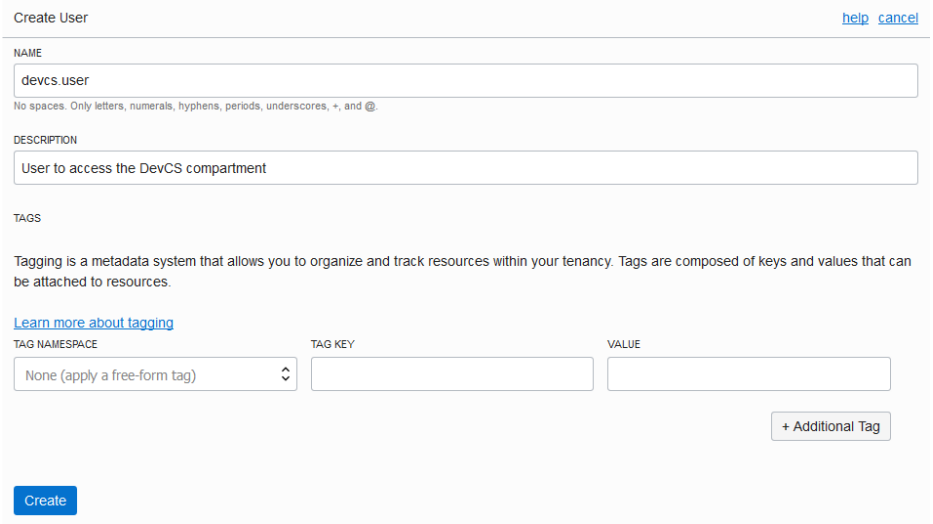
1. On the OCI Console, click ☰ in the top-left corner.
2. Under **Governance and Administration**, select **Identity**, and then select **Compartments**.
3. On the Compartments page, create a compartment to host DevCS resources.
   a. To create the compartment in the tenancy (root compartment), click **Create Compartment**.
   b. In the Create Compartment dialog box, fill in the fields, and click **Create Compartment**.

      Here's an example:



      To learn more about compartments, see Working with Compartments.
4. Create a user to access the DevCS compartment.
   a. In the left navigation bar, under **Governance and Administration**, go to **Identity** and click **Users**.
   b. Click **Create User**.
   c. In the Create User dialog box, fill in the fields, and click **Create**.

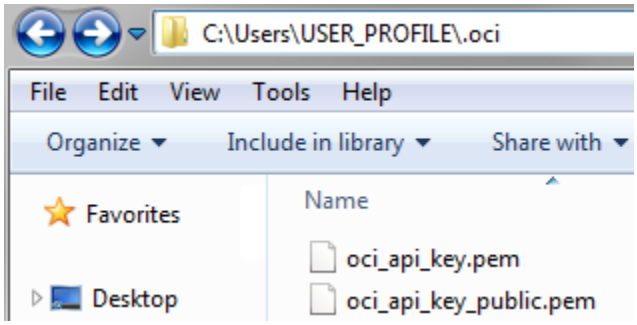
      Here's an example:



To learn more about OCI users, see Working with Users.

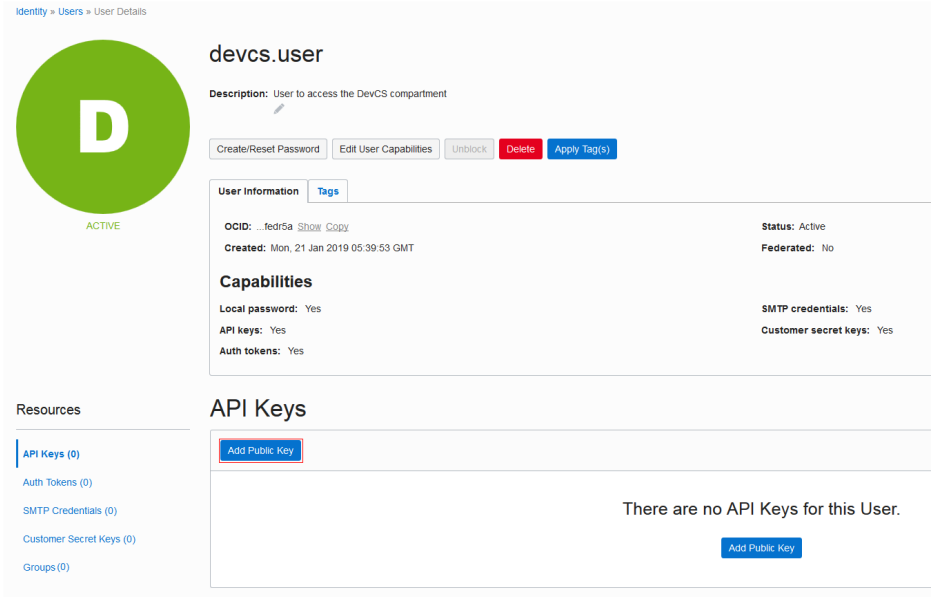
5. On your computer, generate a private-public key pair in the PEM format.

   To find out how to generate a private-public key pair in the PEM format, see How to Generate an API Signing Key.
   Here's an example of private-public key files on a Windows computer:



6. Upload the public key to the user's details page.
   a. Open the public key file in a text editor and copy its contents.
   b. In the left navigation bar of the OCI dashboard, click under **Governance and Administration**, go to **Identity** and click **Users**.
   c. Click the user's name created in Step 3.
   d. In the User Details page, click **Add Public Key**.

      Here's an example:



e. In the Add Public Key dialog box, paste the contents of the public key file, then click **Add**.

To learn more about uploading keys, see How to Upload the Public Key.

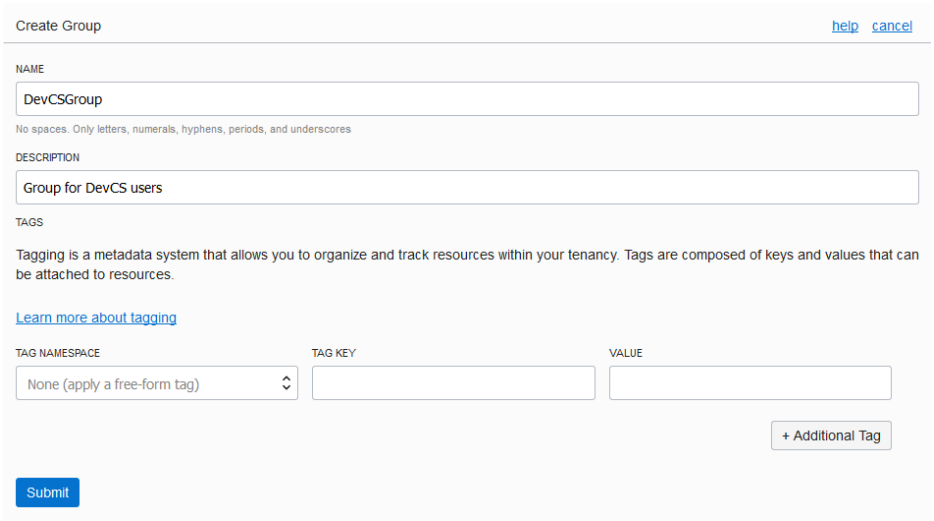
7. On the Groups page, create a group for the user who can access the DevCS compartment and add the user to the group.

   a. In the left navigation bar, under **Governance and Administration**, go to **Identity** and click **Groups**.

   b. Click **Create Group**.

   c. In the Create Group dialog, fill in the fields and click **Submit**.

      Here's an example:

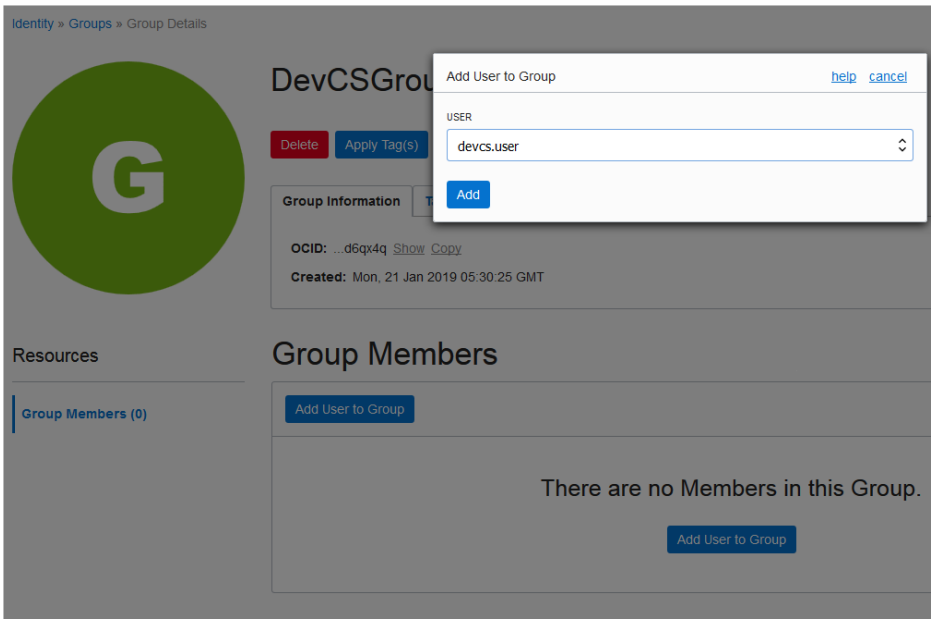


   d. On the Groups page, click the group's name.

   e. On the Group Details page, click **Add User to Group**.

f. In the Add User to Group dialog box, select the user created in Step 3, and click **Add**.

Here's an example:



To learn more about groups, see Working with Groups.

8. In the root compartment, not the DevCS compartment, create a policy to allow the group created in step 6 to access the DevCS compartment.

   a. In the left navigation bar, under **Governance and Administration**, go to **Identity** and click **Policies**.

   b. On the left side of the Policies page, from the **Compartment** list, select the root compartment.

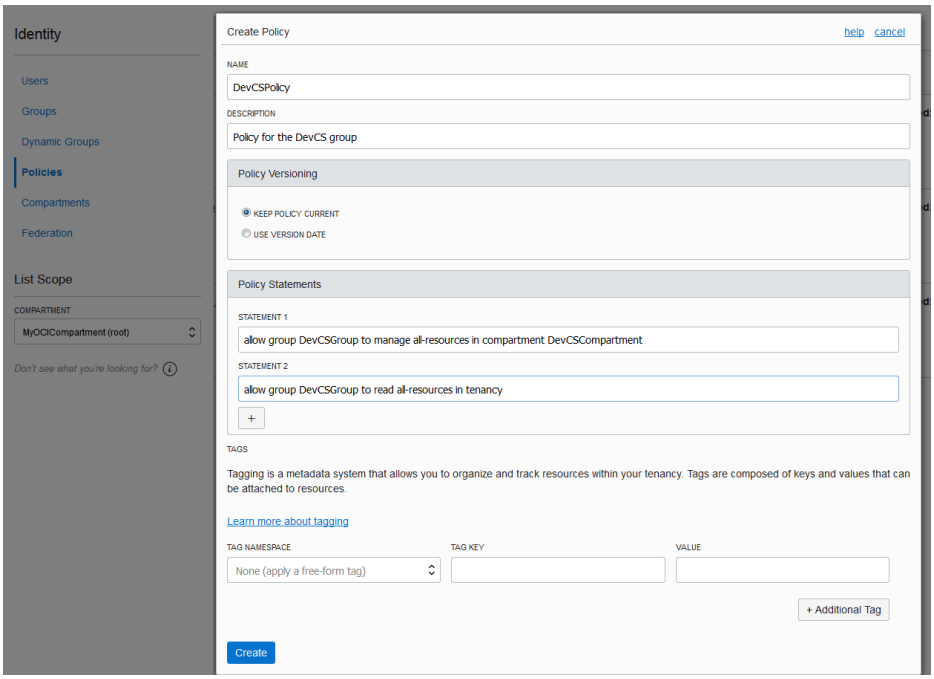
   c. Click **Create Policy**.

   d. In **Name** and **Description**, enter a unique name and a description.

   e. In **Policy Statements**, add these statements.

      - `allow group <group-name> to manage all-resources in compartment <compartment-name>`
        This grants all permissions to the DevCS group users to manage all resources within the DevCS compartment.
      - `allow group <group-name> to read all-resources in tenancy`
        This grants read permissions to the DevCS group so that its users can read—but not use, create or modify—all resources inside and outside the DevCS compartment. The group users can't use, create, or modify the resources. This statement is optional.

      Here's an example:



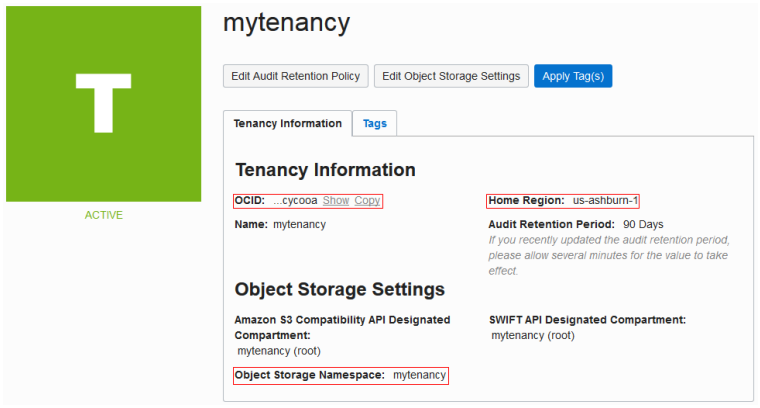
f. Click **Create**.

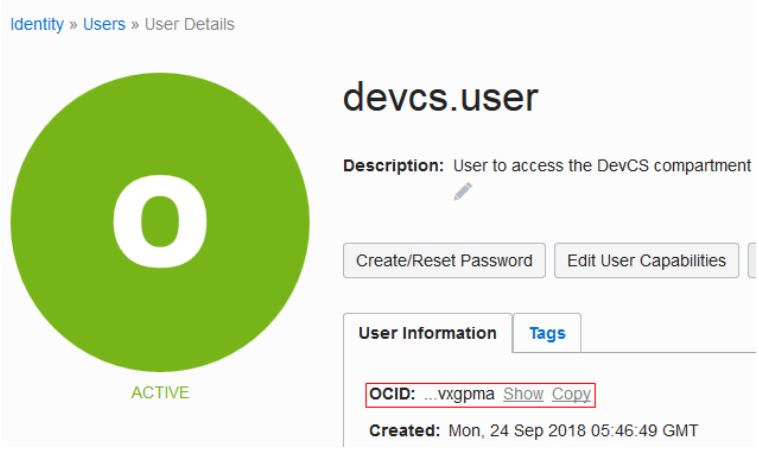
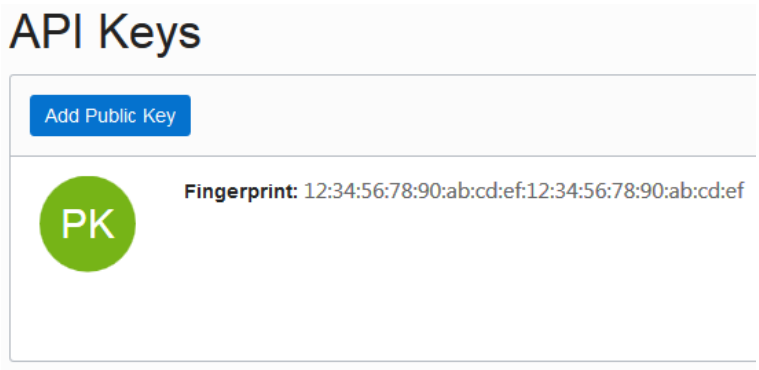
To learn more about policies, see Working with Policies.

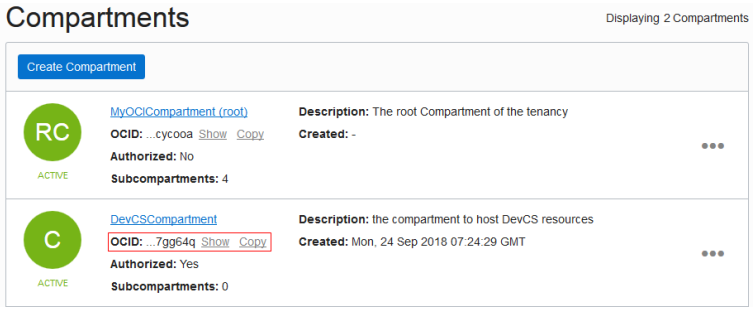
Get the Required OCI Input Values

Every Oracle Cloud Infrastructure resource has an Oracle-assigned unique ID called an Oracle Cloud Identifier (OCID). To connect to OCI, you need the account's tenancy OCID, home region, the compartment's OCID that hosts DevCS resources, and the OCID and fingerprint of the user who can access the DevCS compartment. To connect to OCI Object Storage, you need the Storage namespace. You can get these values from the OCI Console pages.

This table describes how to get the OCI input values required for the connection.

To get these values ...	Do this:
Tenancy OCID, Home Region, and Storage Namespace	On the OCI console, from the left navigation bar, select **Administration** > **Tenancy Details**. The **Tenancy Information** tab displays the tenancy OCID in **OCID**, home region in **Home Region**, and the storage namespace in **Object Storage Namespace**. Here's an example: 

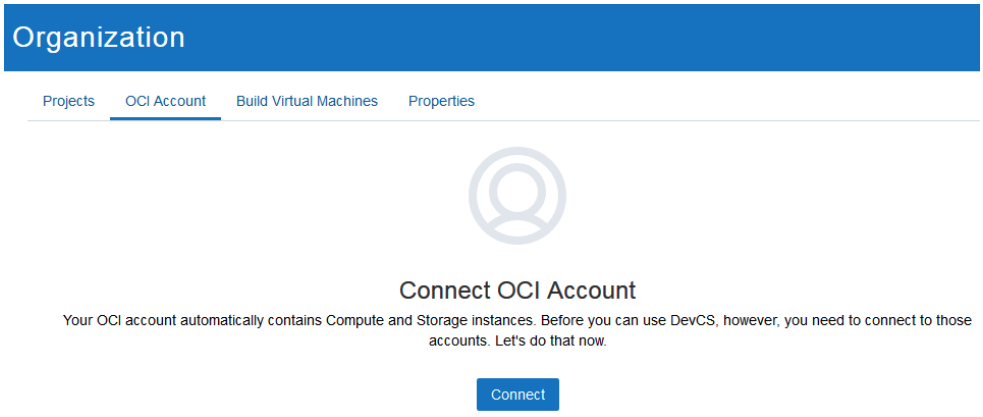
To get these values ...	Do this:
User OCID and Fingerprint	On the OCI console, from the left navigation bar, under **Governance and Administration**, select **Identity** > **Users**. The **User Information** tab displays the user OCID in **OCID**. Click the **Copy** link to copy it to the clipboard. Here's an example of `devcs.user`:  To get the fingerprint of the public key associated with your OCI account, scroll down to the **API Keys** section and copy the fingerprint value. 

To get these values ...	Do this:
Compartment OCID	On the OCI console, from the left navigation bar, select **Identity** > **Compartments**. The Compartments list displays the compartments with the compartment OCID in the **OCID** field. Click the **Copy** link to copy it to the clipboard. Here's an example: 

Set Up the OCI Connection in DevCS

To connect to OCI, get the DevCS compartment's details, user details, and the required OCID values. Then, create an OCI connection from DevCS. If you're not the OCI administrator, get the details from the OCI administrator.

1. In the navigation bar, click **Organization**  .
2. Click the **OCI Account** tab.
3. Click **Connect**.

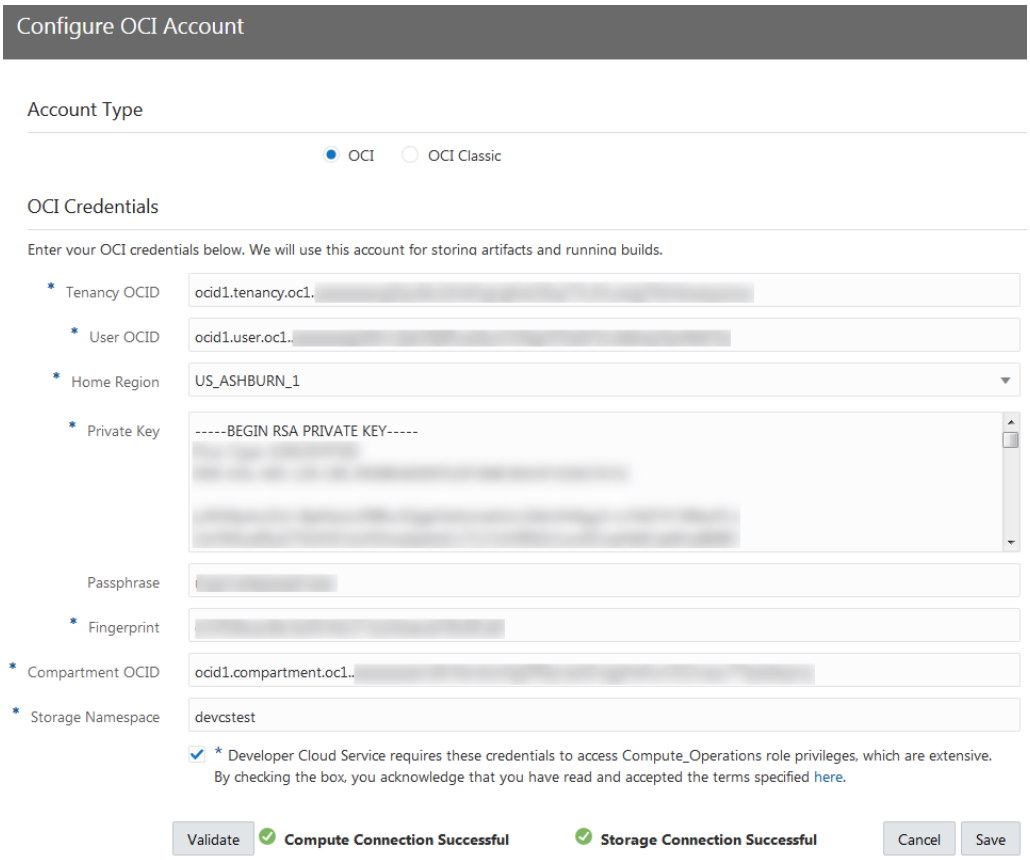


4. In **Account Type**, select **OCI**.
5. In **Tenancy OCID**, enter the tenancy's OCID copied from the Tenancy Details page.

6. In **User OCID**, enter the OCID of the user who can access the DevCS compartment.
7. In **Home Region**, select the home region of the OCI account.
8. In **Private Key**, enter the private key of the user who can access the DevCS compartment.

   The private key file was generated and saved on your computer when you created the private-public key pair in the PEM format. See Step 4 in Set Up the OCI Account.
9. In **Passphrase**, enter the passphrase used to encrypt the private key. If no passphrase was used, leave the field empty.
10. In **Fingerprint**, enter the fingerprint value of the private-public key pair.
11. In **Compartment OCID**, enter the compartment's OCID copied from the Compartments page.
12. In **Storage Namespace**, enter the storage namespace copied from the Tenancy Details page.
13. To agree to terms and conditions, select the terms and conditions check box.
14. To validate the connection details, click **Validate**.
15. After validating the connection details, click **Save**.

Here's an example of an **OCI Account** tab filed with required OCI details.



Set Up the OCI Classic Connection

To connect to OCI Classic, you need the credentials of a user with the **Compute.Compute_Operations** and **Storage.Storage_Administrator** identity domain roles along with the service ID and Authorization URL of OCI Object Storage Classic.

The **Compute.Compute_Operations** role enables you to create, update, and delete VMs on OCI Compute Classic. The **Storage.Storage_Administrator** role enables you to store artifacts on OCI Object Storage Classic.

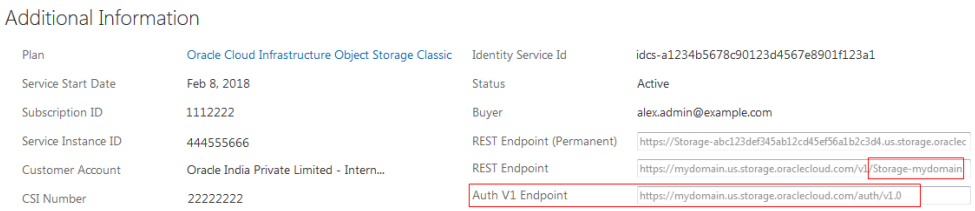
Before you create the OCI Compute Classic connection, you must check the Compute_Operations Role: Terms of Use and get the Service ID and the authorization URL of OCI Object Storage Classic:

Get OCI Object Storage Classic Input Values

1. Open the Oracle Cloud Dashboard.
2. In the **Storage Classic** tile, click **Action**, and select **View Details**.

   If the Storage Classic tile isn't visible, click **Customize Dashboard**. Under Infrastructure, find Storage Classic, click **Show**, and then close the Customize Dashboard window.
3. On the Service Details page, in the **Additional Information** section of the **Overview** tab, note the values of the **Auth V1 Endpoint** URL and the last part of the **REST Endpoint** URL.

   If you're using an Oracle Cloud traditional account, the fields shown in this graphic might differ from the fields on your Service Details page.



Create an OCI Classic Connection from DevCS

After you have the required values, create an OCI Classic connection from DevCS.

You must be the **Organization Administrator** to create the connection.

1. Sign in to DevCS.
2. In the navigation bar, click **Organization**.
3. Click the **OCI Account** tab.
4. To create a connection, click **Connect**. To edit the connection details, click **Edit**.
5. In **Account Type**, select **OCI Classic**.
6. In the OCI Object Storage Classic section, fill in the required details.

a. In **Service ID**, enter the value copied from the last part of the **REST Endpoint** URL field of the Service Details page.

   For example, if the value of **REST Endpoint** URL is `https://demo12345678.storage.oraclecloud.com/v1/`**`Storage-demo12345678`**, then enter `Storage-demo12345678`.

b. In **Username** and **Password**, enter the credentials of the user assigned the **Storage.Storage_Administrator** identity domain role.

c. In **Authorization URL**, enter the URL copied from the **Auth V1 Endpoint** field of the Service Details page.

   Example: `http://storagetria01234-usoracletria12345.storage.oraclecloud.com/auth/v1.0`.

d. Click **Validate**.

7. In the OCI Compute Classic section, fill in the required details.

   a. In **Username** and **Password**, enter the username and password of the user who's assigned the **Compute.Compute_Operations** identity domain role.

   b. To agree to terms and conditions, select the terms and conditions check box.

   c. Click **Validate**.

8. Click **Save**.

Compute_Operations Role: Terms of Use

Here are some special legal terms and guidance that apply to the usage of the `Compute_Operations` role for DevCS.

In addition to these DevCS terms, you should follow security best practices in maintaining the security of the username and password.

- You must create a dedicated username and password for use by DevCS. When creating a username, avoid including personal names or personal information (like birthdays). Your password should always be complex and impossible to guess.
- You understand that a user with the `Compute_Operations` role can view, create, update and delete OCI Compute Classic resources such as VM instances, storage volumes, security rules, and security IP lists. Your failure to maintain security best practices to secure the username and password of the user with the `Compute_Operations` role may create a high risk for you and your organization.
- You should assign the `Compute_Operations` role privileges only to the username created for DevCS.
- Notwithstanding DevCS terms, you acknowledge that Oracle isn't responsible or liable for any action you take in accessing or creating access to the DevCS or OCI Compute Classic.

Add Users to the Identity Domain

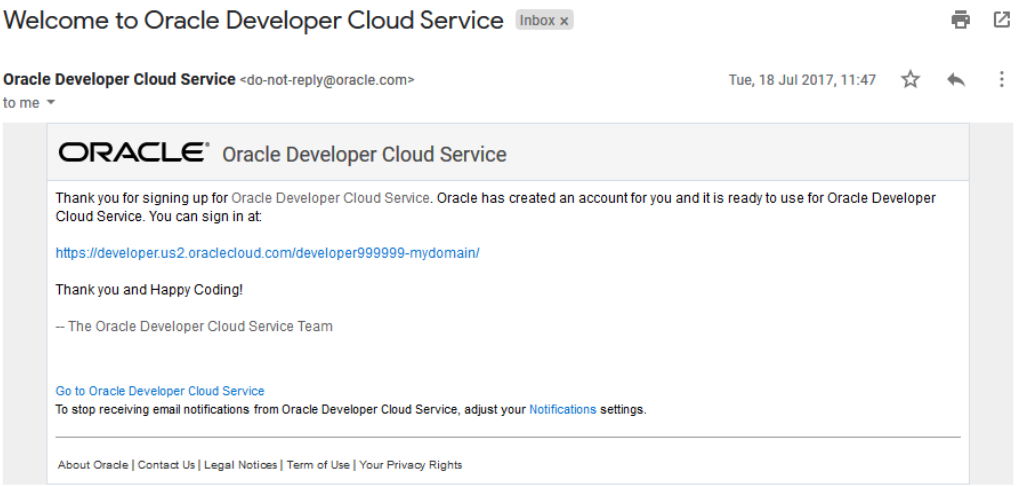
To add users to your organization and projects, make sure they are added to your identity domain and assigned appropriate identity domain roles. An organization is the top-most entity in the project structure of DevCS. All users of the identity domain are available in your organization and then can be added to projects.

To add a user to the identity domain, invite the user to join Oracle Cloud. See Add Users and Assign Roles in *Getting Started with Oracle Cloud*. You must be assigned the Identity Domain Administrator (`TenantAdminGroup`) role to send an invite and assign the identity domain roles.

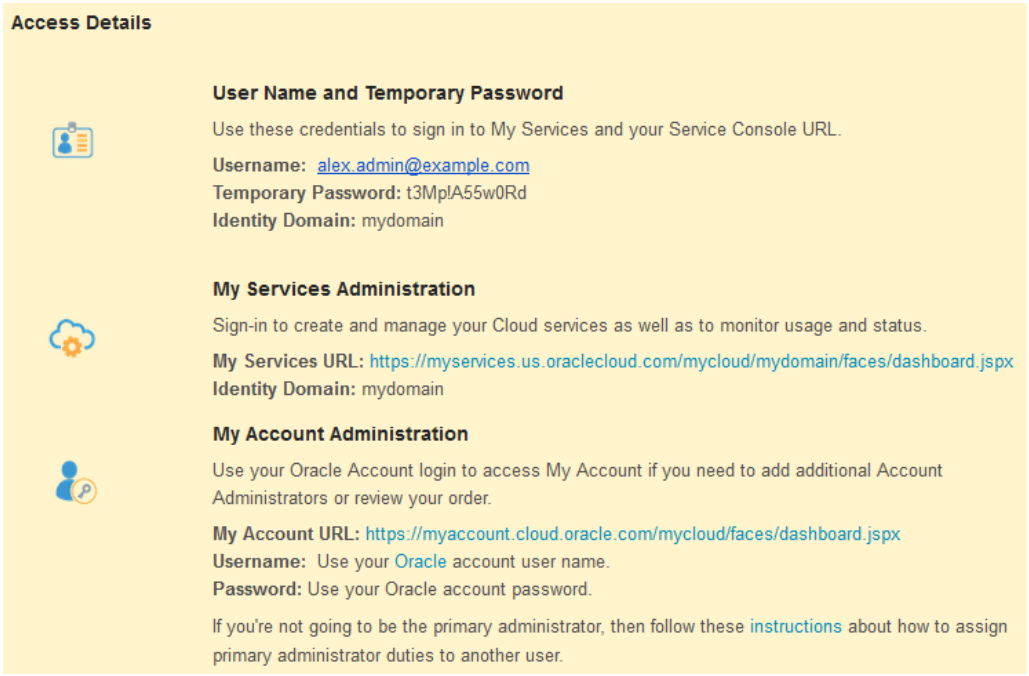
Access Oracle Developer Cloud Service

You can access Oracle Developer Cloud Service (DevCS) from any Oracle Cloud-supported web browser and from supported IDEs.

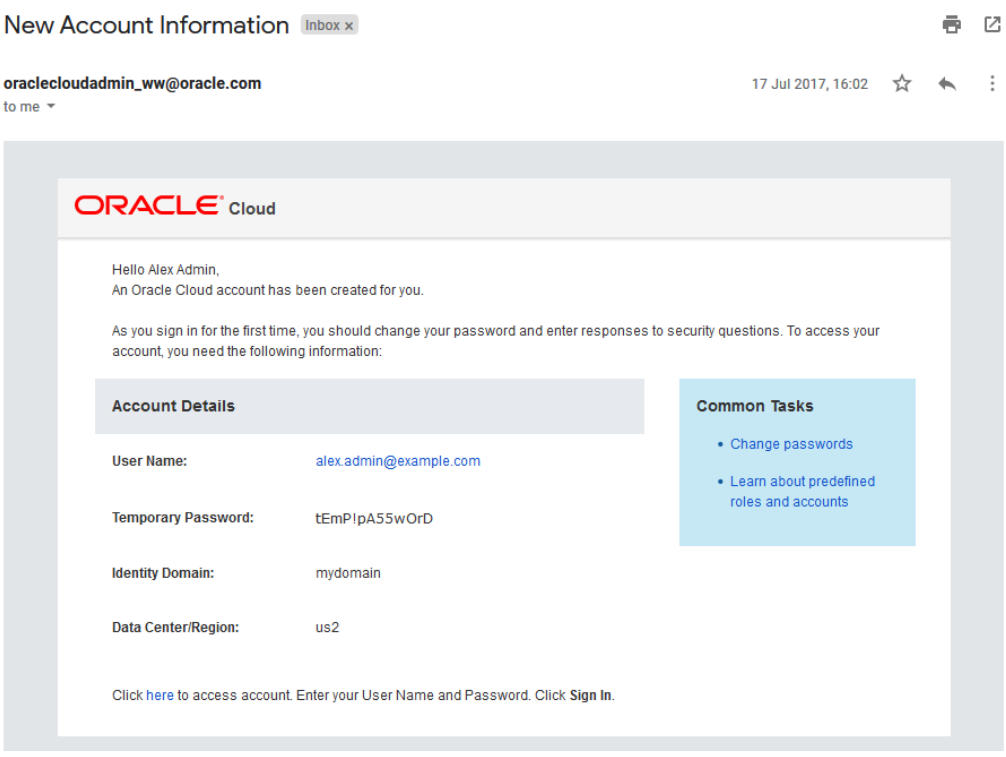
To access DevCS, you need the service URL, plus your identity domain name, username, and password. If you're a new user, you can sign in from the Oracle Cloud home page. If you're a returning user, you can find the service URL from the email with the subject **Welcome to Oracle Developer Cloud Service** or **Verify your Oracle Developer Cloud Service**.



Your identity domain name and credentials are available in the email with the subject **Welcome to Oracle Cloud**.



If you're invited to Oracle Cloud, you can find those details in the email with the subject **New Account Information**.



Access DevCS

You can sign in and open DevCS from the Oracle Cloud home page.

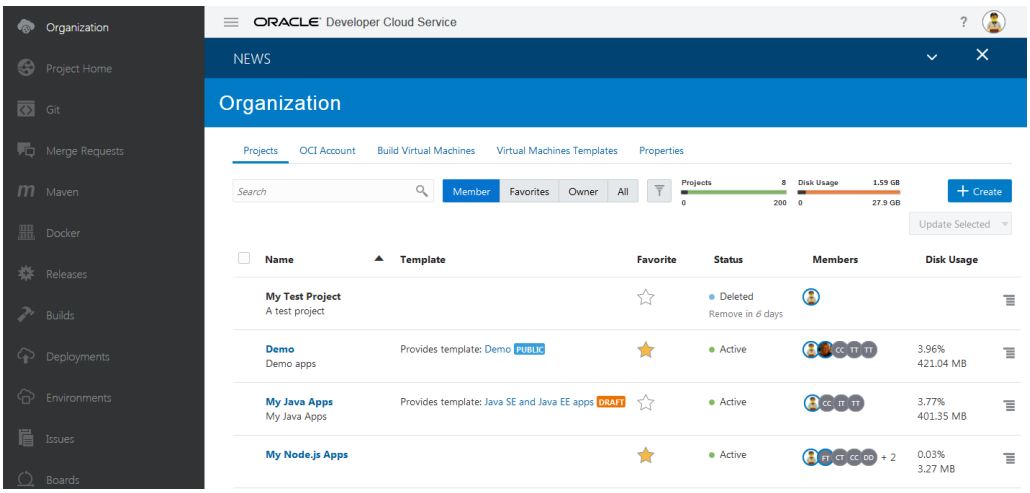
To view the list of supported browsers, see `https://docs.cloud.oracle.com/iaas/Content/GSG/Tasks/signingin.htm#supported_browsers`.

1. In a web browser, go to `https://cloud.oracle.com`, and click **Sign In**.
2. On the Sign-In page, in **Account**, enter your account or tenant name, and click **Next**.
3. On the Oracle Cloud Account sign-in page, enter your Oracle Cloud account credentials, and click **Sign In**.
4. On the OCI Console, click ☰ in the top-left corner.
5. Under **More Oracle Cloud Services**, select **Platform Services**, and then select **Developer**.
6. On the **Instances** tab, click **Manage this instance** ☰ and select **Access service instance**.

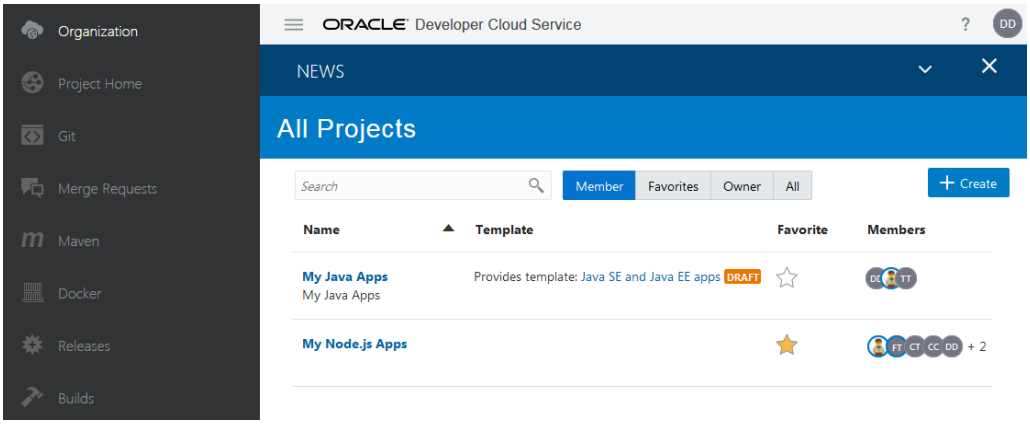
If you're signing in to DevCS for the first time, you should have received an email with the subject **Verify your DevCS**. Open the email and click the URL link in the email body to verify your email. This is required to receive email notifications from the service.

After your email address is verified, you'll receive another email with the subject **Welcome to DevCS**. This email contains the DevCS URL that you can bookmark.

After you sign in to DevCS, you'll see the **Organization** page that displays all the projects you're a member of, as well as your favorite projects, the projects you own, and all the shared projects in your organization.
If you're assigned the `DEVELOPER_ADMINISTRATOR` (Developer Service Administrator) identity domain role, you can manage all projects, OCI connections, virtual machines, and the properties of the organization. To open a project, click its name. You can't open a project if you're not its member.



If you're assigned the DEVELOPER_USER (Developer Service User) identity domain role, you can create a project or click its name to open it.



Access from IDEs

Besides the Oracle Cloud web interface, you can also connect to DevCS from Eclipse IDE, NetBeans IDE, and JDeveloper.

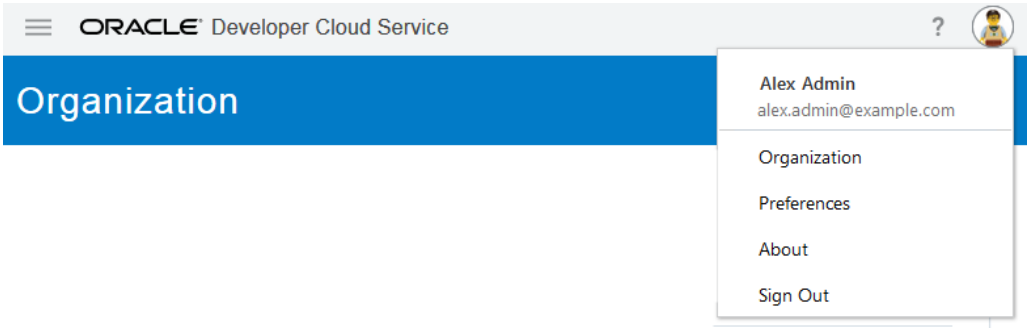
For more information, see:

- Eclipse IDE
- NetBeans IDE
- JDeveloper

Set Your User Preferences

One of the first steps in getting started with Oracle Developer Cloud Service (DevCS) is to set up or join projects. Before doing so, however, you may want to change your display name, email address, and enable or disable email notifications.

You can set your preferences from the User Preferences page. To get to this page, click the user avatar, and select **Preferences**.



Update Your Display Name

By default, DevCS displays your Oracle Cloud account name as your display name across all pages. If you want to change it, you can do so from the User Preference page's Profile tab.

1. On the User Preferences page, click the **Profile** tab.
2. In **First Name** and **Last Name**, update your name.

   The name is saved when the focus moves out of the field.
3. To the left of the **User Preferences** title, click **Close** to return to the last opened page.

Update Your Email Address

By default, DevCS displays your Oracle Cloud email address across all pages and sends email notifications, such as merge request notifications and issue notifications, to this email address.

If you want DevCS email notifications sent to another email address, you can change it on the User Preference page's Profile tab. If you're using an email address as your Oracle Cloud login username, after changing the email address, your original Oracle Cloud email address continues to be your login username.

After you provide another email address, you'll receive a verification email. If you don't verify your email address, you won't receive any email notifications; however, you can continue to use DevCS.

1. On the User Preferences page, click the **Profile** tab.
2. In the **Email Address** field, enter your new email address.

   The email address is saved when the focus moves out of the field.
3. Click the **Re-send email** button.
4. In the email that you receive, click the confirmation link to confirm the email address.

   After the verification, you're redirected to the service page.
5. Open the **Profile** tab again and verify that the **Email Address** field displays the **Verified** label.
6. To the left of the **User Preferences** title, click **Close** to return to the last opened page.

Add Your Avatar Picture

DevCS displays your Gravatar picture as the avatar picture. If you don't have a picture set in Gravatar or don't have a Gravatar account, DevCS displays your name initials as the avatar picture.

To find out more about Gravatar, see `https://gravatar.com/`.

1. Open `http://en.gravatar.com/` in your browser.
2. Click **Create Your Own Gravatar**.

3. Follow the on-screen instructions, enter the required details, and sign up.

   Create your account with the same email address that you used to subscribe to Oracle Cloud.

4. After activating your account, sign in to Gravatar.
5. Upload the avatar picture to your Gravatar account.

The picture uploaded to your Gravatar account is automatically displayed as your avatar picture in DevCS.

Configure Your Global Email Notifications

You can configure your preferences to receive email notifications when a component (such as an issue or a branch of a Git repository) that you're subscribed to is updated. Your preferences apply to all projects where you are a member.

Note that your email address must be verified to receive email notifications.

1. On the User Preferences page, click the **Notifications** tab.
2. Select or deselect the **Notify Me Of** check boxes.

   Some check boxes are selected by default. For a selected component, its notifications from all projects of the organization where you're a member are enabled. You must subscribe or set up a watch on the component to get notifications about its updates.

Select this check box ...	To receive email notifications about:
Issue updates, attachments and comments	Issues you're assigned to, or you're watching.
Merge Request updates and comments	Merge requests where you're a reviewer, or you're watching.
New features, tips, and events	New features, tips, and events from the DevCS team.
Service and system maintenance updates	Service and system maintenance updates from the DevCS team.
Build activities	Jobs you're watching.
SCM/Push Activities	Git repository branches you're watching.
Wiki page updates and comments	Wiki pages you're watching.
Project Updates	User updates when you or a user is added to or removed from a project, or the project role is changed.
Include my Own Updates	Your own changes. If you don't select the check box, you won't receive email notifications for issues, merge requests, and Git updates that you initiated or created even though the **Issues updates, attachments and comments**, the **Merge Request updates and comments**, or the **SCM/Push Activities** check boxes are selected.

3. To the left of the **User Preferences** title, click **Close** to return to the last opened page.

Upload Your Public SSH Key

If you want to connect to a Git repository using SSH, you must first generate a private-public RSA SSH key pair and upload the public key to DevCS. If you use multiple computers to access Git repositories, generate an SSH key pair from each computer and upload its public key.

Generate an SSH Key

To generate an RSA SSH key pair, you can use any SSH client, including the Git CLI.

These steps assume you're using Git CLI to generate the SSH keys.

1. Open the Git CLI.
2. On the command prompt, enter `ssh-keygen -t rsa`.

   To generate a larger key, enter `ssh-keygen -t rsa -b 4096`. If you're using a macOS version 10.13.6 (or higher), enter `ssh-keygen -m PEM -t rsa`.
3. When prompted, enter a file name for the key and press **Enter**.

   If you don't want to specify a file name, leave the name blank and press **Enter**. By default, the key pair files are saved as `id_rsa.pub` and `id_rsa` in the `.ssh` sub-directory under the Git `HOME` directory. For example, on Windows, the files are saved in `C:\Users\<USER_PROFILE>\.ssh\`.
4. Enter a passphrase and press **Enter**. If you don't want to specify a passphrase, leave it blank and press **Enter**.

   When prompted to confirm the passphrase, enter the same passphrase. If you didn't specify a passphrase earlier, leave it blank and press **Enter**.

Git CLI always looks at the `C:\Users\<USER_PROFILE>\.ssh\` directory to access the private key. If you aren't using the Git CLI, you may need to configure your Git client or your IDE to access the private SSH key. Check the Git client's or IDE's documentation to find out how to do that.

Add the Public SSH Key to Your DevCS Account

After generating an SSH private-public key pair, add the public key to your User Preferences page's **Authentication** tab.

1. On the computer where you generated the SSH key pair, navigate to the directory where the public key is saved.
2. Open the public key file in a text editor, select the contents, and copy them to the clipboard.
3. In DevCS, click the user avatar, and select **Preferences**.
4. Click the **Authentication** tab.
5. Click **Add Key**.
6. In the New SSH Key dialog box, enter a unique name and paste the SSH key that you copied in Step 2.

7. Click **Create**.
8. To the left of the **User Preferences** title, click **Close** to return to the last opened page.

See the News Banner

A banner displaying the latest news from the DevCS team is displayed in the page header of the **Organization** and the **Project Home** page.

To navigate between news pages, click the navigation buttons. Click the See More link to lean more about the open news. To expand or collapse the banner, use the **Expand** or **Collapse** icons. To close the banner, click **Close**.

If the banner isn't visible, follow these steps to enable it.

1. On the User Preferences page, click the **General** tab.
2. Select the **Show News Banner on Organization and Project Home** check box.

Set Up IDEs and Git Clients

You can use IDEs such as Eclipse IDE, NetBeans IDE, and Oracle JDeveloper to access projects, as well as your project's Git repositories, issues, and builds. Before using an IDE, you must install and configure the required plugins.

You can also use any Git client, such as the Git command-line interface (CLI), to access Git repositories from your computer. You can't access projects, issues, and builds from a Git client.

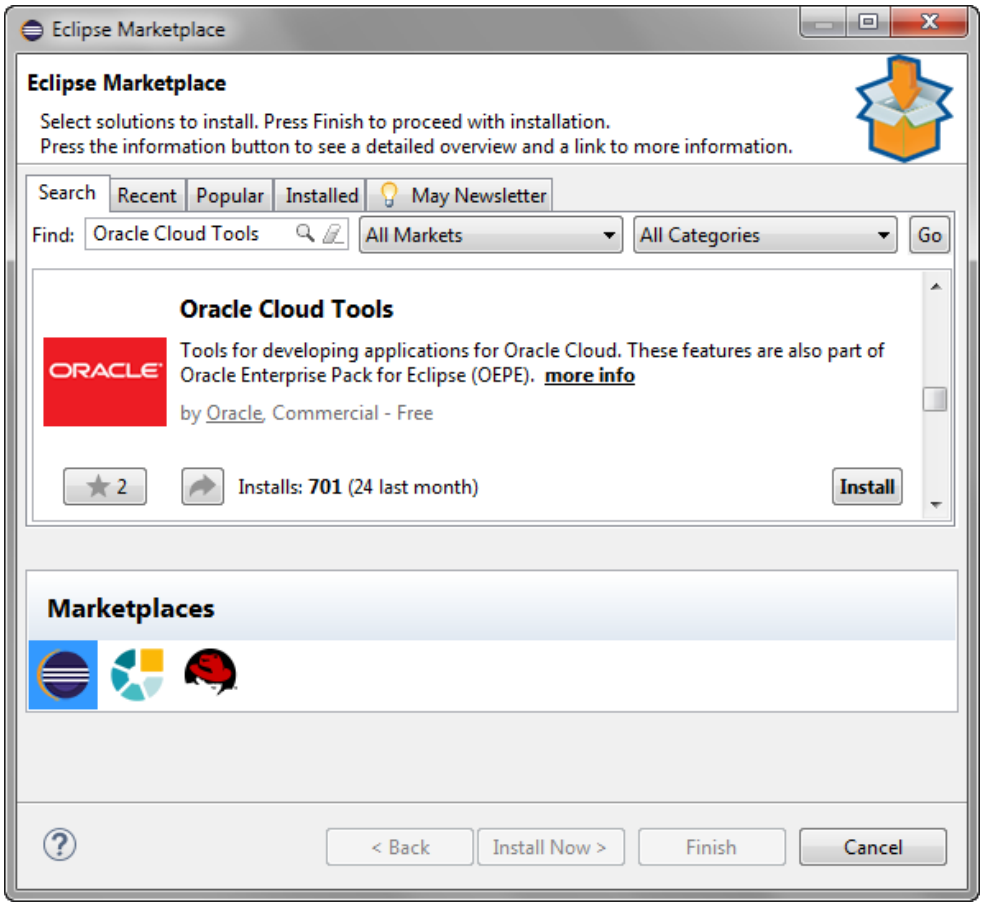
Eclipse IDE

To access DevCS projects and Git repositories from the Eclipse IDE, install the plugins required by Oracle Cloud and the Git from the Eclipse marketplace. After you install the plugins, you can use the Eclipse IDE to develop applications for Oracle Cloud.

If you're using the Oracle Enterprise Pack for Eclipse (OEPE) IDE, the Oracle Cloud and the Git plugins are installed by default.

Eclipse IDE uses EGit to access and manage Git repositories. For more information about EGit and the Git actions you can perform, see `http://eclipse.org/egit/` and the EGit documentation at `http://wiki.eclipse.org/EGit/User_Guide`.

1. Open the Eclipse IDE.
2. From the **Help** menu, select **Eclipse Marketplace**.
3. In the **Search** tab, search for `Oracle Cloud Tools`, and click **Go**.
4. In the search result, find the **Oracle Cloud Tools** plugin and click **Install**.



5. Click **< Install More**.
6. In the **Search** tab, search for `EGit`, and click **Go**.
7. In the search result, find the **EGit - Git Integration for Eclipse** plugin, and click **Install**.
8. Click **Install Now**.
9. On the Confirm Selected Features page of the wizard, make sure that the check boxes of **Oracle Cloud Tools** and **EGit** are selected, and click **Confirm**.
10. In the Review Licenses page, select the **I accept the terms of the license agreements** option to agree to the licensing terms and click **Finish**.
11. Wait for the software to install.
12. If prompted, restart the IDE.

After installing the Git plugin, you should set the username and email properties. DevCS pages display your username and email address as the committer's name and email ID.

1. From the **Window** menu of the Eclipse IDE, select **Preferences**.
2. In the Preferences dialog box, search for `Git`.
3. In the **Team** > **Git** > **Configuration** page, set the properties.

NetBeans IDE

To access your projects and Git repositories from the NetBeans IDE, install the plugins required by DevCS and Git.

1. Open the NetBeans IDE.
2. From the **Tools** menu, select **Plugins**.
3. In the Plugins dialog box, click the **Available Plugins** tab.
4. Search for `Oracle Developer Cloud Service` and `Git`, and select their **Install** check boxes.

   If a plugin is pre-installed, check the **Installed** tab and make sure the plugin is active.
5. Click **Install**.
6. Read the license agreement and select the **I accept the terms in all of the license agreements** check box.
7. Click **Install**.

After installing the Git plugin, you should set the username and email properties. DevCS pages display your username and email address as the committer's name and email ID.

1. From the **Team** menu, select **Git** > **Open Global Configuration**.
2. In the `.gitconfig` file, set the properties, and save the file.

JDeveloper

Oracle JDeveloper comes pre-installed with the plugins required by DevCS and Git. No additional configuration is required.

Git Command-Line Interface

If you can't use an IDE to access and update files of a Git repository, you can install and configure a Git client on your computer. The Git command line-interface (CLI) is the most popular Git client.

Here are the steps to download, install, and configure the Git CLI.

1. Download and install the Git CLI.

   On Windows, use the Git Bash CLI to access project Git repositories. You can download Git Bash (version 1.8.x or later) from `http://git-scm.com/downloads`.

   On Linux and Unix, install Git using the preferred package manager. You can download Git for Linux and Unix from `http://git-scm.com/download/linux`.
2. Configure variables to set up your name and email address. DevCS pages display your username and email address as the committer's name and email ID.
   - To configure your user name, set the `user.name` variable.

     Example: On the Git CLI command prompt, enter `git config --global user.name "John Doe"`

- To configure your email address, set the `user.email` variable.

  Example: `git config --global user.email "johndoe@example.com"`

- To configure the proxy server or disable SSL, set `http.sslVerify` and `http.proxy` variables.

  Examples:

  ```
  git config --global http.sslVerify false
  ```

  ```
  git config --global http.proxy http://www.testproxyserver.com:80/
  ```

> **Tip:**
>
> To find out the value of a variable, use the `git config <variable>` command.
>
> Example:
>
> ```
> git config user.name
> ```

Use Projects

After signing in to Oracle Developer Cloud Service (DevCS), you can create a project, open a shared project, or open a project you are a member of.

Create a Project

From the Organization page, you can create an empty project, a project with a Git repository, import a project from an OCI Object Storage or an OCI Object Storage Classic container, or create a project from a template.

Empty Project

If you haven't decided which applications you want to upload, or want to start afresh, create an empty project. An empty project is a project with no pre-configured Git repository or any other artifact.

1. On the Organization page, click **+ Create Project**.
2. On the Project Details page of the New Project wizard, in **Name** and **Description**, enter a unique project name and a project description.
3. In **Security**, select the project's privacy.
4. Click **Next**.
5. On the Template page, select **Empty Project**, and click **Next**.
6. On the Project Properties page, from **Wiki Markup**, select the project's wiki markup language.

   Project team members use the markup language to format wiki pages and comments.
7. Click **Finish**.

With an Initial Git Repository

To upload application files soon after creating a project, create a project with an initial Git repository. You can choose the Git repository to be empty, populated with a readme file, or populated with data imported from another Git repository.

1. On the Organization page, click **+ Create Project**.
2. On the Project Details page of the New Project wizard, in **Name** and **Description**, enter a unique project name and a project description.
3. In **Security**, select the project's privacy.
4. Click **Next**.
5. On the Template page, select **Initial Repository**, and click **Next**.
6. On the Project Properties page, from **Wiki Markup**, select the project's wiki markup language.

   Project team members use the markup language to format wiki pages and comments.
7. In **Initial Repository**, specify how to initialize the Git repository.
   - If you prefer a blank repository or want to push a local Git repository to the project, select **Empty Repository**.
   - Some Git clients and IDEs can't clone an empty Git repository. Select **Initialize repository with README file** if you're using such a client or an IDE. DevCS creates a `readme.md` file in the Git repository.

     You can edit the contents of the `readme.md` file after creating the project, or delete the file if you don't want to use it.
   - To import a Git repository from another platform such as GitHub or Bitbucket, or from another project, select **Import existing repository**.

     In the text box, enter the URL of the external Git repository. If the repository is password protected, enter the credentials in **Username** and **Password**. Note that DevCS doesn't store your credentials.
8. Click **Finish**.

From an Exported Project

If you've created a project before and backed up its data to an OCI Object Storage bucket or an OCI Object Storage Classic container, you can create a project and import the data from the backed up project.

To import project data from an OCI Object Storage bucket or an OCI Object Storage Classic container, you need the these details:

OCI Object Storage	OCI Object Storage Classic
Name of the target bucket	Name of the target container
Name of the exported archive file	Name of the exported archive file

OCI Object Storage	OCI Object Storage Classic
Private key and fingerprint of a user who has the `BUCKET_INSPECT` (or `BUCKET_READ`) and `OBJECT_READ` permissions of the bucket	Credentials of a user with the **Storage.Storage_Administrator** or **Storage_ReadOnlyGroup** identity domain role.
Details of the compartment that hosts the bucket Contact the OCI administrator for the details and get the required input values. See Get the Required OCI Input Values.	Service ID and the authorization URL of OCI Object Storage Classic Contact the identity domain administrator or the OCI Object Storage Classic administrator for the details and get the required input values. See Get OCI Object Storage Classic Input Values.

After you have the required input values, import the project.

1. On the Organization page, click **+ Create Project**.
2. On the Project Details page of the New Project wizard, in **Name** and **Description**, enter a unique project name and a project description.
3. In **Security**, select the project's privacy.
4. Click **Next**.
5. On the Template page, select **Import Project**, and click **Next**.
6. In the Storage Connection section of the Project Properties page, to import the project from OCI Object Storage container, in **Account Type**, select **OCI**.

   To import the project from OCI Object Storage Classic container, in **Account Type**, select **OCI Classic**.
7. If you selected **OCI** as **Account Type**, enter the required connection details.
   a. In **Tenancy OCID**, enter the tenancy's OCID copied from the Tenancy Details page.
   b. In **User OCID**, enter the user's OCID value who can access the bucket.
   c. In **Home Region**, select the home region of the OCI account.
   d. In **Private Key**, enter the private key of the user who can access the bucket.
   e. In **Passphrase**, enter the passphrase used to encrypt the private key. If no passphrase was used, leave the field empty.
   f. In **Fingerprint**, enter the fingerprint value of the private-public key pair.
   g. In **Compartment OCID**, enter the compartment's OCID copied from the Compartments page.
   h. In **Storage Namespace**, enter the storage namespace copied from the Tenancy Details page.
8. If you selected **OCI Classic** as **Account Type**, enter the required connection details.
   a. In **Service ID**, enter the value copied from the last part of the **REST Endpoint** URL field of the Service Details page.

      For example, if the value of **REST Endpoint** URL is `https://demo12345678.storage.oraclecloud.com/v1/`**`Storage-demo12345678`**, then enter `Storage-demo12345678`.

b. In **Username** and **Password**, enter the credentials of the user who can access the archive file.

c. In **Authorization URL**, enter the URL copied from the **Auth V1 Endpoint** field of the Service Details page.

   Example: `http://storagetria01234-usoracletria12345.storage.oraclecloud.com/auth/v1.0`.

9. Click **Next**.
10. On the Project Properties page, from **Wiki Markup**, select the project's wiki markup language.

    Project team members use the markup language to format wiki pages and comments.
11. In **Container**, select the storage bucket or the container where the data was exported.
12. In **File**, select the exported file.
13. Click **Finish**.

If the import fails, an empty project is created. You can try to import the data again without creating a project. To check the import log, see the **History** tab of the **Data Export/Import** page under **Project Settings**.

From a Project Template

Using a project template, you can quickly create a project with predefined and populated artifacts such as Git repositories and build jobs. When you create a project from a project template, the defined artifacts of the project template are copied to the new project. If you don't want to use a copied artifact, you can delete it. Note that after you create a project from a template, any updates made to the project template aren't reflected in the created project.

These types of project templates are available.

Project Template	Description
Public templates	The DevCS team creates and manages the public templates. They are available to all users across all identity domains and are marked by a **Public Template** label.
Shared templates	Your organization users create and manage shared templates. They are listed by name and are available to all users of the organization.
Private templates	Not listed by name to general users, but accessible only through their private keys. To create a project from a private project template, you must've its private key. The templates are visible by name only to the members of the project template.

1. On the Organization page, click **+ Create Project**.
2. On the Project Details page of the New Project wizard, in **Name** and **Description**, enter a unique project name and a project description.
3. In **Security**, select the project's privacy.
4. Click **Next**.

5. On the Template page, select the project template, and click **Next**.

   To create a project from a private template, select **Private Template**, and click **Next**. On the Private Template Selection page, enter the private key in **Private Key**, and click **Next**.

6. On the Project Properties page, from **Wiki Markup**, select the project's wiki markup language.

   The markup language is used to format wiki pages, and comments on Issues and Merge Request pages.

7. Click **Finish**.

In the new project, these artifacts are copied from the project template:

Artifacts	Description
Git repositories	Defined Git repositories of the project template are copied to the new project. You can use the copied Git repositories and change their files for your use, or delete them. In the navigation bar, click **Git** to view the copied Git repositories.
Build jobs and pipelines	All build jobs and pipelines of the project template are copied to the new project. You can change these jobs, create their copies, or delete them. In the navigation bar, click **Builds** to view the copied jobs and pipelines.
Deployment configurations	All deployment configurations of the project template are copied to the new project. You can change the deployment configurations or delete them. In the navigation bar, click **Deployments** to see the copied deployment configurations.
Wiki pages	All wiki pages of the project template are copied to the new project. You can change the wiki pages or delete them. In the navigation bar, click **Wiki** to see the copied wiki pages.
Announcements	All active project announcements of the project template are copied to the new project. You can't edit the copied announcements as they are read-only, but you can activate or deactivate them. In the navigation bar, click **Project Administration**, and then click **Announcements** to activate or deactivate them.
Links	All link rules of the project template are copied to the new project. Link rules enable you to convert plain text to links when the text is entered in the commit and merge request comments. In the navigation bar, click **Project Administration** and then click **Links** to see the copied link rules.

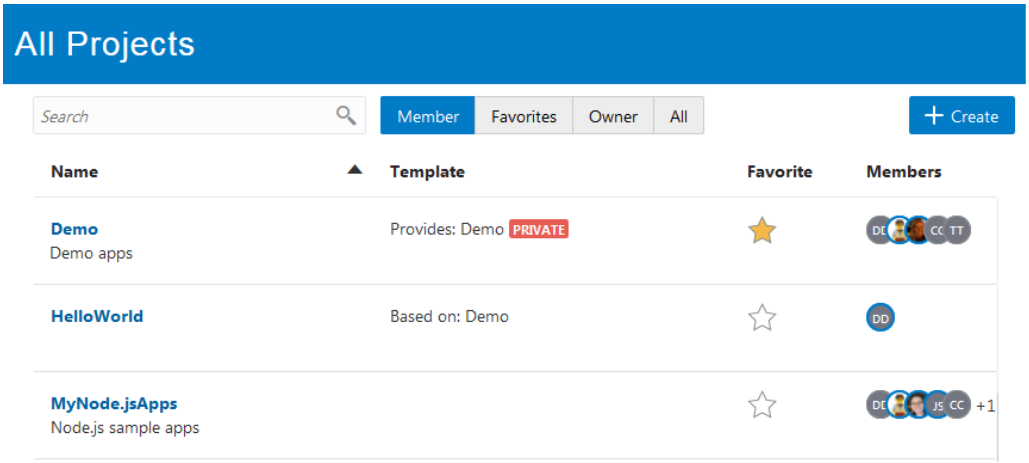
From IDEs

You can create a DevCS project from JDeveloper and NetBeans IDE.

See Create a DevCS Project in JDeveloper and Create a DevCS Project in NetBeans IDE.

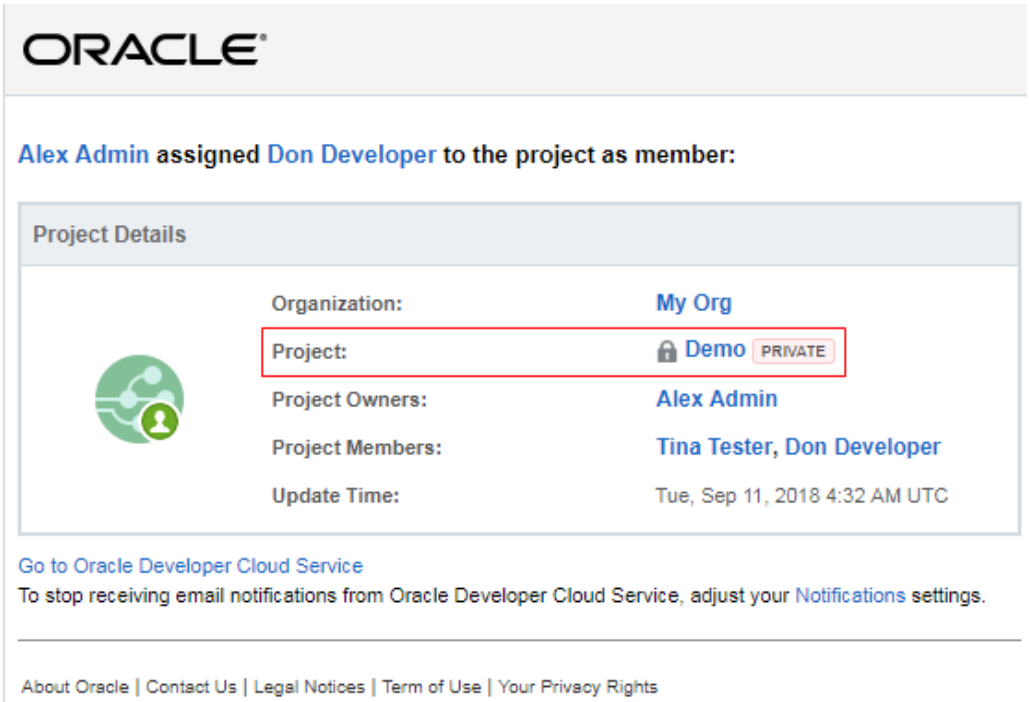
Open a Project

You can open a project only if you're a member or an owner, or if the project is shared. To open a project, on the **Organization** page, click its name. To search for a project, use the filter toggle buttons or the search box.

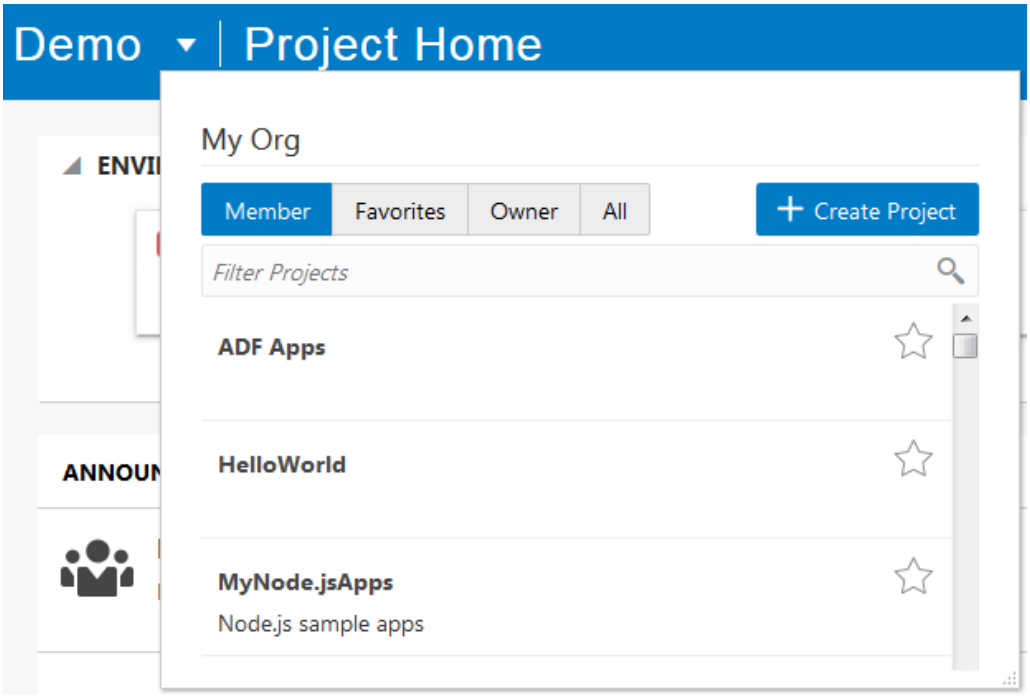


To quickly access a project, click **Favorite** ☆ and add it to your favorites list. To see your favorite projects, click the **Favorites** toggle button.

If you're invited to join a project, you'll find the project link in the email you received when you were added to the project.



To switch to another project from an open project, click next to the project name. From the menu, click the project name to open.



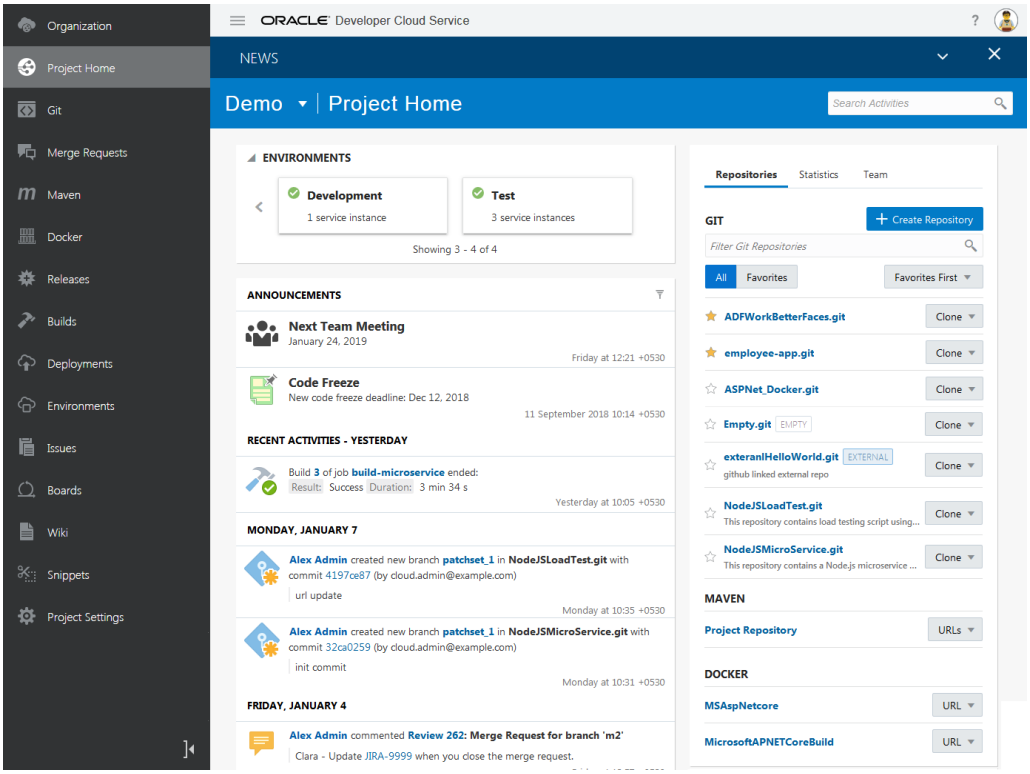
After opening the project, you land on the **Project Home** page.

Review a Project's Summary

From the **Project Home** page, you can view a summary of the project's actions, repositories, team members, and statistics.

Action	How To
Navigate between different pages of DevCS	Use the project navigation bar on the left side of the page
See your project's software development environments	Expand **Environments**. Click an environment to view its service instances.
Check the latest activities of the project	Use the recent activities feed. To filter the feed, click **Select activity categories** .
See the Git, Maven, and linked Docker repositories of the project	Click the **Repositories** tab.
See issues and merge requests that require your attention	Click the **Graphs and Statistics** tab.

Action	How To
Know your team members	Click the **Team** tab.



The **Project Home** page remembers the last opened tab (Repository, Graphs, or Team) in the current browser session and opens it automatically next time you open the **Project Home** page. If you sign out or close the browser and then sign in and open the **Project Home** page, the **Repositories** tab opens by default.

Add and Manage Project Users

After creating a project, you'd want to add team members to collaborate on the project. You'd also want to allow or limit their access to project data or actions they can perform on the project.

In a project, you can add and manage team members from the **Team** tab of the **Project Home** page. Before you add a user, make sure that the user is a member of the identity domain and is assigned the `DEVELOPER_ADMINISTRATOR` (Developer Service Administrator) or the `DEVELOPER_USER` (Developer Service User) identity domain role.

You must be assigned the project **Owner** role to add and manage project users.

Action	How To
Add a user to the project	1. In the navigation bar, click **Project Home**. 2. Click the **Team** tab. 3. Click **+ Create Member**. 4. In the Create Member dialog box, in **Role**, specify the new member's role. 5. In **Username**, enter or select the user from the list. 6. Click **Add**.
Add multiple users to the project	1. In the navigation bar, click **Project Home**. 2. Click the **Team** tab. 3. Click **+ Create Member**. 4. In the Create Member dialog box, select the **Multiple Users** check box. 5. In **Username**, enter or select the user from the list, and click **Add User**. The selected user adds to the **Username List** text box. If you know the usernames of users to add, enter the usernames manually separated by a space, a comma, a semicolon, or a new line. 6. Click **Add**.
Change a user's project role	To change a user's role to **Owner**, mouse over the user name and click **Promote to Owner**. To change a user's role to **Member**, mouse over the user name and click **Demote to Member**.
Remove a user from the project	Before removing a user, change the user's ownership of assigned issues and merge requests. Select the user and click **Remove**.

Add Users from Another Project

If the users that you want to add to your project are members of another project that you can access, you can copy that project's user list and add the users to your project.

1. Open the project that has users already added.
2. In the navigation bar, click **Project Home**.
3. Click the **Team** tab.
4. Click **Export**.
5. In the Members List Export dialog box, copy the names of project members.
6. Click **OK** or **Close** to close the dialog box.
7. Open the project where you want to add the copied users.

8. In the navigation bar, click **Project Home**  .
9. Click the **Team** tab.
10. Click **+ Create Member**.
11. In the New Member dialog box, select the **Multiple Users** check box.
12. In **Username List** text box, paste the copied names of project members.
13. Click **Add**.

3
Plan Your Project

After creating a project and adding team members, the next step would be plan your project. Project planning includes tasks such as defining goals and objectives, creating a schedule, identifying stakeholders, and setting up various software development environments.

This table describes the Oracle Developer Cloud Service pages you'd use to plan your project.

Use this page ...	To:
Environments	Add and manage project's environments and their service instances.
Issues	Create and track issues.
Boards	Manage issues using Agile boards.

Manage Software Development Environments

An environment lets you define and manage Oracle Cloud PaaS and Oracle Cloud Infrastructure service instances as a single entity.

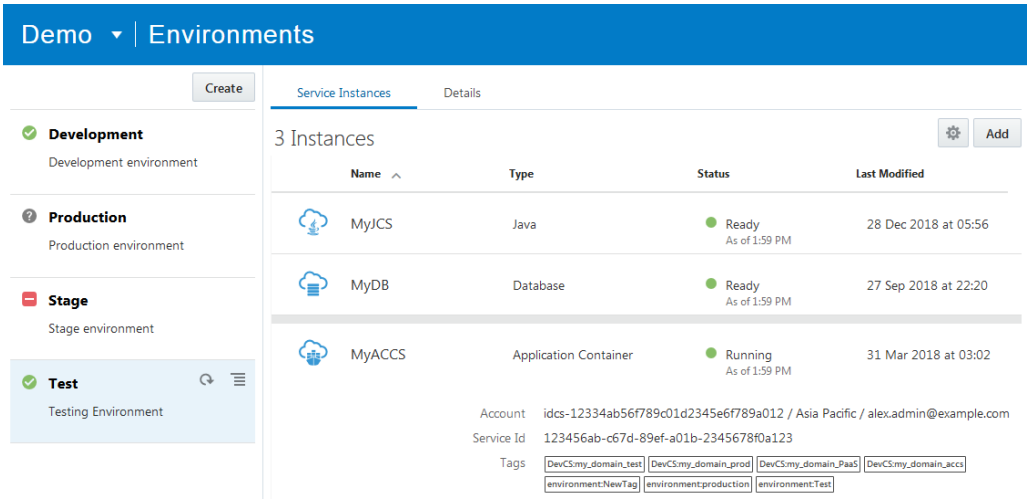
You might create an environment for your QA team with an Oracle Database Cloud Service instance to host data, say, and maybe an Oracle Java Cloud Service instance to deploy the application to and run Selenium tests. You could then create a Stage environment that uses the same Oracle Database Cloud Service instance as the QA environment, but a different Oracle Java Cloud Service instance to deploy the application to. You can also instantly see the health of all service instances comprising each environment right on the Project Home page or on the Environments page, without having to stop what you're working on to go out to Oracle Cloud My Services page.

You can access and manage the project's environments from the Environments page. You can create and delete environments, and add or remove service instances from existing environments. You can also update the details of the environment and view the details of its service instances. For each environment, the **Service Instance** tab lets you capture information in a single place so you won't have to hunt for it later, such as the health status of service instances, their account names, tags, and service IDs. The **Details** tab displays its details such as name, description, and tags of the selected environment.

Tags associate a service instance to an environment. When you add a service instance to an environment, a new tag is created, but you may remove it and add your own tag or use the service instance's tag. Open the service console to see the tags (if any) of its instances.

If a tag is used in multiple environments of the same project or other projects, all service instances associated with the tag are automatically available in those

environments. Removing a tag from an environment removes service instances using the same tag from other environments too.



Set Up an Environment

You can create an environment and add service instances to it from different identity domains. For example, you can add an Oracle Database Cloud Service instance from one identity domain and an Oracle Java Cloud Service instance from another identity domain.

To add or remove a service instance, you need credentials of a user who is assigned the administrator role for the service type. To assign or modify roles, see Modifying Identity Cloud Service User Roles in *Managing and Monitoring Oracle Cloud*.

1. In the navigation bar, click **Environments** .
2. Click **Create** (or **Create Environment** if the page is empty).
3. In **Environment Name**, enter a unique name. In **Description**, enter a description.
4. Click **Create**.
5. In the **Service Instances** tab, click **Add**.
6. In the Add Service Instances dialog box, select the check boxes of service instances and click **Add**.

   By default, the dialog box shows the service instances that you can access from the current identity domain. To search for services from another identity domain or account, click **Edit** and enter the details in the popup that opens.
   To search for services in a traditional account, enter the identity domain name. To search for a service in an IDCS account, enter the Identity Service ID in the `idcs-<hexadecimalID>` format. You can find the identity domain name or the Identity Service ID from the service details page that you open from My Services Dashboard.
7. In the Add Environment Tags dialog box, specify tags to associate the selected service instances to the environment, and click **Add**.

   By default, a tag is provided, but you may remove it and add your own tags. Spaces aren't allowed. Press Enter after adding each new tag.

8. If necessary, repeat steps 4-6 to find and add service instances from different identity domains and data centers.

Manage an Environment

After creating an environment, you can add and manage its instances.

This table lists actions you can perform to manage an environment and its instances.

Action	How To
Edit the name and description of an environment	In the **Details** tab, click **Edit**. Edit the details and click **Save**.
Manage environment definition queries	An environment may contain service instances from multiple identity domains and a definition query defines the instances of an identity domain. To see the details of those identity domains or edit them, click **Manage Environment Definition**  . From the Manage Definition Environment dialog box, click **New** to add a new identity domain query, or click **Action**  to edit or delete an existing query.
Start, stop, or restart an instance	In the **Service Instances** tab, mouse-over the service instance, click **Action**  , and select **Start**, **Stop**, or **Restart**. To check the status of the service, select **Refresh Status** from the **Action**  menu.
Remove a service instance from an environment	To remove a service instance from an environment, remove its tags. In the **Service Instances** tab, mouse-over the service instance, click **Action**  , and select **Remove**. In the Remove Service Instance dialog box, select the check boxes of tags to remove. In **Password**, enter the password of the user whose username is displayed in the dialog box, and click **Remove**.
Delete an environment	In the environments list, mouse-over the environment, click **Action**  , and select **Delete**. Remember that service instances of the environment aren't deleted.

Track and Manage Tasks, Defects, and Features

Issues help you track new feature requests or enhancements, assign tasks to team members, or file bugs.

You can create, update, and search issues from the **Issues** page, Agile boards, and from IDEs. You can also use REST APIs to create, retrieve, and update issues (see REST API for Managing Issues in Oracle Developer Cloud Service).

In a project, you can create an issue as a Task, Defect, or a Feature. If your team uses an Agile Scrum board to update issues, Epic and Story types of issues are also available. An Epic is a larger issue typically comprised of multiple smaller sub-issues or Story issues. An Epic can span multiple sprints and must have sub-issues.

Here's a summary of the key steps you'd perform to create and track issues.

1. If you're a project owner, set up products, components, and releases of your project. These would be useful when you and your team members create and assign issues to identify tasks, defects, and features.

   If the default fields of issues don't meet your requirement, create custom fields.
2. Create issues and assign them to your team members.
3. Update issues.

   You can update issues from the **Issues** page or from an Agile board.

Set Up Issue Products and Custom Fields

Before creating and assigning issues to project members, you can define products, components, default owners of components, and releases of your project. You can create multiple product categories, components, and sub-components; customize the releases; and add custom fields for your project.

 You must be assigned the project **Owner** role to add and manage issue products, components, and custom issues.

Create and Configure Issue Products

When you define a product, you also define its releases and components. A product is a category that represents an entity. A component is a subsection of a product. A release is a release name or number of the product.

You can create multiple products for a project and select them from the **Products** drop-down list on the create or edit issue page. Each product must have at least one component and one release. For example, you can create a **Report** product with **1.0**, **2.0**, **3.0**, and **PS1** as its releases, and **Sales**, **Marketing**, and **Demographics** as its components.

You can define products, components, and releases from the Products tab of the Administration: Issue Tracking page.

To open the Products tab of the Issue Tracking page:

1. In the navigation bar, click **Project Administration**  .
2. Click **Issue Tracking**.
3. Click the **Products** tab.

This table describes the product management actions you can perform.

Action	How To
Create a product	1. Click **+ New Product**. 2. On the Create Product page, in **Name**, specify a unique product name. 3. To create a release, click **+ New Release**, and enter a release name. To make a release the default release of the product, click **Mark as Default** ✓. 4. To create a component, click **+ New Component**, enter a component name, and select its default owner (optional). To make a component the default component of the product, click **Mark as Default** ✓. 5. To create a Found In tag, click **+ New Found In Tag**, and enter a tag name. 6. Click **Done**. To reorder a release or a component, mouse over the name and use the drag-and-drop action to move it up or down.
View or edit a product	From the products list, select the product. On the right side of the page, view or edit its details.
Delete a product	You can't delete a product until issues or merge requests refer to it. First, remove all issues and merge requests that refer to the product, and then remove the product. 1. In the products list, click **Delete** ✕ to the right of the product name. 2. In the Delete Product dialog box, click **Yes** to confirm.

Create and Configure Issue Custom Fields

If the default fields of issues don't meet your requirement, you can custom fields for the issues of your project. You can create and manage the fields from the Custom Fields tab of the Issue Tracking page. While creating or updating an issue, the custom fields are available in the Details section of the New or Edit Issue page.

1. In the navigation bar, click **Project Administration** ⚙.
2. Click **Issue Tracking**.
3. Click the **Custom Fields** tab.

You can create these types of custom fields:

- Single line input text
- Single selection
- Multi selection
- Long text input
- Time and Date
- Check box

This table describes the custom field management actions you can perform.

Action	How To
Create a custom field	1. Click **+ New Custom Field**. 2. On the Create Custom Field page, in **Name**, specify a unique name. 3. In **Label**, enter the field's display label. 4. If you don't want the custom field to appear as a parameter when new issues are created, deselect the **available for New Issues** check box. 5. From the **Type** drop-down list, select the field type. If you select **Single Selection** or **Multi Selection**, click **+ New Value** to specify the field's options. 6. Click **Done**.
View or edit a custom field	From the custom fields list, select the field. On the right side of the page, view or edit its details. You can't change a custom field's **Name** or **Type**. To edit the value of **Name** or **Type**, remove the custom field and create the custom field again.
Hide a custom field	From the custom fields list, select the field. On the right side of the page, select the **Obsolete (hidden)** check box.
Delete a custom field	1. In the custom fields list, click **Delete** to the right of the field name. 2. In the Delete Custom Field dialog box, click **Yes** to confirm. All existing issues are automatically updated to remove the custom field.

Create Issues

You can create an issue from the Issues page, Agile boards, IDEs, and from REST API. When you create an issue, it's assigned a unique ID and is added to the issues list on the Issues page.

When you create an issue, you specify its summary, type, severity and priority, due date, tags, and release. You can assign the issue to a team member or to yourself, or leave the field blank to assign the issue later to a team member.

Create an Issue from the Issues Page

1. In the navigation bar, click **Issues** .
2. Click **New Issue**.
3. On the New Issue page, in **Summary** and **Description**, enter the issue's title and description.
4. In **Details**, specify the issue type, its severity and priority, product details, release, ownership, and project tags.
5. In **Time**, specify the due date and estimate (in days).

One day is estimated of 8 hours. To specify 3 hours, enter 0.375. To specify 2 days and 2 hours, enter 2.250.

6. In **Agile**, specify the effort estimate in Agile story points.
7. If there are any custom fields defined in your project, fill in the details, as required.
8. Click **Create Issue**.

Create an Issue from an IDE

You can create and manage issues from Eclipse IDE, NetBeans IDE, and JDeveloper.

- Manage DevCS Issues in the Eclipse IDE
- Manage DevCS Issues in the NetBeans IDE
- Manage DevCS Issues in JDeveloper

Search Issues

You can search for issues using the pre-defined filters under **Standard Searches**, **My Searches**, **Shared Searches**, or **Global Searches**. If you can't find the issue, you can run a basic search or an advanced search.

To run a basic search, use the **Search Issues** box in the upper-right corner of the Issues page. You can search for a term in the summary, description, or comments of issues. To clear the search term, click **Clear Filter** .

To run an advanced search, use the **Advanced Searches** link. You can search for issues using various parameters such as sprints, product, version, date, owner, type, and priority.

To save the search query as a filter, click **Save this search**. To see the search query expression, click **Show Search String**. Later, if you want to edit the search query, click **Edit this search**.

Save a Custom Search

You can save your basic or the advanced search query as a custom search filter for future use.

1. On the Issues page, run a basic or an advanced search.
2. On the search results page, click **Save this search**.
3. In the Save Search dialog box, enter the search name.

   The custom search filter is available to you only. To share the search filter with project members, in the Save Search dialog box, select the **Shared** check box. In **Share with the following users**, select the users with whom you want to share the search query.

   To share the search filter with all project members, select the **Share with everyone** check box.
4. Click **OK**.

If you didn't select the **Shared** check box, the search query appears as a filter under **My Searches**. If you selected the **Shared** check box, the search query appears as a

filter under **Shared Searches**. If you selected the **Share with everyone** check box, the search query appears as a filter under **Global Searches**.
To edit a custom search query, mouse over the query under **My Searches** and click **Edit** . To delete a custom search query, mouse over the query under **My Searches** and click **Delete** .

Share Custom Search Filters

You can share your existing custom search filters with other project members, which they can use to view the issues as you want.

Element	Description
Share a search filter with specific project members	1. In **My Searches**, mouse over the filter link, and click **Share** . 2. In the Start Sharing Search dialog box, select the project member names in **Share with following users**. 3. Click **OK**. The filter link moves from **My Searches** to **Shared Searches**.
Share a search filter with all project members	1. In **My Searches**, mouse over the filter link, and click **Share** . 2. In the Start Sharing Search dialog box, select the **Share with Everyone** check box. 3. Click **OK**. The filter link moves from **My Searches** to **Global Searches**.
Stop sharing a search filter	1. In **Global Searches** or **Shared Searches**, mouse over the filter link and click **Stop Share** . 2. In the Stop Sharing Search dialog box, click **OK**. If the search query is being used by other project members, the dialog box shows their list. When you stop sharing a search query, it's removed from the **Shared Searches** or **Global Searches** list of all project users.

View and Update Issues

To view or update an issue, click the issue's summary or the ID link on the Issues page. An issue link could also be found in the recent activities feed, wikis, Agile boards, and merge requests.

While updating an issue, you can change its status, properties, reassign it to another member, and change its priority or severity. You can also add comments in the **Comments** tab, upload attachments in the **Attachments** tab, and check the update history of an issue in the **History** tab. Updates made to issues can also be tracked in the recent activities feed of the Project Home page.

Resolve an Issue

You can resolve an issue as Fixed, Invalid, Duplicate, Will not fix, Works for me, or Need info.

1. Click the issue link to open it in the Issues page.
2. From the **Status** drop-down list, select `Resolved`.
3. From the **Resolution** drop-down list, select the resolution.

Sub-status	Description
Fixed	Indicates the issue has been fixed and is awaiting feedback from the QA team. After verifying the fix, the QA team sets the issue's status to **Verified** or **Closed**.
Invalid	Indicates the issue isn't a valid issue.
Will not fix	Indicates the issue won't be fixed.
Duplicate	Indicates the issue is a duplicate of an existing issue. Enter the issue ID of the existing issue in **Duplicate Of**.
Works for me	Indicates the issue can't be reproduced.
Need info	Indicates the current issue description is insufficient to reproduce the issue and more information is required.

Mark an Issue as Duplicate

If find a duplicate issue, mark it as a duplicate and specify the original issue.

1. Click the issue link to open it in the Issues page.
2. From the **Status** drop-down list, select **Resolved**.
3. From the **Resolution** drop-down list, select **Duplicate**.
4. In **Duplicate Of**, enter the original issue ID or the summary text, and select the original issue.
5. Click **Save**.

Update Time Spent on an Issue

When you work on an issue, create a time spent entry each time you update the issue.

1. Click the issue link to open it in the Issues page.
2. In the **Time** section, click **Add Time Spent**.
3. In the Add Time Spent dialog box, in **Time Spent**, specify the number of days you've spent on the issue.
4. To subtract the value specified in **Time Spent** from the existing value of **Remaining** (if set), use the default **Reduce remaining ... days by entered Time Spent** option.

   If **Remaining** isn't set, then the value specified in **Time Spent** is subtracted from **Estimate**. The option is disabled if the **Estimated** field isn't set.

   To specify the remaining days manually, select the **Set to** option and specify the remaining estimate.
5. In **Comment**, add a comment.
6. Click **OK**.

The **Time Spent Log** section shows the time spent entry of the time spent and updates the graph. To edit the time spent entry, click **Edit** and update the fields in the Edit Time Spent dialog. To remove a time spent entry, click **Remove** and update the fields in the Update Time Spent dialog box. The remaining time is adjusted automatically.

Associate an Issue with a Sprint

You can associate an issue with a sprint from the Edit Issue page. You can associate only one sprint with an issue.

1. Click the issue link to open it in the Issues page.
2. In the **Agile** section, from the **Sprint** drop-down list, click the search box, and select the sprint from the list.
3. Click **Save**.

Create a Relationship Between Issues

You can create a parent-child relationship between issues.

Action	How To
Create a child issue to an issue	You can create multiple child issues to an issue. 1. Click the issue link to open it in the Issues page. 2. Click **+ New Sub-issue**. 3. Enter details for the new issue and click **Create Issue**. 4. In the header, click the parent issue ID to open the parent issue. In the **Associations** section, verify the child issue ID.
Add a parent issue to an issue	You can add only one issue as a parent to an issue. 1. Click the issue link to open it in the Issues page. 2. In the **Associations** section, in **Parent Issue**, enter the issue ID or summary text of the parent issue, and select it. 3. Click **Save**.

Update Multiple Issues

On the Issues page, you can update multiple issues in a batch to apply the same update.

1. In the issues list, press the Ctrl key or the Shift key and click the rows of issues.

   You can also use the Space bar and Up-Down arrow keys to select the issues. To select all issues, click **Select All**.
2. Click **Update Selected**.
3. On the Mass update page, select the check boxes of fields to update and specify their values.

   Note that the **Component** check box is enabled when its **Product** is selected.

The contents of **Found In** and **Release** are determined by **Product**. If **Product** isn't specified, the intersection of all known products is used. For example, if product `P1` has **Found In** set to `1.0, 2.0` and product `P2` has **Found In** set to `1.0, 1.5`, then with no product specified, the **Found In** is set to `1.0`. The same logic is applied for **Release** too.

4. Click **Next**.
5. On the Issues will be Updated page, verify the summary, and click **Save**.

Issues that fail the update are listed with a description of the error. To resolve the error of multiple issues, select the error issues and click **Update Selected**. You're navigated to the Issues Selected page where the previous changes you made are shown.

If all issues are successfully updated, you're navigated back to the Issues page.

Update Issues from IDEs

You can also update the issues from IDEs such as Eclipse IDE, NetBeans IDE, and JDeveloper.

See these for more information:

- Manage DevCS Issues in the Eclipse IDE
- Manage DevCS Issues in the NetBeans IDE
- Manage DevCS Issues in JDeveloper

Watch an Issue

You can set up a watch on an issue and get email notifications when a project user updates an issue, adds a comment, or adds or removes an attachment.

Action	How To
Issues assigned to you	By default, you get email notifications of issues assigned to you. If you aren't getting the email notifications, select the **Issue updates, attachments and comments** check box in your user preferences page. 1. In the branding bar, click the user avatar, and select **Preferences**. 2. Click the **Notifications** tab. 3. Select the **Issue updates, attachments and comments** check box, if not selected. 4. To the left of the **User Preferences** title, click **Close** to return to the last opened page.

Action	How To
Issue created by another user	1. In the branding bar, click the user avatar, and select **Preferences**. 2. Click the **Notifications** tab. 3. Select the **Issue updates, attachments and comments** check box, if not selected. 4. To the left of the **User Preferences** title, click **Close** to return to the last opened page. 5. Open the issue in the Issues page. 6. In the **Details** section, in **CC**, enter and select your name. You may also enter other names of other users if you want to notify them too. 7. Click **Save**. To stop watching, remove your name from the **CC** field.
Issues you created but are assigned to another user	By default, you get email notifications of issues created by you. When you create an issue and assign it to another user, your name is set in the **CC** field of the issue. Open the issue in the Issues page and verify your name in the **CC** field of the **Details** section. To stop watching, remove your name from the **CC** field.

Use Agile Boards to Manage and Update Issues

The Agile methodology of software development is a type of incremental model that focuses on process adaptability and customer satisfaction. In Oracle Developer Cloud Service (DevCS), you use the Agile methodology to manage issues in Scrum and Kanban boards.

If you are new to Agile, see `http://agilemethodology.org/` for more information.

Before creating a board, appoint a team member as the Agile board's leader. The leader would be responsible to manage and update issues of the board. The leader could set up team meetings to discuss the progress of issues and then update them in the board.

Here's a summary of the key steps you'd perform as the board's leader to create and manage issues:

1. If required, create an issue query that returns issues you'd want to add to the Agile board.
2. Create an Agile board (Scrum or Kanban).
3. Configure the working days, progress states, and other properties of the board.
4. Manage sprints or active issues.
5. Update progress states of issues.
6. Review reports and adjust the sprints, issues, or the board accordingly.

Agile Boards Concepts and Terms

Before you start using the Agile boards, it's important that you know about key components and concepts of the Boards page.

Component	Description
Board	A Board is used to display and update issues of the project using the Agile methodology. There are two types of boards available: Scrum and Kanban. When you create a board, you associate it with an issue query and the issues returned by the query are added to the board. You can create your own board or use a board created by a team member. In a board, you update issues by moving them to different progress states of the board. Each progress state has some pre-defined conditions that specify which issues can be assigned to a progress state.
Scrum	In a Scrum board, tasks are broken small actions to be completed in fixed duration cycles, called as Sprints.
Kanban	In a Kanban board, tasks are managed with a focus on continuous delivery.
Sprint	A Sprint is a short duration (usually, a week or two) during which your team members try to implement a product component. You add the product component related issues to a sprint. When you start working on a product component, you start (or activate) its related sprints. To update issues of a sprint, you must first activate the sprint and add the sprint to the Active Sprints view.
Story Points	A Story Point is a metric that defines the relative effort of work and helps to understand how complex the issue is.
Backlog view	In a Scrum board, the Backlog view displays issues of the board, active and inactive sprints of the board, and the sprints from other boards that contain issues matching the board's query. Each sprint lists issues added to it. The Backlog section (the last section of the Backlog page) lists all open issues that aren't part of any sprint yet. The Backlog view doesn't show the resolved and closed issues. In a Kanban board, the Backlog view displays active issues (issues being actively worked on) in the Active Issues section and a backlog list of issues (issues aren't being actively worked on) in the Backlog section. The Epic issues don't appear in the Backlog view.
Active Sprints view	Available in a Scrum board, the Active Sprints view lists all active sprints of the board and enables you to update an issue status by dragging and dropping it to the respective status columns.
Active Issues view	Available in a Kanban board, the Active Issues view enables you to manage the progress of active issues.
Reports view	Displays various reports and charts that summarize the progress of issues.

Create and Configure Agile Boards

An Agile board contains issues returned by an issue filter. If none of the pre-defined or shared issue filters meet your requirement, create a custom search query and save it as a filter.

Create a Board

You can create a board from the Boards page. When you create a board, you specify the board type, an issue search query, and the estimation criteria.

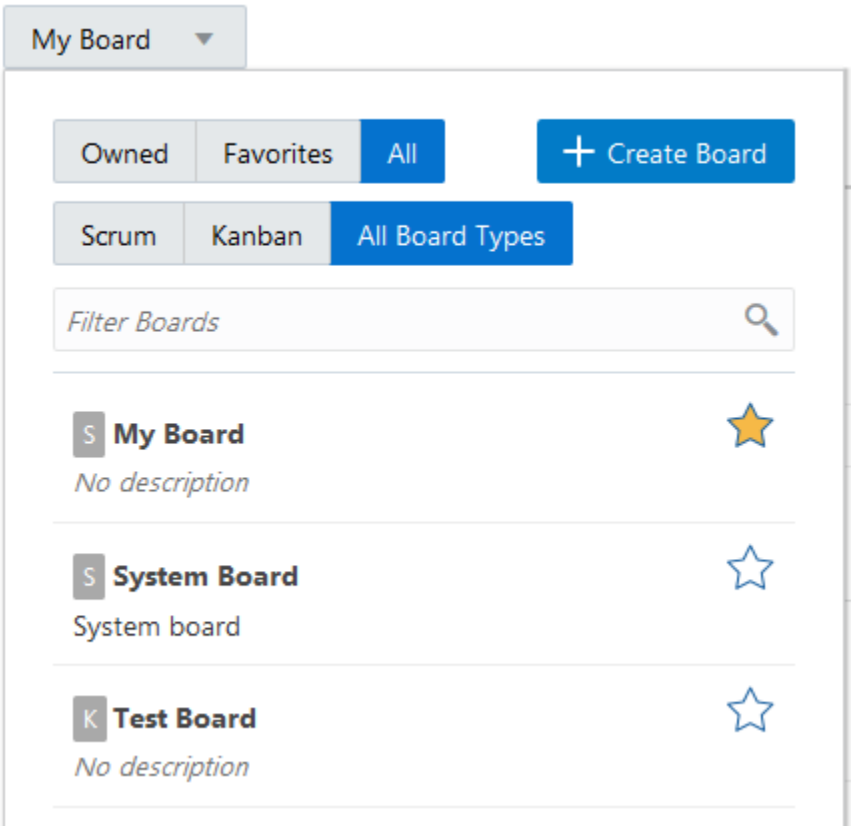
You must be a project member to create a board.

1. In the navigation bar, click **Boards** .
2. Click **+ Create Board**.
3. In the Create Board dialog box, enter a name and select the board type.
4. In **Search**, select the standard or custom issue search query. By default, **All Issues** is selected.
5. In **Estimation**, select the estimation type as **Story Points** or **Estimated Days**.
6. Click **Create**.

A board is created and issues matching the search query are added to the board, and you're navigated to the Backlog view. The board's owner role is also granted to you.

Note that **Resolved**, **Verified**, and **Closed** issues aren't added to the board. To add new issues to a board, edit the issue search query to reflect the issues in its search result. The issues are automatically reflected in the Backlog list of the board.

You can also create a board from the Switch Board menu. From the board name menu, click **+ New Board**.



After creating a board, you can configure its working days, progress states, and conditions.

Add and Manage Progress States of a Board

A progress state defines the progress of issues in a board. By default, each board has three progress states (To Do, In Progress, and Completed), but you can add more. Each progress state has some pre-defined conditions. A condition defines an issue's state. You cannot add, edit, or delete a condition.

You can add and manage progress states from the Configure Board page of the board.

1. Open the board.
2. From the **Board** drop-down list, select **Configure**.
3. Click the **Progress States** tab.

Action	How To
Edit a progress state	1. From the progress states list, select the progress state. 2. To edit the name and description, in **Name** and **Description**, enter a new name and description. 3. To update the capacity (number of issues in the progress state), update the value in **Suggested Issue Capacity**. If the number of issues exceeds the suggested capacity, a warning icon and a message appears in the Active Sprints or the Active Issues view. 4. To remove a condition, select the condition from the **Conditions** list, click > and move it to the **Unassigned Conditions** list. To add a condition, select the condition from the **Unassigned Conditions** list, click < and move it to the **Conditions** list. For example, if you remove the **Resolved - WorksForMe** from the **Completed** progress state, issues in the **Resolved - WorksForMe** state don't appear in the board.
Add a progress state	A condition can be associated with one progress state only. Before you add a progress state, remove the conditions that you want to apply to the new progress state from their existing progress states. 1. Click **+ Add Progress State**. 2. In **Name** and **Description**, enter a name and description. 3. In **Suggested Issue Capacity**, specify the number of issues to be allowed. If the number of issues exceeds the suggested capacity, a warning icon and a message appears in the Active Sprints or the Active Issues view. 4. To mark the state as the Completed state, select the **Completed State** check box. The check box is disabled if a **Completed** state exists or the current state isn't the last state in the list. 5. To add a condition, select the condition from the **Unassigned Conditions** list, click **Move to Conditions** and move it to the **Conditions** list. 6. To remove a condition, select the condition from the **Conditions** list, click **Move to Unassigned Conditions** and move it to the **Unassigned Conditions** list.
Reorder progress states	In the progress states list, use the Up and Down order buttons to change the orders of the states. The buttons appear when you mouse over the state name. The order of states in the list is reflected in the **Swimlanes** and **Columns** views. The **Completed** state must be the last state in the list. If **Completed** isn't the last state in the list, a **Warning** icon appear next to the state name. Any changes made to the page aren't saved until the **Completed** state is the last state in the list.
Delete a progress state	In the progress states list, mouse over the progress state, and click **Delete**. All conditions of the deleted progress state move to the **Unassigned Conditions** list and are available to new progress states. You can't delete the **Completed** state, but you can delete other states.

Click **Save** when you're finished.

Configure Working Days of a Board

You can configure the working days and non-working of a week by modifying the board's calendar.

The working and non-working days that you specify affect the output of the Sprint Report, Issues Report, and the Burndown Chart.

1. Open the board.
2. From the **Board** drop-down list, select **Configure**.
3. Click the **Working Days** tab.
4. On the Configure board page, specify standard working and non-working days.
   - In **Standard Working Days**, select or deselect the check boxes of the working weekdays.
   - In **Non-Working Days**, click **+ Add** to add a non-working date (such as a holiday). From the calendar, select the date.

     To edit a non-working day, select it from the list, and select the new date in the calendar.
   - Select (or deselect) the **Show Non-Working Days in Sprint Report Chart** check box to show (or hide) the non-working days in the sprint reports.

     If selected, the non-working days appear in gray at the top in the burndown charts.
5. Click **Save**.

Configure and Manage a Board

From the Board menu, you can select options to configure, duplicate, and delete the board. From the Configure Board page, you can edit and update the name, description, associated issue search query, and estimation criterion of a board.

Action	How To
Edit a board's name and description	1. From the **Board** drop-down list, select **Configure**. 2. In the **General** tab, in **Name** and **Description**, update the values. 3. Click **Save**.
Edit the board's search query and estimation	1. From the **Board** drop-down list, select **Configure**. 2. In the **General** tab, in **Search** select the search query. In **Estimation**, enter the new estimation value. 3. Click **Save**.

Action	How To
Enable or disable time tracking	If the time tracking is enabled, then the **Active Sprints** page shows the estimation in **Remaining Days** only. The **Backlog** page shows the estimation in **Remaining Days** if estimation metric is **Estimated Days**. If the time tracking is disabled, then the **Backlog**, **Active Sprints** and **Reports** pages show the chosen metric for **Estimation** (estimated days or story points) instead of **Remaining Days** and **Time Spent**. 1. From the **Board** drop-down list, select **Configure**. 2. In the **General** tab, in **Time Tracking**, select **On** or **Off** to enable or disable time tracking. 3. Click **Save**.
Create a copy of a board	To create another board with similar properties of an existing board that you can access, instead of creating a new board and manually copying properties, you can create a copy of the board. The copied board has properties include time tracking, progress states, and working days. 1. From the **Board** drop-down list, select **Copy Board**. 2. In the Copy Board dialog box, click **Copy**.
Delete a board	You can't delete a Scrum board with active sprints. You must complete the active sprints before you delete the board. You can delete a Kanban board with active issues or archives. All issues of the deleted board are returned to the backlog. 1. From the **Board** drop-down list, select **Delete Board**. 2. In the Delete Agile Board dialog box, select the **I understand that my agile board will be permanently deleted** check box and click **Delete**.

Use Scrum Boards

Using a Scrum board, you manage and update issues using sprints.

A Scrum board has three views: Backlog, Active Sprints, and Reports. The Backlog view lists all active and inactive sprints of the board, and a backlog list of issues. The Active Sprints view enables you to manage the progress of issues of an active sprint. The Reports view displays various issue reports.

Create and Manage Sprints

You can create and manage sprints from the Backlog view of a Scrum board. You must be a project owner or the board owner to create, edit, update, or delete a sprint.

When an issue is assigned to a sprint, the sprint is displayed in all boards whose issue query returns any issue of the sprint in its result. You might find such sprints in your board. Note that you can't edit or start sprints that you didn't create, or sprints that weren't created in the current board.

If there are no issues assigned to a sprint, the sprint is available only in the board in which it was created.

Action	How To
Create a sprint	1. From the toggle buttons, click **Backlog**. 2. In the Backlog view, click **+ Add Sprint**. 3. In the Add Sprint dialog box, enter the sprint name. If the Scrum board uses story points, then specify the sprint's capacity. 4. Click **OK**.
Edit a sprint	1. From the toggle buttons, click **Backlog**. 2. In the Backlog view, for an inactive sprint, click **...** and select **Edit sprint**. If the sprint is active (or started), click **Edit Sprint**. 3. In the Edit Sprint dialog box, you can update the sprint's name, board, start and end dates, and its capacity in story points. The **Story Points** field is available if story points were selected as the estimation for the board. 4. Click **OK**.
Start a sprint	When you or your team begin work on a sprint and want to update the issues of the sprint, you must first start (or activate) it. 1. From the toggle buttons, click **Backlog**. 2. In the Backlog view, for the sprint that you want to start, click **Start Sprint**. 3. In the Start Sprint dialog box, specify the start and end dates of the sprint. If necessary, update the sprint's name and its estimate. 4. Click **Start**. The started sprint is now available in the Active Sprints view.
Reorder a sprint	In the Backlog view, by default, the inactive sprints (also called as future sprints) in the order they were created. You can change their display order manually. For the inactive sprint that you want to move up or down, click **...** and select **Move sprint up** or **Move sprint down**. You cannot change the order of active sprints. The active sprints are ordered by Start Date in the Backlog view. If two sprints have the same Start Date, then they are ordered by name.
Move a sprint to another board	1. From the toggle buttons, click **Backlog**. 2. In the Backlog view, for an inactive sprint, click **...** and select **Edit sprint**. If the sprint is active (or started), click **Edit Sprint**. 3. In the Edit Sprint dialog box, from the **Board** drop-down list, select the target board. 4. Click **OK**.

Action	How To
Delete a sprint	You can delete an inactive sprint from the Backlog view of the board. You can't delete an active sprint. For the inactive sprint that you want to delete, click the **...** and select **Delete Sprint**. The sprint is deleted and all issues of the sprint are moved to the Backlog list.

Add and Manage Issues of a Sprint

From the Backlog view, you can add issues to or remove issues from a sprint using drag-and-drop actions.

When you add or remove issues from a sprint, keep a watch on the capacity of the sprint. If the total story points of the sprint's issues are more than the story points capacity of the sprint, a warning message appears. In such a case, you can either increase the capacity of the sprint or move some issues to another sprint.

Action	How To
Add an issue to a sprint	1. From the Backlog list or from the sprint that contains the issue, drag the issue to the target sprint. 2. In the blue dotted rectangle that appears when you drag the issue to the target sprint, drop the issue in the blue rectangle. You can also right-click the issue and select **Send to** > *target sprint name* to move the issue to the target sprint. A Sprint field is also added to the issue indicating the sprint it's associated with. If you're unable to use the drag-and-drop action, click the issue link to open it in the Issues page. Navigate to the **Agile** section. From the **Added To** drop-down list, select the sprint.
Create an issue from the sprint	In the sprint, below the issues table, click **New Issue**. On the New Issue page, enter the issue's details, and click **Create Issue**. The new issue is automatically associated with the current sprint.
Remove an issue from a sprint	1. From the sprint that contains the issue, drag the issue to the **Backlog** list. 2. In the blue dotted rectangle that appears when you drag the issue to the Backlog list, drop the issue in the blue rectangle. You can also right-click the issue and select **Send to** > **Backlog** to remove the issue from the sprint. If you're unable to use the drag-and-drop action, click the issue link to open it in the Issues page. Navigate to the **Agile** section. In the **Added To** field, click  .

Update Issues of an Active Sprint

The Active Sprints view enables you to manage the progress of issues of an active sprint.

You can use either the **Swimlanes** sub-view or the **Columns** sub-view to view the issues of the active sprint. The *Swimlanes* sub-view displays issues categorized into

issue owners (member whom the issue is assigned to). For each issue owner, the issues are categorized into vertical progress (or status) columns.

The *Columns* sub-view displays the issues categorized into vertical progress columns.

By default, each board contains three columns: To Do, In Progress, and Completed. If required, you can add more progress columns to the board from the Configure Board page.

Update an Issue's Progress in an Active Sprint

You can update an issue's progress in the Active Sprints view by dragging it from one progress column to another.

If you're unable to use the drag-and-drop action, click the issue ID to update its progress from the Edit Issue page.

1. Open the board that owns the active sprint.
2. Click **Active Sprints**.
3. Select the issue list view: **Swimlanes** or **Columns**.
4. To update an issue's progress, drag and drop it from one column to another.

   For example, when a team member starts work on an issue, drop the issue to the **In Progress** column (if exists).
5. In the Change Progress wizard, from the **To** drop-down list, select the new status of the issue. If necessary, enter a comment in the **Comment** field.

   If you want to update the time spent on the issue, click **Next**.
6. Click **OK**.

If the board uses story points, the number to the right of the column name is updated. An activity is also added to the **History** tab of the issue's **Activity** section.

Update Time Spent on an Issue

When you move an issue from one state to another, you can also update the time spent on the issue in the Change Progress wizard.

In the Add Time Spent page, in **Time Spent**, specify the number of days you've spent on the issue. In **Remaining**, use the default **Reduce remaining ... days by entered Time Spent** option to automatically subtract the value specified in **Time Spent** from the existing value of **Remaining**, if **Remaining** was set previously.

If **Remaining** was not set, then the value specified in **Time Spent** is subtracted from **Estimate**. The option is disabled if the **Estimated** field isn't set.

To specify the remaining days manually, select the **Set to** option and specify the remaining days.

Reschedule a Sprint

While updating issues of a sprint, you can change the start or end date of the sprint, or update its capacity from the Edit Sprint dialog box.

1. In the Backlog view, for the sprint you want to reschedule, click **Edit sprint**.
2. In the Edit Sprint dialog box, change the start and end dates.

To update the sprint's capacity, update **Story Points** . The field is available if story points were selected as the estimation of the board.

3. Click **OK**.

Complete a Sprint

You can complete a sprint from the Active Sprints view of the board.

You must be a project owner or the board owner to mark the sprint as completed.

1. Open the board the sprint belongs to.
2. Click **Active Sprints**.
3. In the sprint drop-down list on the left, select the sprint.
4. Click **Complete Sprint**.
5. In the Complete Sprint dialog box, select the **I understand that it will be removed from the Active Sprint view** check box, and click **Complete Sprint**.

After a sprint is complete, it's removed from the Active Sprints view. A warning displays if there are any incomplete issues in the sprint. All incomplete issues go back to the next inactive sprint, or to the Backlog section if there are no inactive sprints. The Sprint Report page opens showing the day-by-day progress of the sprint issues.

Review Issue Reports of a Scrum Board

Various reports are available in the Reports view of a Scrum board.

Report	Description
Burndown Chart	Displays the remaining amount of work to be finished in a sprint or an epic.
Cumulative Flow Chart	Displays the total number of issues in each of the board's progress states over time.
Control Chart	Displays information about issue's progress state change event on the timeline.
Sprint Report	Displays complete, incomplete and open issues of a sprint.
Epic Report	Displays complete, incomplete and open stories of an epic.
Velocity Report	Displays the velocity chart of completed sprints.

See Review Agile Reports and Charts.

Use Kanban Boards

Using a Kanban board, you manage issues using Active issues.

A Kanban board has three views: Backlog, Active Issues, and Reports. The Backlog view lists active issues (issues that're being actively worked on) and a backlog list of issues (issues aren't being actively worked on). The Active Issues view enables you to manage the progress of active issues. The Reports view displays various issue reports.

Add and Manage Active Issues

From the Backlog view, you can add issues to or remove issues from the Active Issues list using drag-and-drop actions.

When you add or remove issues from the Active Issues list, keep a watch on its capacity. If the total story points of active issues are more than the story points capacity of the board, a warning message appears. In such a case, you can either increase the capacity of the board or remove some issues from the Active Issues list.

Action	How To
Activate an issue	1. From the Backlog list, drag the issue to the **Active Issues** section. 2. In the blue dotted rectangle that appears when you drag the issue to the **Active Issues** section, drop the issue in the blue rectangle. In the Backlog list, you can also right-click the issue and select **Send to** >**Active Issues** to move the issue to the Active Issues list. Issues already added to sprints of Scrum boards aren't available in the Kanban board Backlog list. If you're unable to use the drag-and-drop action, click the issue link to open it in the Issues page. Navigate to the **Agile** section. From the **Added To** drop-down list, select the Active Issues option under the board name.
Create an issue from the Active Issues section	In the Active Issues section, below the issues table, click **New Issue**. In the New Issue page, enter the issue's details, and click **Create Issue**. The new issue is automatically activated and associated with the Kanban board.
Remove an issue from the Active Issues list	1. From the Active Issues section, drag the issue to the **Backlog** list. 2. In the blue dotted rectangle that appears when you drag the issue to the Backlog list, drop the issue in the blue rectangle. You can also right-click the issue and select **Send to** > **Backlog** to remove the issue. If you're unable to use the drag-and-drop action, click the issue link to open it in the Issues page. Navigate to the **Agile** section. In the **Added To** field, click **Remove**  .

Update Active Issues

The Active Issues view enables you to manage the progress of active issues.

You can use either the **Swimlanes** sub-view or the **Columns** sub-view to view the active issues. The *Swimlanes* sub-view displays issues categorized into issue owners (member whom the issue is assigned to). For each issue owner, the issues are categorized into vertical progress (or status) columns. The *Columns* sub-view displays the issues categorized into vertical progress columns.

By default, each board contains three columns: To Do, In Progress, and Completed. If required, you can add more progress columns to the board from the Configure Board page.

From the Active Issues view, you can update an active issue's progress state and archive the completed issues.

Update an Active Issue's Progress State

You can update an active issue's progress in the Active Issues view by dragging it from one progress column to another.

If you're unable to use the drag-and-drop action, click the issue ID and update its progress from the Edit Issue page.

1. Open the board that owns the active issues.
2. Click **Active Issues**.
3. Select the desired issue list view: **Swimlanes** or **Columns**.

   If necessary, use the sort list boxes to sort the active issues.
4. To update an issue's progress, move it from one column to another.

   For example, when a team member starts work on an issue, move the issue to the **In Progress** column (if exists).
5. In the Change Progress wizard, from the **To** drop-down list, select the new status of the issue. If necessary, enter a comment in the **Comment** field.

   If you want to update the time spent on the issue, click **Next**.
6. Click **OK**.

An activity is added to the **History** tab of the issue's **Activity** section.
Update Time Spent on an Issue

When you move an issue from one state to another, you can update the time spent on the issue in the Change Progress wizard.

In the Add Time Spent page, in **Time Spent**, specify the number of days you've spent on the issue. In **Remaining**, use the default **Reduce remaining ... days by entered Time Spent** option to automatically subtract the value specified in **Time Spent** from the existing value of **Remaining**, if **Remaining** was set previously.

If **Remaining** was not set, then the value specified in **Time Spent** is subtracted from **Estimate**. The option is disabled if the **Estimated** field isn't set.

To specify the remaining days manually, select the **Set to** option and specify the remaining days estimate.

Archive Completed Issues

You can create an archive of all issues listed in the Completed progress state from the Active Issues view of the board. The archived issues are then removed from the Active Issues view.

Before you archive completed issues, make sure that all parent and child issues are moved to the Completed progress state before you create an archive. An error is reported if the Completed progress state contains a completed parent issue with an open child issue, or a completed child issue with an open parent issue.

1. Open the board.
2. Click **Active Issues**.

3. Verify the issues list in the Completed progress state.

   Note that the list may include some issues that're filtered out of the display of the current **Active Issues** view by the board's query.

4. Click **Archive Completed Issues**.

5. In the Archive Completed Issues dialog box, in **Archive Name** edit the archive name (by default it is `Archive <date-time stamp>`) if required, verify the list of issues in the Completed progress state, select the **I understand that the archived issues will be removed from the Active Issue view** check box, and click **Archive Issues**.

   If you edited the default archive name, make sure that it's unique across all Kanban boards and Scrum sprint names. The name must not be longer than 255 characters.

The completed issues are archived in the specified archive name and the Issues Report of the archive opens in the **Reports** view.

Review Issue Reports of a Kanban Board

Various reports are available in the Reports view of a Kanban board.

Report	Description
Burndown Chart	Displays the remaining amount of work to be finished in issues or epic.
Cumulative Flow Chart	Displays the total number of issues in each of the board's progress states over time.
Control Chart	Displays information about issue's progress state change event on the timeline.
Issues Report	Displays active and archived issues.
Epic Report	Displays complete, incomplete and open stories of an epic.
Velocity Report	Displays the velocity chart of completed issues.

See Review Agile Reports and Charts.

Review Agile Reports and Charts

Various types of reports and charts are available for your Scrum and Kanban boards.

Burndown Chart

Use the Burndown Chart to view the remaining amount of work to be finished in issues or epics.

The Burndown Chart is available in Scrum boards and Kanban boards for issues and epics.

1. Open the board.
2. Click **Reports**.
3. Click **Issues** or **Epic** .
4. Click the **Burndown Chart** tab.

Scrum Boards

For a Scrum board, the chart for your active sprint is displayed.

- To use a different sprint, click the **Sprint** drop-down.
- To use a different estimate criterion, click the **Burndown** drop-down and select from **Estimated Days**, **Story Points**, or **Number of Issues**. The Y-axis in the chart reflects this setting.

The Burndown Chart displays the configured Tracking Statistic for the active sprint, start and end dates, capacity for the sprint, and a guideline for completing the statistic you're tracking in the sprint. The horizontal axis in a Burndown Chart tracks time, and the vertical axis represents your configured Tracking Statistic: story points, estimation days, or number of issues. Use a Burndown Chart to see the total work remaining and to increase the accuracy of predicting the likelihood of achieving the sprint goal. By tracking the remaining work throughout the iteration, your team can manage its progress and respond appropriately if things don't go as planned. If Time Tracking is enabled for the board, the Burndown Chart always shows Remaining Days and Time Spent.

The Burndown Chart includes all issues in the sprint, those that've been completed and those that are pending. The mapping of statuses to your board determines when an issue is considered Not Completed or Completed.

At the bottom of the page there's a historical table of events associated with the sprint, including issues that're incomplete at the end of the sprint.

Kanban Boards

For a Kanban board, the chart for your active issues is displayed. To choose an archived issue version, click the **Issues** list and select the desired option.

The Burndown Chart displays the configured Tracking Statistic for the active issues. The horizontal axis in a Burndown Chart tracks time, and the vertical axis represents your configured Tracking Statistic: story points, estimation days, or number of issues. Use a Burndown Chart to see the total work remaining and to increase the accuracy of predicting the likelihood of achieving the goal. By tracking the remaining work throughout the iteration, your team can manage its progress and respond appropriately if things don't go as planned. If Time Tracking is enabled for the board, the Burndown Chart always shows Remaining Days and Time Spent.

The Burndown Chart includes all issues, completed and pending. The mapping of statuses to your board determines when an issue is considered Not Completed or Completed.

At the bottom of the page there's a historical table of events associated with the issues, including issues that're incomplete.

Sprint Report

Use the Sprint Report to view completed and not completed, or open, issues of a sprint.

The Sprint Report is available for sprints in Scrum boards only.

1. Open the board.
2. Click **Reports**.

3. If necessary, click .
4. Click the **Sprint Report** tab. The Sprint Report Chart for your active sprint is displayed.
   - To select a different sprint, from the **Sprint** drop-down list, select the sprint.
   - To select a different estimate criterion, from the **Burndown** drop-down list, select **Estimated Days**, **Story Points**, or **Number of Issues**. The Y-axis in the chart reflects this setting.

The Sprint Report provides a day-by-day progress report with much of the same information that's in the Burndown Chart, albeit in a slightly different format. The Sprint Report shows the list of issues in each sprint. It can provide useful information for your Sprint Retrospective meeting and for mid-sprint progress checks. The mapping of statuses to your board determines when an issue is considered "Completed" or "Not Completed". If Time Tracking is enabled for the board, the Sprint Report chart will always show Remaining Days and Time Spent.

At the bottom of the page, the Sprint Report displays tables of completed, open, and removed issues.

Issues Report

Use the Issues Report to view active issues and archived issues.

The Sprint Report is available for issues in Kanban boards only.

1. Open the board.
2. Click **Reports**.
3. If necessary, click **Issues** .
4. Click the **Issue Report** tab. The Issue Report Chart of active issues is displayed.

   To select an archived issue version, click the **Issues** drop-down list and select the desired option.

The Issues Report provides a day-by-day progress report with much of the same information that is in the Burndown Chart, albeit in a slightly different format. The Issues Report shows the list of active issues and completed issues. If Time Tracking is enabled for the board, the Issue Report chart will always show Remaining Days and Time Spent.

At the bottom of the page, the Issues Report displays a table of issues. If **Active Issues** (default) is selected in the **Issues** drop-down list, the table lists completed and non-completed issues. If an archive is selected in the **Issues** drop-down list, the table lists only completed issues.

For each issue, the report shows the original estimate value and modified values in Estimated Days (or Story Points).

Epic Report

Use the Epic Report to view completed and not completed, or open, stories of an epic.

The Epic Report is available in Scrum boards and Kanban boards for epics.

1. Open the board.

2. Click **Reports**.
3. Click **Epic** .
4. Click the **Epic Report** tab.

For Scrum Boards

The Epic Report Chart for your active sprint is displayed.

- To select a different epic, from the **Epic** drop-down list, select the epic.
- To select a different estimate criterion, from the **Burndown** drop-down list, select **Estimated Days**, **Story Points**, or **Number of Issues**. The Y-axis in the chart reflects this setting.

The Epic Report provides a day-by-day progress report with much of the same information that's in the Burndown Chart, albeit in a slightly different format. The Epic Report shows the list of stories (or sub-issues) in each epic. It can provide useful information for your Epic Retrospective meeting and for mid-sprint progress checks. The mapping of statuses to your board determines when a story is considered "Completed" or "Not Completed". If Time Tracking is enabled for the board, the Epic Report chart will always show Remaining Days and Time Spent.

At the bottom of the page, the Epic Report displays tables of completed, open, and removed stories.

For Kanban Boards

The Epic Report Chart for your active issues is displayed. To select an archived issue version, click the **Issues** list and select the desired option.

The Epic Report provides a day-by-day progress report with much of the same information that's in the Burndown Chart, albeit in a slightly different format. The Epic Report shows the list of stories (or sub-issues) in each epic. It can provide useful information for your Epic Retrospective meeting and for progress checks. The mapping of statuses to your board determines when a story is considered "Completed" or "Not Completed". If Time Tracking is enabled for the board, the Epic Report chart will always show Remaining Days and Time Spent.

At the bottom of the page, the Epic Report displays tables of completed, open, and removed stories.

Velocity Report

Use the Velocity Report to view the velocity chart of completed sprints.

The Velocity Report is available for Scrum boards only.

1. Open the board.
2. Click **Reports**.
3. Click **Velocity** .
4. From the **Estimation** drop-down list, select **Estimated Days**, **Story Points**, or **Number of Issues**.
5. Depending on the value selected in **Estimation**, Velocity Chart is displayed for Committed and Completed values.

If Story Points is selected in **Estimation**, then the chart also shows the **Suggested Capacity** for each sprint as a dashed horizontal line.

The Velocity Report shows a Velocity chart, which shows a graph of the last seven completed sprints for the selected estimation. It also shows a table listing the completed sprints, the number of issues in each sprint, estimated values committed, and estimated values completed. Active sprints aren't shown or listed.

Using the Velocity Report, you can plan the amount of work that can be committed to future sprints. Managers can see whether the team met the original estimation and can plan the effort required for new or future sprints.

At the bottom of the page, the Velocity Report displays the sprint table. The columns change depending on the value selected in **Estimation**.

Cumulative Flow Chart

Use the Cumulative Flow Chart to view the total number of issues in each of the board's progress states over time.

The Cumulative Flow Chart is available in Scrum boards and Kanban boards for issues only.

1. Open the board.
2. Click **Reports**.
3. If you are using a Scrum board, click **Sprint**  . If you're using a Kanban board, click **Issues**  .
4. Click the **Cumulative Flow Chart** tab.

Scrum Boards

The Cumulative Flow Chart displays the total number of issues in each of the board's progress states over time for the active sprint. The issues listed in the chart are the same issues displayed in the Sprint Report.

These events can change the number of issues in a progress state:

- An issue is added to the sprint
- An issue is removed from the sprint
- An issue's progress state in the sprint changes because its status or resolution was changed

The progress states correspond to the board's current list shown in the Configure Board page. The chart is a stacked area chart enabling the user to view the number of issues in each progress state and also the total number of issues in the sprint at any given point on the timeline. The color for each progress state is randomly assigned. The user can also show or hide any of the progress states on the chart by clicking on their names in the chart legend. The events table has a column for each progress state and shows the number of issues for each progress state that's affected by the event.

Kanban Boards

The Cumulative Flow Chart displays the total number of issues in each of the board's progress states over time for the Active Issues or an archive. The issues listed in the chart are the same issues displayed in the Issues Report.

These events can change the number of issues in a progress state:

- An issue is added to active issues or an archive
- An issue is removed from active issues or an archive
- An issue's progress state in active issues or an archive changes because its status or resolution was changed

The progress states correspond to the board's current list shown in the Configure Board page. The chart is a stacked area chart enabling the user to view the number of issues in each progress state and also the total number of issues in active issues or archive at any given point on the timeline. The color for each progress state is randomly assigned. The user can also show or hide any of the progress states on the chart by clicking on their names in the chart legend. The events table has a column for each progress state and shows the number of issues for each progress state that's affected by the event.

Control Chart

Use the Control Chart to view information for issues about their progress state changed event on the timeline.

The Control Chart is available in Scrum boards and Kanban boards for Sprint Report and Issues Report. The Control chart is a scatter chart and each point on the chart represents a progress state changed event on the timeline. The progress state changed event occurs when an issue is moved from one progress state to another progress state.

1. Open the board.
2. Click **Reports**.
3. If you are using a Scrum board, click **Sprint**  . If you are using a Kanban board, click **Issues**  .
4. Click the **Control Chart** tab.

The Y-axis of the chart shows the number of days that an issue spent in its previous progress state (the progress state from which the issue was moved). For example, if the status of an issue was changed on Jan 10 from **To Do** to **In Progress**, the chart displays a point on Jan 10 and the Y-axis shows the number of days the issue was in the **To Do** progress state.

The colors for the progress states are randomly assigned and the user can hide or show the points for each progress state by clicking on the names of the progress states in the chart legend. Click the chart legend to display a line on the chart that shows the average number of days spent in a progress state.

The events table has a column for each progress state in the board, where the values are the number of days spent in the progress state at the time of each event. Note that the average number of days isn't displayed in the table as it's displayed in the chart.

4

Use Project's Repositories

In a project, you can use Git repositories to access and upload your source code files. Use the Maven repository to upload library files and dependencies.

This table describes the Oracle Developer Cloud Service pages you'd use to access repositories of a project.

Use this page ...	To:
Git	Add and manage project's Git source code repositories. You can view files and commits, manage branches and tags, and compare the source code of files.
Maven	View, upload, and search artifacts in the project's Maven repository.

Manage Code Files Using Git Repositories

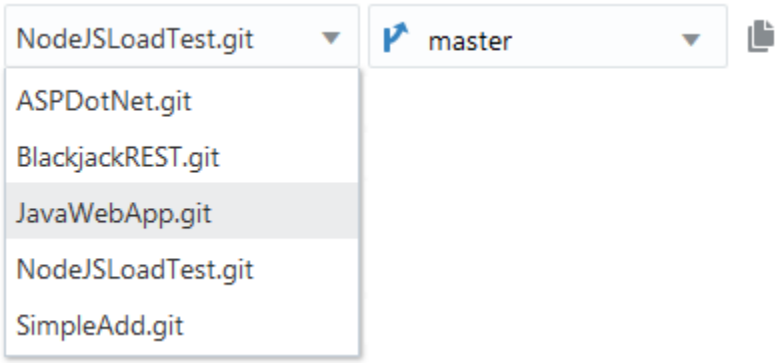
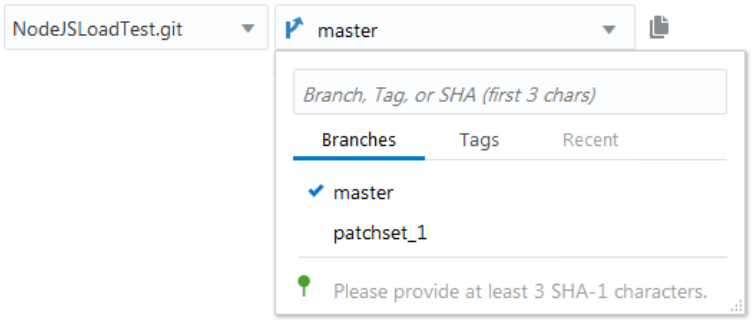
A project uses Git repositories hosted on Oracle Cloud to store and version control your application's source code files.

Git Concepts and Terms

Git is a distributed version control in which you clone the entire remote (or central) repository, including its history to your computer. You add and commit the files on your computer and, when you're done, push the commits to the remote repository.

If you are new to Git, read the Git documentation at `https://git-scm.com/book/` and `http://git-scm.com/doc` to learn more about Git repositories and Git basics, such as remote repositories, cloning, commits, pushes, SHA-1 checksum hashes, branches, and tags.

Here are the terms that this documentation uses to describe the Git terms and components of a project.

Term	Description
Project Git repository	A remote or hosted Git repository of your project. A project can host multiple Git repositories. You can view all Git repositories from the **Repositories** drop-down list on the Git page. 
Local Git repository	A cloned Git repository on your computer.
External Git repository	A Git repository that's hosted outside the project. It could be a Git repository of another project, or a Git repository available on another platform, such as GitHub or Bitbucket.
Revision	A snapshot of the Git repository at a given time. The revision could be a branch, tag, or a commit. The **Revisions** menu displays the revisions (branches, tags, and commits) of the selected Git repository.  When entering a search criteria, add a space at the end of the search term to search for an exact match. To search for a commit, in the search box at the top of the menu, enter the first three characters of the SHA-1 checksum hash of the commit. The commit that matches the search term appears next to **Commit**  at the bottom of the menu. To copy the revision name to the clipboard, click **Copy**  . For example, while viewing files of a particular commit, you can copy the SHA-1 checksum hash of the commit.

Term	Description
Files view	Display the Git repository's files and allows you to manage them.
Logs view	Displays the Git repository's commit history in a list and graphical format.
Refs view	Displays the Git repository's branches and tags and allows you to manage them.
Compare view	Compares and displays the differences between two revisions of a Git repository.

Migrate to Git

If you've been using a version control system, such as CVS or Subversion and want to migrate to Git, you can use the Git commands in the Git command-line interface to migrate.

To migrate from ...	Use this command:
CVS	`git-cvsimport` For more information, see `http://git-scm.com/docs/git-cvsimport`.
Subversion (SVN)	`git svn` For more information, see `http://git-scm.com/docs/git-svn`.
Other version control system	See the Git Book at `http://git-scm.com/book/es/v2/Git-and-Other-Systems-Migrating-to-Git`.

Set Up a Git Repository

To set up a Git repository for your project, create a repository in the project, and then upload application files to it. After you've set up the repository, all project users can access its files.

Create and Manage Git Repositories

You can add multiple Git repositories to a project. You can create an empty Git repository, a Git repository with a readme file, or import files from another Git repository.

You may want to create an empty repository if you plan to upload your application files to it from your computer or import files from another Git repository. Some IDEs and Git clients can't clone an empty Git repository. If this is the case with the IDE or Git client you use, you may want to create a Git repository initialized with a file.

You must be assigned the project **Owner** role to create and manage Git repositories.

You can create a Git repository from these pages:

- **Repositories** tab of the **Project Home** page
- **Git** page

- **Project Administration** : **Repositories** page

Create an Empty Git Repository

1. Click **+ Create Repository**.
2. In the New Repository dialog box, in **Name** and **Description**, enter a unique name and a description.

   After you create a repository, you can't change its name.
3. In **Initial Content**, select the **Empty Repository** option.

   To initialize the repository with a file, select the **Initialize repository with README file** option.
   You can add to and format the contents of the readme file using the Markdown markup language. If you don't want to keep the file after DevCS creates the repository, you can delete it.
4. Click **Create**.

Import an External Git Repository

If you've been using a Git repository on another platform such as GitHub or Bitbucket, you can import files from the external Git repository to your project's Git repository.

When you import an external Git repository, DevCS creates a Git repository in the project and copies the branches, tags, and commit history to it from the external Git repository. No changes made to the external Git repository after the import succeeds are reflected in the imported Git repository.

1. Click **+ Create Repository**.
2. In the New Repository dialog box, in **Initial content**, select **Import existing repository**.
3. In the text box, enter the URL of in the external Git repository.

   If the imported Git repository is password protected, enter the repository credentials in **Username** and **Password**. Note that DevCS doesn't store the credentials.
4. Click **Create**.

You can also import an existing Git repository to an empty project Git repository from the Git page. If the added hosted Git repository is empty, enter the Git repository's URL in the **Import existing repository** section of the Git page. Enter repository credentials, if required, and click **Import**.

Mirror an External Git Repository

If you've been using a Git repository on another platform, such as GitHub or Bitbucket, and don't want to import it to a project's Git repository, you can mirror it in DevCS. Mirroring copies the repository to DevCS and then DevCS automatically synchronizes its files time to time. In an active DevCS project, the repositories are synchronized approximately every five minutes, but the duration may vary as it depends on the number of external Git repositories in all projects of the DevCS organization.

Note that you can't add or update files or manage branches of a mirrored Git repository from the **Git** page of the project.

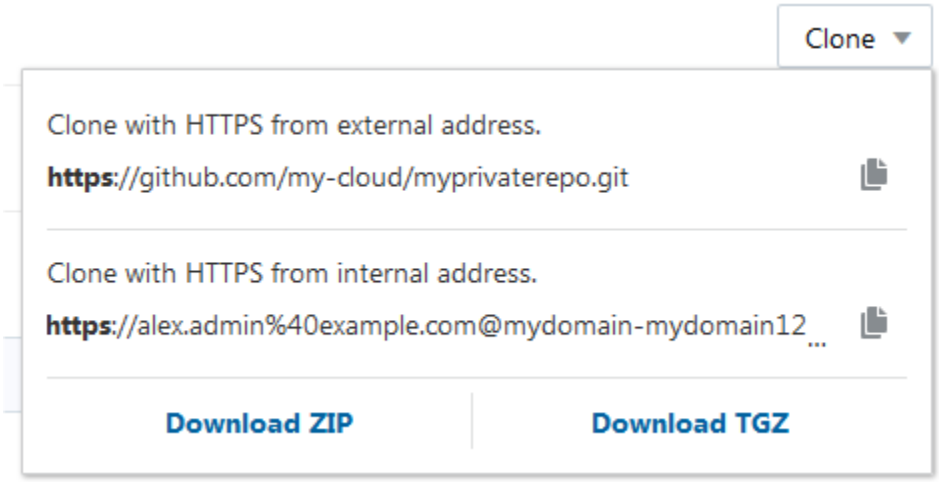
If the external Git repository is a private repository (or password protected), you must create an authentication token, such as GitHub's personal access token or BitBucket's App Password, and use it to provide access to the external Git repository. Don't provide your account's password.

1. In the navigation bar, click **Project Administration** .
2. Click **Repositories**.
3. Under **External Repositories**, click **+ Link External Repository**.
4. In the New Repository dialog box, enter the URL of the external Git repository in **URL** and the repository description in **Description**.
5. Expand the **Credentials for non-public repos** section and provide the credentials to access the external Git repository.

   In **Username**, enter the username of the external repository account. In **Token**, enter the authentication token.
6. Click **Create**.

The external repository is now available on the **Git** page and the **Repositories** tab of the **Project Home** page.
When you add an external Git repository, DevCS shows two URLs in the **Clone** drop-down menu of the repository.

Example:



Use the external address URL (the first URL in the menu) to access the repository directly. You may want to use it to access the repository's updates immediately, but would need to enter credentials to access a private repository. Use the internal address URL (the second URL in the menu) to access the mirrored repository. You may want to use it to access a private repository as it doesn't require credentials.

Manage a Git Repository

After you've created a Git repository, you can edit its description, set its default branch, index it, and delete it. However, you can't change its name.

Action	How To
Edit a Git repository's description	On the Git page, from the **Repositories** drop-down list, select the Git repository. In the **Files** or **Logs** view, click the repository description to edit it. Alternatively, on the **Project Settings**: **Repositories** page, mouse over the Git repository name, click **Options**, and select **Edit**. In the **Description** field of the Edit Repository dialog box, enter or edit the repository description, and click **Update**.
Set the default branch	When you open a Git repository on the Git page, the contents of the default branch are displayed. By default, the `master` branch of a Git repository is set as the default branch. However, you can set any branch as the default branch of a Git repository. On the **Project Settings**: **Repositories** page, mouse over the Git repository name, click **Options**, and select **Edit**. From the **Default Branch** drop-down list of the Edit Repository dialog box, select the branch, and click **Update**.
Index a Git repository	Indexing a Git repository creates or updates the Git repository index file with the latest changes. A Git index file is a binary file that serves as a virtual staging area for the next commit and contains a sorted list of object path names, along with permissions and the SHA-1 of a blob object. To index a repository, on the **Project Settings**: **Repositories** page, mouse over the Git repository name, click **Options**, and select **Index**.
Delete a Git repository	On the **Project Settings**: **Repositories** page, mouse over the Git repository name, click **Options**, and select **Delete**. In the Remove Repository dialog box, click **Yes**.

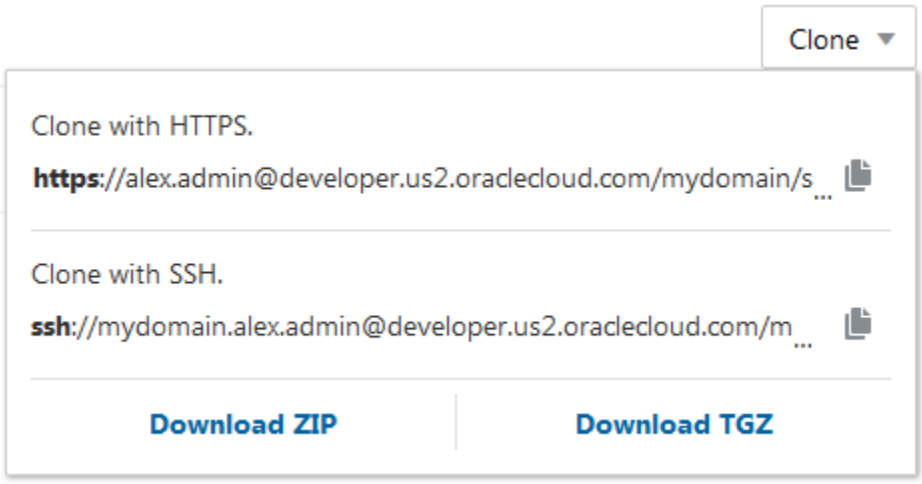
Upload Files From Your Computer to the Project's Git Repository

To upload your application source code files from your computer to the project's Git repository, you can start by using a Git client to clone the project's Git repository to your computer. After cloning, you can add files, commit the changes to the cloned Git repository, and then push the commit to the project's Git repository.

1. Copy the project Git repository's URL.

   On the Git page, from the **Repositories** drop-down list, select the Git repository.

   From the **Clone** drop-down list, click **Copy to clipboard** to copy the HTTPS or the SSH URL.



2. Open the Git client. For example, the Git CLI.
3. Navigate to the directory where you want to clone the remote Git repository.
4. Using the Git client, clone the project's Git repository.

   For example, if you're using the Git CLI, use the `git clone <repository-url>` command. Use the Git repository's URL copied from step 1.

   HTTPS example:

   ```
   git clone https://john.doe%40oracle.com@developer.us.oraclecloud.com/
   developer1111-usoracle22222/s/developer1111-
   usoracle22222_myproject/scm/developer1111-usoracle22222_myproject.git
   ```

   SSH example:

   ```
   git clone ssh://usoracle22222.john.doe
   %40oracle.com@developer.us.oraclecloud.com/developer1111-
   usoracle22222_myproject/developer1111-usoracle22222_myproject.git
   ```

5. Open the directory to access files.

   You'd notice a `.git` subdirectory in the repository directory. Don't add, delete, or modify the files of the `.git` subdirectory.

6. Copy your application files to the cloned Git repository directory.
7. To add new files to the repository, use the Git client to add them to the repository index.

   For example, if you're using the Git CLI, use the `git add` command.

   ```
   git add readme.txt
   ```

   To add a directory and its files, navigate to the directory and use `git add .`
8. Commit the updated files to the cloned Git repository.

   For example, if you are using the Git CLI, use the `git commit` command to save the changes.

   ```
   git commit -am "Sample comment"
   ```

9. Push the commit from the cloned Git repository to the hosted Git repository.

   For example, if you are using the Git CLI, use the `git push` command.

```
git push origin master
```

See this Tutorial to set up a Git repository and add files to it using the Git CLI.

Push a Local Git Repository to the Project's Git Repository

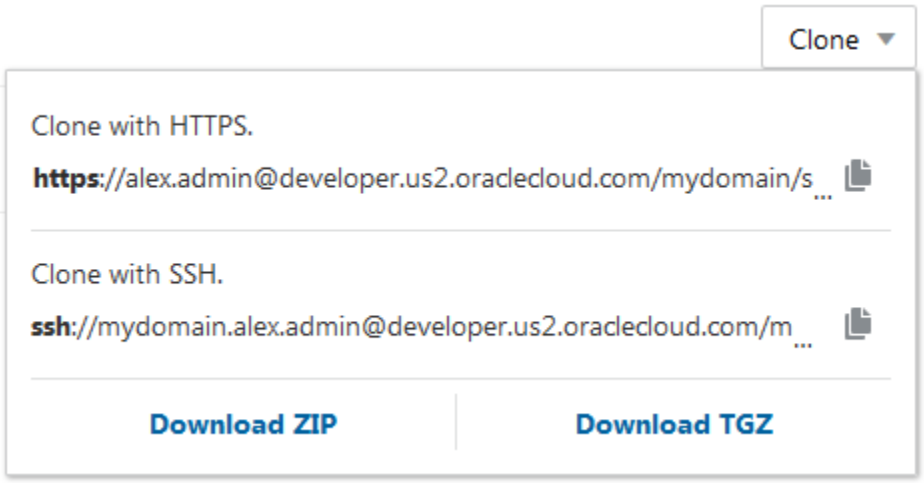
If your application source code files are available in a local Git repository, you can push them to a project's empty Git repository.

You can use any Git client to push the local Git repository to the remote Git repository.

1. Copy the URL of the project's Git repository.

   On the Git page, from the **Repositories** drop-down list, select the Git repository.

   From the **Clone** drop-down list, click **Copy to clipboard** to copy the HTTPS or the SSH URL.



2. Open the Git client. For example, the Git CLI.
3. Navigate to the local Git repository directory.
4. Add the project's Git repository as the remote repository of the local repository. Use the Git repository's URL copied from step 1.

   For example, if you're using the Git CLI, use the `git remote add <remote-repository-name> <repository-url>` command.

   HTTPS example:

   ```
   git remote add origin https://john.doe
   %40oracle.com@developer.us.oraclecloud.com/developer1111-
   usoracle22222/s/developer1111-usoracle22222_myproject/scm/
   developer1111-usoracle22222_myproject.git
   ```

   SSH example:

   ```
   git remote add origin ssh://usoracle22222.john.doe
   %40oracle.com@developer.us.oraclecloud.com/developer1111-
   usoracle22222_myproject/developer1111-usoracle22222_myproject.git
   ```

   The above example adds a remote named `origin` for the repository at `developer.us.oraclecloud.com/developer1111-usoracle22222_myproject/developer1111-usoracle22222_myproject.git`.

5. Push the local Git repository to the project's Git repository.

   For example, if you're using the Git CLI, use the `git push` command.

   ```
   git push –u origin master
   ```

6. In your project, open the Git page and check the files in the project's Git repository.

Access a Git Repository using SSH

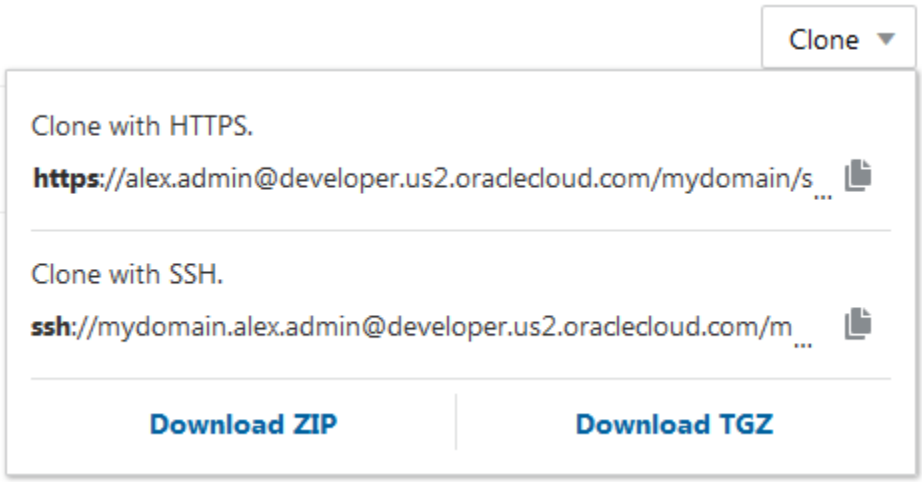
1. On the computer that you'll use to access the Git repository, generate a SSH key pair and upload its private key to DevCS. See Upload Your Public SSH Key. Make sure that the private key on your computer is accessible to the Git client.

   Ignore this step if you've already uploaded the SSH public key.

2. Copy the Git repository's SSH URL.

   On the Git page, from the **Repositories** drop-down list, select the Git repository.

   From the **Clone** drop-down list, click **Copy to clipboard** to copy the SSH URL.



3. Open the Git client. For example, the Git CLI.
4. Navigate to the directory where you want to clone the remote Git repository.
5. Using the Git client, clone the project's Git repository.

   For example, if you're using the Git CLI, use the `git clone <repository-ssh-url>` command.

   Example:

   ```
   git clone ssh://usoracle22222.john.doe
   %40oracle.com@developer.us.oraclecloud.com/developer1111-
   usoracle22222_myproject/developer1111-usoracle22222_myproject.git
   ```

   If you've already cloned the Git repository to your computer using HTTPS, use the `git add remote` command to add the SSH URL of the Git repository.

   Example:

   ```
   git remote add ssh-origin ssh://usoracle22222.john.doe
   %40oracle.com@developer.us.oraclecloud.com/developer1111-
   usoracle22222_myproject/developer1111-usoracle22222_myproject.git
   ```

6. Commit the updated files to the cloned Git repository.
7. Push the commit from the cloned Git repository to the hosted Git repository.

   Example:

   ```
   git push ssh-origin master
   ```

Add and Manage Files of a Git Repository

You can add and update files of a Git repository online from the Git page or clone the Git repository to your computer and update the files locally.

Manage Files from the Git Page

You can browse, add, edit, view commit history, rename, and delete files of a Git repository. You can't add or update files of a linked external Git repository.

You must be a project member to add or updates files of a Git repository.

1. In the navigation bar, click **Git**.
2. From the **Repositories** drop-down list, select the Git repository. From the **Revisions** drop-down list, select the branch.
3. On the right side of the page, click **Files**, if necessary.
4. Browse and click a directory name to open it.
   To go back to a file or a sub-directory of a parent directory, click ***/*** and select the file or directory from the menu. To go to the root directory, click . To copy the path of a file or a directory, click **Copy to clipboard** .

Action	How To
Add a file	1. Click **+ File**. 2. In **File Name**, enter the name of the file along with its extension. 3. In the code editor, enter file contents. 4. Click **Commit**. 5. In the Commit Changes dialog box, enter the commit summary in the first text box and details in the **Details** text box, and click **Commit**. To save the file in a new directory or a directory structure, in **File Name**, enter the file path. You can enter a relative path or an absolute path. To specify an absolute path, add a `/` in the beginning. Example: • Enter `test/text_file.txt` to save the `text_file.txt` file in the `test` directory on the current path. If the `test` directory doesn't exist, it's created. • Enter `/test/text_file.txt` to save the `text_file.txt` file in the `test` directory on the root. If the `test` directory doesn't exist, it's created.

Action	How To
View a file	To view contents of a file, in the **Files** view, browse, and click the file name link. The file opens in the **File** view of the Git page. If you open a text file or an image file, its contents are displayed in a read-only editor. Contents of a binary file aren't displayed. If the text content exceeds the width of the editor, use the arrow keys to scroll left, right, up, and down. You can also use the scroll buttons to scroll horizontally. Move the cursor to the left or the right edge of the editor and click **Right Scroll** or **Left Scroll** to scroll one character at a time. To view the file in raw (unformatted) format in the web browser, click **Edit**, and select **Raw**. The contents of the opened file are displayed in a new tab or a window of the web browser. If the file is a text file or an image (such as `.png`, `.jpg`, `.bmp`, and `.gif`), it's displayed in the browser. The contents of a binary file such as .zip and .rar aren't displayed, but you can use the browser URL to download it.
View annotations and commits of a file	Open the file and click **Blame**. The Blame view displays annotations of the open file for each updated code line (or group of code lines) with commit information. The annotation includes commits that affected code lines, author, the date-time stamp of the commit, and the commit message.
Edit, rename, or move a file	Open the file and click **Edit**. Edit the file's contents in the code editor. To rename the file or move it to another directory, in the file name text box, enter the new name or path. Click **Commit** to save.
Delete a file	To delete a file, click **Actions** next to **Edit**, and select **Delete**. In the Commit Changes dialog box, enter the commit summary in the first text box and details in the **Details** text box, and click **Commit**.

Use Git Commands to Manage Files from Your Computer

To access and manage your project's Git repository files from your computer, use a Git client, such as the Git CLI.

This table lists common Git commands that you can run in the Git CLI to work on files in your local Git repository.

Run this command ...	To:
`git clone <repository-url>`	Clone a project's Git repository to your computer. Example: `git clone https://john.doe%40example.com@developer.us.oraclecloud.com/developer1111-usoracle22222/s/developer1111-usoracle22222_myproject/scm/developer1111-usoracle22222_myproject.git`

Run this command ...	To:
`git add <filename>`	Add a file that you've added to the repository's directory to the repository's index. Example: `git add readme.txt` To add all new files to the index, use `git add --all` To add a directory and its contents to the index, navigate to the directory and use `git add .`
`git rm <filename>`	Remove a file from the repository. Example: `git rm readme.txt`
`git status`	Check the status of added and edited files. Example: `git status`
`git branch`	Create a branch. Example: `git branch new_branch` To list all existing branches of the repository, use `git branch`.
`git checkout`	Switch to (or checkout) a branch. Example: `git checkout new_branch` You can use the `git checkout -b` command to create a branch and switch to it immediately. Example: `git branch -b new_branch`
`git merge`	Merge a branch with the checked out branch. Example: `git merge new_branch`
`git commit`	Commit changes to the local Git repository. Example: `git commit -m "Initial commit"`
`git push`	Push commits to the project's Git repository. Example: `git push -u origin master`
`git pull`	Incorporate changes from the project's Git repository to the local Git repository. Example: `git pull origin master`

To display the Git help index, run the `git help` command. Run the `git help git` command to open the help index in a web browser. To display help for a particular command, run `git help <command>`.

Associate a DevCS Issue with a Commit

When you save changes to a Git repository, you may want to link a DevCS issue that's assigned to you with the commit.

To associate an issue with a commit, add `Task-URL: <issue-url>` in the commit message.

Example:

```
Update for Defect 4 Task-URL:https://john.doe
%40oracle.com@developer.us.oraclecloud.com/developer1111-usoracle22222/s/
developer1111-usoracle22222_myproject/task/4
```

If the commit is successful, the SHA-1 checksum hash of the commit is added to the issue. Open the issue in Issues page and verify the SHA-1 checksum hash in **Commits** under **Associations**.

Copy the URL of a Git Repository or a File

From the Git page, you can copy and share the URL of a Git repository, the URL of a file in the Git repository, and the URL of a line in a file of the Git repository.

Before you share the URL, remember that only members of the project can use the URL to access the file or clone the repository. If the project is a shared project, organization members can access files in the project's repository or clone the repository, but they can't update them.

Action	How To
Copy the URL of a Git repository	To clone a Git repository or to access it using a Git client, you use the URL of the repository. You can copy the URL from the **Repositories** tab of the **Project Home** page, the **Git** page, and from the **Project Settings**: **Repositories** page. In the **Repositories** tab of the **Project Home** page or the **Project Settings**: **Repositories** page, search for the Git repository, and click the **Clone** drop-down list to see the HTTPS and SSH URLs of the repository. To the right of the URL, click **Copy** to copy the URL to clipboard. You may also select the URL and press `Ctrl + C` or use the mouse context menu to copy the URL to the clipboard. The SSH URL of an external Git repository isn't available.
Copy the URL of a file	In the **Files** view of the **Git** page, open the file. From the address bar of the browser, copy the URL.
Copy the URL of a line in a file	In the **Files** view of the **Git** page, open the file. On the left side of the line, in the number column, click the line number. The entire line is selected. From the address bar of the browser, copy the URL. Example: To copy the URL of line number 2 of `myfile.txt`, click the line number 2. Clicking the line number updates the URL in the browser's address bar to `http://developer.us2.oraclecloud.com/my-org/#projects/demo/scm/demo.git/blob/myfile.txt?revision=master&sl=2`. You can copy and use this URL to open `myfile.txt` in the `demo.git` repository – `master` branch with line number 2 selected.
Copy the URL of a group of lines in a file	In the **Files** view of the **Git** page, open the file. On the left side of the line, in the number column, click the line numbers with the `Shift` key pressed to select them. From the address bar of the browser, copy the URL. Example: With the `Shift` key pressed, clicking line numbers 2 through 5 of `myfile.txt` selects those lines. The URL in the browser's address bar changes to `http://developer.us2.oraclecloud.com/my-org/#projects/demo/scm/demo.git/blob/myfile.txt?revision=master&sl=2-5`. Copy the URL and share it with project members. When the URL is entered, the `myfile.txt` file of the `demo.git` repository – `master` branch opens with line numbers 2 through 5 selected.

View the History of Files and Repositories

You can use the **Logs** view of the **Git** page to view the commits, branching, and merging history of a file or Git repository and its revisions.

1. From the **Repositories** drop-down list, select the Git repository. From the **Revisions** menu, select the branch.
2. To view the commit history of a file, browse, and open the file.

   Skip this step to view the commit history of the selected Git repository.
3. On the right side of the page, click **Logs**.

Action	How To
View the commit history in a list format	In the **Logs** view, click the **History List** toggle button. To view the history of another branch or tag, in the text box to the right of the **History** toggle button, enter branch or tag names. You may also click the text box and select the revisions from the drop-down list. You can add multiple branches or tags. To view the history of all revisions of the selected Git repository, remove all revision names from the text box.
View the commit history as a graph	In the **Logs** view, click the **History Graph** toggle button. In the graph: • Each dot represents a commit. To see the details of the commit, click the dot. • A splitting line represents a branch. • Joining lines represent a merge. • Latest commits appear at the top of the graph.

Use Branches

Branching lets you work on different features and updates at any time without affecting the original source code.

Before you start working on a new feature or update major portions of the source code, it's considered a good practice to create a branch and commit your changes to the new branch. This way your changes don't affect the original source code and are safe to test and review. To learn more about the Git branch workflow, read the *Git Branching - Branching Workflows* topic in the Git book at `https://git-scm.com/book/en/v2/`.

By default, all Git repositories have one default `master` branch. However, you can add more branches to the repository. You can also subscribe to email notifications for commits made to the repository's branches, and you can compare, rename, and delete branches.

Create a Branch

From the **Refs** view of the **Git** page, you can create a branch from the base branch, from the head (tip) of an existing branch, or from a tag.

You can't create a branch in an empty Git repository. You must first clone the repository to your computer, add and commit files to the default `master` branch that's automatically created, and then push the branch to the project Git repository. After the default `master` branch is pushed to the project Git repository, you can create more branches.

You can also mark a branch as a private branch. Only branch owners can push commits to a private branch.

You must be a project member to to create a branch.

Action	How To
Create a branch from a base branch	1. In the **Refs** view of the **Git** page, click **Branches** . 2. From the **Repositories** drop-down list, select the repository. 3. Click **+ Create Branch**. 4. In the New Branch dialog box, in **Name**, enter the branch name. From the **Base** drop-down list, select the base revision name. 5. To mark the branch as a private branch, select the **Private Branch** check box. 6. Click **Create**.
Create a branch from the head (tip) of another branch	1. In the **Refs** view of the **Git** page, click **Branches** . 2. From the **Repositories** drop-down list, select the repository. 3. Click **+ New Branch**. 4. In the branch list, to the right of source branch name, click **Actions** , and select **Branch**. 5. In the New Branch dialog box, enter the name of the new branch. 6. To mark the branch as a private branch, select the **Private Branch** check box. 7. Click **Create**.
Create a branch from a tag	1. In the **Refs** view of the **Git** page, click **Tags** . 2. From the **Repositories** drop-down list, select the repository. 3. Click **+ New Branch**. 4. In the tags list, to the right of the tag name, click **Actions** and select **Branch**. 5. In the New Branch dialog box, enter the name of the new branch. 6. To mark the branch as a private branch, select the **Private Branch** check box. 7. Click **Create**.

Protect a Branch

By default, any project member can rename or delete a repository branch, and push or merge another branch into it. To protect a branch from these actions, set restrictions on the branch.

You must be assigned the project **Owner** role to set restrictions on a branch.

1. In the navigation bar, click **Project Administration**.
2. Click **Branches**.
3. In **Repository** and **Branches**, select the Git repository and the branch.
4. On the Branches page, set the restrictions.

> **Tip:**
>
> On the **Refs** page, you can also click the **Open**, **Private**, **Requires Review**, or the **Frozen** label of a branch to edit its protection settings.

This table describes the various branch protection actions you can perform from the Branches page.

Action	How To
Requires review and restrict merge actions	Select the **Requires Review** option and configure the review options. See Set Review and Merge Restrictions on a Repository Branch.
Restrict push actions to project owners and branch owners	Select the **Private** option. To define branch owners, click **Owners** and select the user. You can select multiple users. Note that to push commits to a private branch from your computer, always use SSH. Also, to run a build of job that uses a private branch, configure the job to use SSH.
Lock a branch	Select the **Frozen** option. No changes are allowed to a locked branch by any user.
Prevent forced pushes to the branch	Select the **Do not allow forced pushes** check box. The check box isn't available when the **Requires Review** or the **Frozen** option is selected as force push aren't allowed on a review or a frozen branch.
Prevent the rename and delete actions on the branch	Select the **Do not allow renaming and deleting branch** check box. The branch can be renamed or deleted after you deselect the check box. The check box isn't available when the **Requires Review** or the **Frozen** option is selected.

Manage a Branch

After you create a branch, you can rename it, compare it with another branch of the Git repository, or delete it.

You must be a project owner or member to edit and update a branch. You can perform the branch management actions from the **Refs** view of the **Git** page.

Action	How To
Rename a branch	You can't rename a restricted branch or the `master` branch. After renaming a branch, update all related merge requests, build jobs, and deployment configurations to use the new branch name. When you rename a branch, Git creates a branch with the new name and transfers all content from the old branch to the new branch. After the transfer is complete, the old branch is removed. 1. In the branch list, to the right of branch name, click **Actions**, and select **Rename**. 2. In the Rename Branch dialog box, in **Name**, enter the new branch name. 3. Select the **I want to rename the branch** check box and click **Rename**.
Compare a branch	In the branch list, to the right of branch name, click **Actions**, and select **Compare**. By default, the branch is compared with the `master` branch.
Protect or set restrictions on a branch	In the branch list, to the right of branch name, click **Actions**, and select **Protection Settings**.
Delete a branch	You can't delete a restricted branch or the `master` branch. After you delete a branch, you must update, close, or remove all related merge requests, build jobs, and deployment configurations. 1. In the branch list, to the right of branch name, click **Actions**, and select **Delete**. 2. In the Delete Branch dialog box, select the **I want to delete the branch** check box, and **Delete**.

Use Tags

Tagging lets you mark a specific point of time in the history of the repository. For example, you can create a tag to mark the Git repository state of an application's stable state, before a release.

Create and Manage Tags

From the **Refs** view of the **Git** page, you can create and manage a Git repository's tags.

You must be a project owner or member to create and manage a tag.

Action	How To
Create a tag	1. In the **Refs** view of the **Git** page, click **Tags**  . 2. From the **Repositories** drop-down list, select the repository. 3. Click **+ New Tag**. 4. In the New Tag dialog box, in **Name**, enter the tag name. In **Base**, enter the base revision name. Click **Create**.
Rename a tag	1. In the tags list of the **Tags**  view, to the right of the tag name, click and select **Rename**. 2. In the Rename Tag dialog box, in **New Name**, enter the new tag name, select the **I want to rename the tag** check box, and click **Rename**.
Compare a tag	In the tags list of the **Tags**  view, to the right of the tag name, click and select **Compare**. On the Compare page that opens, by default, the tag is compared with the default branch.
Delete a tag	In the tags list of the **Tags**  view, to the right of the tag name, click and select **Delete**. In the Delete Tag dialog box, select the **I want to delete the tag** check box and click **Delete**.

Compare Revisions

You can compare any two revisions of a Git repository. The base revision indicates the starting point of the comparison and the compare revision indicates the end point of the comparison. The revision could be a branch, a tag, or a commit SHA-1 checksum hash.

1. On the right side of the **Git** page, click **Compare**.
2. From the **Base Revision** drop-down list on the left, select the base revision.

   By default, the **Git** page selects the last commit of the repository as the base revision and the selected revision as the compare revision.
3. From the **Compare Revision** drop-down list on the right, select the compare revision.

You can compare these revisions:

- Branch versus branch
- Tag versus tag
- Commit versus commit
- Branch versus tag
- Commit versus branch
- Tag versus commit

On the Compare Result page, the **Changed Files** tab and the **Commits** tab. The **Changed Files** tab lists files that have changed between the base revision and the compare revision. The **Commits** tab lists all commits that have happened between the base revision and the compare revision since their common commit. The **Commits** tab is enabled if **From Merge Base** is selected in **From Merge Base or Revisions** ... .

Action	How To
Compare with a parent of the base revision	From the **Base Revision** drop-down list, click the **Parents** tab, and then click the SHA-1 checksum hash of the parent commit.
View differences between the base revision and the compare revision since the last common commit to both revisions	From the **From Merge Base or Revisions**... drop-down list, select **From Merge Base (...)** . Select **Revisions (..)** to show the differences between the heads (or tips) of the base revision and the compare revision.
Switch the base revision and the compare revision	Click **Switch Base** .
Create or open a merge request	If a merge request exists with the **Compare Revision** as the review branch, click the merge request button to open the merge request review page. If a merge request doesn't exist, click **+ Merge Request** to create a merge request with **Base Revision** as the target branch and the **Compare Revision** as the review branch.
View the compare options	Click **Diff Preferences** to view various compare options.

Add Comments to a File

While comparing files, you can add inline comments to the source code changes made to a file. The comments are visible to all users of the project.

1. Browse and open the file.
2. On the right side of the page, click **Logs**.
3. For the commit that changed the file and added the changes you want to comment on, click the button with the first seven characters of the commit's SHA-1 checksum hash as the label.
4. In the **Changed Files** tab of the Compare view, mouse-over the line number of the file and click **Add Comment**

   If you selected the **Unified** view, click the line number in the second column. If you selected the **Side by Side** view, click the line number of the file on the right.
5. In the **Leave a comment** box, enter the comment, and click **Comment**.

The comment is added as an inline comment to the file and is visible to all project members. To reply to a comment, click **Reply** , enter the comment in the **Leave a reply** box, and click **Comment**.

Watch a Git Repository

You can watch a Git repository branch and receive email notifications when any file of the branch is updated in the project's Git repository.

To get email notifications, enable them in your user preferences, and then set up a watch on the branch from the **Refs** view of the **Git** page.

Action	How To
Enable email notifications	In your user preferences page, select the **SCM/Push Activities** check box.
Watch a branch	1. Open the project. 2. In the navigation bar, click **Git**. 3. On the right side of the page, click **Refs**. 4. If necessary, click **Branches** . 5. In the branch list, to the right of the branch name, click **cc**. Alternatively, click **Actions** , and select **Subscribe**. A **Subscribed** icon appears indicating that you are subscribed to email notifications of the branch updates. To unsubscribe, click **cc** again.

Search in Git Repositories

You can search for a file name, directory name, or a term in the source code files, file paths, and file revisions of the project's Git repositories.

The search field supports common programming languages, such as HTML, JavaScript, CSS, and Java. You can use these features while searching terms:

- Language-aware
- Auto-suggest
- Symbols (class and function names) and file names
- CamelCase

Action	How To
Search for a term in a Git repository and a revision	1. From the **Repositories** drop-down list, select the Git repository. From the **Revisions** drop-down list, select the revision. 2. In the top-right corner of the page, in the **Search Code** box, enter the search term or select it from the suggestion list. 3. Click **Search** .

Action	How To
Search for a term in all revisions of a Git repository	1. From the **Repositories** drop-down list, select the Git repository. From the **Revisions** drop-down list, select the revision. 2. In the top-right corner of the page, in the **Search Code** box, enter the search term or select it from the suggestion list. 3. Click **Search**. 4. In the **Revisions** drop-down list, click **Reset**. The **Revisions** drop-down list now shows **All Revisions**.
Search for a term in all Git repositories	1. From the **Repositories** drop-down list, select the Git repository. From the **Revisions** drop-down list, select the revision. 2. In the top-right corner of the page, in the **Search Code** box, enter the search term or select it from the suggestion list. 3. Click **Search**. 4. From the **Repositories** drop-down list, select the **All Repositories** option, or click **Reset**.

The search result page displays all files, file paths, and file revisions that contain or match the search term (or symbol). To reset the search, to the left of the **Search Code** box, click **Back**.

Download an Archive of a Git Repository

If a branch or a tag of a Git repository isn't required and you plan to delete it, it's considered a good practice to create an archive of it and back it up before you delete it. From the **Refs** view of the **Git** page, you can download the archive file (zip or tgz) of a Git repository branch or a tag.

Action	How To
Download the archive of a branch	1. In the **Files** view of the **Git** page, from the **Repositories** drop-down list, select the repository. From the **Revisions** menu, select the branch. 2. From the **Clone** drop-down list, click **Download ZIP** or **Download TGZ**. Alternate method: 1. In the **Refs** view of the **Git** page, click **Branches**. 2. From the **Repositories** drop-down list, select the repository. 3. In the branches list, to the right of the branch name, click ..., select **Download**, and then select **zip** or **tgz**.

Action	How To
Download the archive of a tag	1. In the **Refs** view of the **Git** page, click **Tags**  . 2. From the **Repositories** drop-down list, select the repository. 3. In the tags list, to the right of the tag name, click , select **Download**, and then select **zip** or **tgz**.

Manage Binaries and Dependencies Using the Project's Maven Repository

When a project is created, Oracle Developer Cloud Service (DevCS) creates a hosted Maven repository in the project, which is also called as the project Maven repository. You can use the repository to store binary files and dependencies. If you're developing Maven applications, you can use the Maven repository to store and access build artifacts.

If you see the **No Storage Configured** error message, contact the Organization Administrator (a user with the **Developer Service Administrator** identity domain role) to enable the storage configuration of the organization. If you're the Organization Administrator, configure the storage configuration of the organization. See Connect to OCI or OCI Classic.

Maven Concepts and Terms

Apache Maven is a software project management tool that uses the Project Object Model (POM) concept to manage a project's build.

If you're new to Maven, see `https://maven.apache.org` to learn about Maven basics such as POM files and Maven repositories.

In DevCS, you use the project's Maven repository to store build artifacts and dependencies for your project's applications. Usually, you store the dependencies on the project's Maven repository that aren't available on a public Maven repository, such as Maven Central Repository or Oracle Maven Repository.

Here are the terms that this documentation uses to describe the Maven terms and components of a project.

Term	Description
POM file	An XML file that contains configuration about how to build the application. Usually, the file is saved as `pom.xml`. For more information, see `https://maven.apache.org/guides/introduction/introduction-to-the-pom.html`.
Browse view	Displays and allows you to browse artifacts of the project's Maven repository.
Upload view	Allows you to upload artifacts manually to the project's Maven repository.
Artifact Search view	Enables you to search artifacts in the project's Maven repository.

To upload and access the files of the repository programmatically, configure the POM file of your application. You can use the project Maven repository among other projects of the organization for local builds as well as project builds.

Upload an Artifact Manually

From the Upload view, you can upload artifacts manually to the project's Maven repository without installing Maven on your computer. You must be a project owner or member to upload an artifact to the project Maven repository.

1. In the navigation bar, click **Maven** m.
2. On the right side of the page, click **Upload**.
3. In the **Upload Artifacts** section, use the drag-and-drop operation to drop files to the drop area, or click the select artifact files link, browse, and select the files.
4. In the table below the Upload Artifacts section, if necessary, update the **Classifier** field of the selected artifact.

   If you're uploading only one artifact, you can leave the classifier field empty. If you're uploading multiple artifacts, provide the classifier value for each artifact. The classifier helps to distinguish artifacts that were built from the same POM file but differ in their content. The classifier string is appended to the artifact name, after the version number.

   For example, if you're uploading artifacts with identical names but different extensions (such as `fileX-1.0.jar` and `fileX-1.0.pdf`), you may provide classifiers, such as `main` and `documentation` for these files. After the files are uploaded, they are renamed to `fileX-1.0-main.jar` and `fileX-1.0-documentation.pdf`.
5. After you add the artifacts, you must specify their Maven coordinates manually or from a POM file. These coordinates are used when uploading artifacts to the project's Maven repository.

   If you want to specify the artifacts' Maven coordinates manually, note the following:

   - The auto-suggest list of **GroupId**, **Version**, and **ArtifactId** are based on Maven indexes. If no index data is available, the auto-suggest list isn't displayed.
   - By default, the **Generate POM** check box is selected. The upload process deploys the artifact and generates the default POM file, `maven-metadata.xml`, and associated `sha1/md5` checksum files. If `maven-metadata.xml` already exists, it's updated.

     If you deselect the check box, the upload process deploys the artifact to the target deployment path based on Maven attributes. The POM file and the `maven-metadata.xml` file aren't generated.
6. Click **Start Upload**.

   You can track the transfer status and its progress in the drop area of the Upload Artifacts section. To cancel the upload process, click **Cancel Upload**. The upload process will also be cancelled if the page is refreshed or closed.

Upload Artifacts Using the Maven Command-Line Interface

You can also use the Maven command-line interface to upload artifacts to the project's Maven repository.

The Maven repository URL is available on the **Project Home** page of your project. Use the `dav:` URL to upload files and the `http://` URL to view them in the browser.

Note that the credentials in `settings.xml` aren't required to access the project Maven repository when running a build. The build job has full access to the project Maven repository for uploads and downloads.

1. Download and install Maven on your local computer.

   You can download Maven from `http://maven.apache.org/download.cgi`.

2. Open `MVN_HOME/conf/settings.xml` in a text editor and make the following changes.

   a. Specify the proxy server, if necessary.

      Example:

      ```
      <proxies>
          <proxy>
              <active>true</active>
              <protocol>http</protocol>
              <host>PROXY_URL</host>
               <port>80</port>
               <nonProxyHosts>www.anything.com|*.somewhere.com</
      nonProxyHosts>
           </proxy>
      </proxies>
      ```

   b. Specify a unique ID and your DevCS user name and password to access the project Maven repository.

      Example:

      ```
      <servers>
          <server>
              <id>remoteRepository</id>
              <username>USERNAME</username>
              <password>PASSWORD_IN_PLAINTEXT</password>
          </server>
      </servers>
      ```

   c. Specify a unique ID, name, and URL for the project Maven repository. You can copy the Maven repository URL from the **Repositories** tab of the **Project Home** page.

      Example:

      ```
      <profiles>
          <profile>
              <id>default</id>
              <repositories>
      ```

```
            <repository>
                <id>remoteRepository</id>
                <name>My Remote Repository</name>
                <url>dav:https://developer.us2.oraclecloud.com/……/
maven/</url>
                <layout>default</layout>
            </repository>
        </repositories>
    </profile>
</profiles>
```

3. Open the command-line and follow these commands to upload files to the hosted Maven repository. Ensure that the `MVN_HOME/bin` path is available in the `PATH` variable.
   a. Navigate to the directory that contains the files that you want to upload.
   b. Create the `pom.xml` file, if it hasn't been created already.

      For more information about `pom.xml`, see `http://maven.apache.org/guides/introduction/introduction-to-the-pom.html`.
   c. Run the `mvn deploy` command to upload files.

      Example: `mvn deploy:deploy-file -DpomFile=c:\myproject\pom.xml -Dfile=c:\myproject\myfile.jar -DrepositoryId=remoteRepository -X -Durl=dav:https://developer.us2.oraclecloud.com/……/maven/`

Search Artifacts

To search for artifacts, use the Artifact Search view.

You can specify the following search criterion:

- GroupID
- ArtifactID
- Version
- Classifier
- Size and updated date (available in **Show Advanced Options**)

The search result is grouped by Maven coordinates in the **Artifacts** (default) tab and by files in the **Files** tab.

Download an Artifact Manually

You can download an artifact manually from the Browse view of the Maven page.

1. In the navigation bar, click **Maven** m.
2. If necessary, on the right side of the page, click **Browse**.
3. Browse and select the artifact that you want to download.

   You can also click **Artifact Search** to search for the artifact and then click its name to open it in the **Browse** view.
4. With the artifact selected, in Artifact Details on the right, click **Download**.

The browser downloads the artifact and saves it to your computer.

Copy Distribution Management Snippets

To upload or download dependencies while running a build, add the dependency management snippet or the dependency declaration snippet to the POM file.

Action	How To
Copy the Dependency Management snippet	1. In the Browse view, navigate to the root directory. 2. In the **Artifact Details** section, expand **Distribution Management**. 3. In the **Maven** tab, click **Copy** to copy the `<distributionManagement>` code snippet to the clipboard. 4. Open the `pom.xml` file of your project in a code editor (or a text editor) and paste the contents of the clipboard under the `<project>` element.
Copy the Dependency Declaration snippet	1. Browse the Maven repository and select the artifact. You may also click **Artifact Search** to search for the artifact and then click its name to open it in the **Browse** view. 2. In the **Artifact Details** section, expand **Dependency Declaration**. 3. In the **Maven** tab, click **Copy** to copy the `<dependency>` code snippet to the clipboard. 4. Open the `pom.xml` file of your project in a code editor (or a text editor) and paste the contents of the clipboard under the `<dependencies>` element.

Tip:

You can copy the Maven repository's URL from the distribution snippet. You can also copy the URL from the **Project Home** page.
On the **Project Home** page, click the **Repositories** tab. In the Maven section, from the **Clone** menu of **Project Repository**, select **HTTP** or **Webdav**, and click **Copy** to copy the URL to clipboard.

Use the **HTTP** URL to connect to the Maven repository using the HTTP protocol. Use the **Webdav** URL to connect to the repository using the Webdav protocol.

Maven Repository Administration

You can configure the Maven repository to limit the number of snapshots and overwrite an artifact if another with same groupID, artifactID, and version value is uploaded.

Configure Auto-Cleanup of Snapshots

By default, when you upload a new snapshot of an artifact to the project's Maven repository manually or through job builds, the repository retains the old versions of the snapshots. You can configure the project to remove the old versions when a new version is uploaded.

You must be assigned the project **Owner** role to configure the auto-cleanup.

1. In the navigation bar, click **Project Administration**.
2. Click **Repositories**.
3. In **Maven Repository**, if necessary, expand **Configure auto cleanup for Snapshot versions**.
4. Select the **Purge** check box.
5. In **Default Max Snapshots**, enter a number between 2 and 500 to specify the maximum versions to retain. By default, 2 versions of the snapshots are retained.

All changes are saved automatically when you navigate to another field. After the rule is enabled, any new upload of a snapshot will remove its older versions if the number of snapshots exceed the value in **Default Max Snapshots**.

You can also add exceptions to the auto-cleanup and **Default Max Snapshots rule** and customize the snapshot counts. The project Maven repository retains the snapshot counts of group IDs and artifacts defined in the **Customized Snapshot Counts** section and uses the default value specified in **Default Max Snapshots** for artifacts that don't have exceptions defined.

To define exceptions and customize snapshot count:

1. In the **Maven Repository** section of the Repositories page, select the **Purge** check box and configure the default auto-cleanup as described above.
2. To add an artifact group or an artifact name as an exception, click **+ Add** in the **Customized Snapshot Counts** section.
3. Specify these details:
   - **Group Id** (Required): Enter or select the Group ID of the artifact. You can select the ID from the list or start typing and then select the ID from the list of suggestions. The auto-suggest list is based on the Maven indexes. If no index data is available, the auto-suggest list doesn't display.
   - **Artifact Id** (Optional): Enter or select the Artifact ID of the artifact. You can select the ID from the list or start typing and then select the ID from the list of suggestions. The auto-suggest list is based on the Maven indexes. If no index data is available, the auto-suggest list doesn't display.
   - **Snapshot Count**: Select the number of snapshots to retain in the project Maven repository. By default, 2 snapshots are retained.

To remove an exception, on the right, click **Remove**. For a long list of exceptions, you can use the **Filter** field and enter a search coordinates criteria to see the exceptions matching the criteria. If you enter an exception with duplicate coordinates, an error message `Unable to save. Coordinates already exists. Enter unique coordinates` appears.

Configure Overwrite for Artifacts

By default, when a user tries to upload a Maven release artifact with the same groupID, artifactID, and version values of an existing artifact to the project's Maven repository, the upload fails. If a build tries to upload an artifact with duplicate groupID, artifactID, and version values, it fails too. To allow the duplicate artifact to be uploaded, you can configure the project's Maven repository to allow the build to upload artifacts with duplicate groupID, artifactID, and version values.

You must be assigned the project **Owner** role to configure the overwrite property.

1. In the navigation bar, click **Project Administration**.
2. Click **Repositories**.
3. In **Maven Repository**, if necessary, expand **Configure Overwrite Property for Release Artifacts**.
4. Select the **Allow** check box.

5

Collaborate with Your Team

After adding team members and setting up your project, you can use merge requests for peer review of code files, wikis to share documentation, and snippets to share common code files.

This table describes the Oracle Developer Cloud Service pages you'd use to collaborate your project with team members.

Use this page ...	To:
Merge Requests	Merge branches and peer review the source code.
Wiki	Create and view wiki pages of the project.
Snippets	Create and share common code files and snippets.

Review Source Code with Merge Requests

The review of the source code helps in avoiding bugs, identifying design issues, and catching design and implementation problems that might affect the application performance. To get the source code reviewed, you must create a merge request.

Merge Requests Concepts and Terms

As the name suggests, a merge request is a request to merge a branch into another. Before merging the branch, you may want your team members to review updates made to the branch and share their feedback. A merge request combines the review and merge processes into one easy collaborative process.

You can also link related issues and builds to the merge request that are automatically updated or triggered when you merge branches.

Here are the terms that this documentation uses to describe the merge request features and components.

Term	Description
Review branch	Branch to be reviewed and merged.
Target branch	Branch that the review branch will merge into.
Reviewer	Project user invited to review the changed files of the review branch.
Requester	Project user who created the merge request.
Subscriber	Project user who isn't a reviewer, but is watching the merge request.
Default reviewer	Project user who's automatically added as a reviewer if a branch is selected as the review branch. Only a project owner can create default reviewers of a branch.

Term	Description
Approved	Reviewer's feedback with no objection to the changes made to the source code in the review branch.
Rejected	Reviewer's feedback with objections to changes made to the source code in the review branch and a recommendation not to merge the review branch into the target branch.
General comment	A comment in the Conversations tab of the merge request.
Inline comment	A comment added to a line of a file under review.
Pending (or unpublished) comment	An inline comment that you didn't publish when you added it.

To understand the workflow of a merge request, let's consider you're a software developer assigned a new feature to implement. These steps summarize the action you'd perform to set up a merge request and merge branches.

1. Create a branch from a stable branch (say `master`) of the source code Git repository. You'd add or update the files of the new branch to implement the new feature.

   You can do this in the cloned Git repository on your computer or on the DevCS **Git** page.
2. On your computer, pull the latest content from the project's Git repository, checkout the new branch, update the required files, and commit and push the checked out branch to the project's Git repository.
3. If required, create a build job to generate artifacts from the new branch to verify the stability of the application.
4. Create a merge request with the new branch as the review branch and the stable branch as the target branch.
5. Add your manager and other team members as reviewers.
6. To resolve the feature related issues when you close the merge request, link the issues to the merge request.
7. Depending on the review feedback, you may need to update some files and check the stability of the branch. To trigger a build of the job automatically when you update the files of the review branch, link the job to the merge request.
8. Again, based on the feedback and build status of the linked jobs, you may want to merge the branch with the stable branch or abandon it. If you merge the branches, the linked issues are automatically resolved.

If you're invited to a merge request, you can add comments to the updated files, and share your feedback whether you've any objection to merge branches.

1. Open the merge request.
2. Check the commits made to the review branch and compare the changed files.
3. Add general or inline comments, if necessary.
4. Submit your feedback as **Approved** if you find the code updates acceptable, or **Rejected** if you have objections.

If you're a project member but aren't invited to a merge request, you can add comments but you can't share your feedback.

It isn't necessary to add reviewers to a merge request. If you're sure that the changes made to the review branch don't require a review, you can merge both branches without a review. If you're comfortable using Git, you can merge branches from a Git client or from an IDE without creating a merge request.

Merge Request States

A merge request can be in one of various states such as Open, Merged, or Closed.

State	Description
Open	Code review is in progress. A merge request's status remains **Open** until the branches are merged or the request is closed.
Merged	Code review is complete and the review branch has been merged with the target branch. The review is closed for inline comments, but can accept general comments.
Closed	Code review is closed without merging the review branch with the target branch. The review is closed for inline comments, but can accept general comments.

Create and Manage Merge Requests

After you create a merge request, you add reviewers and link related issues and jobs to it.

Create a Merge Request

You can create a merge request from the Merge Request page. You can't create a merge request if the branch that you want to be reviewed has any merge restrictions set or is already under review in another merge request.

You must be a project member to create a merge request.

1. In the navigation bar, click **Merge Requests** .
2. Click **+ Create Merge Request**.
3. On the Branch page of the New Merge Request wizard, in **Repository**, specify the Git repository.
4. In **Target Branch**, select the branch that the review branch would merge into.
5. In **Review Branch**, select or enter the name of the branch to be reviewed. If the branch doesn't exist, it's created.

   If the review branch is already under review in another merge request, the branch name won't appear in the **Review Branch** list.
6. Click **Next**.

7. On the Details page of the New Merge Request wizard, in **Linked Issues**, add issues related to the merge request.

   When merging branches or closing the merge request, you can choose to mark the linked issues as resolved.

8. In **Linked Builds**, add jobs related to the merge request.

   Builds of the linked jobs run automatically when the review branch is updated.

9. In **Tags**, add project tags to associate them with the merge request.

   You can use these tags to search merge requests.

10. In **Summary**, enter a summary (or title) of the merge request. If not specified, the default summary `Merge Request for branch <review_branch_name>` is set.

11. In **Reviewers**, select team members who'll review the updates.

    Names of default reviewers are added automatically, however, you may choose to remove them.

    To add all reviewers of the last merge request you created, select **Reviewers**  .

12. Click **Next**.

13. On the Description page of the New Merge Request wizard, enter a description, and click **+ Create**.

    You can use the project's wiki markup to format the description.

After the merge request is created, all reviewers are assigned the **Reviewer** role and you're assigned the **Requester** role. Email notifications are also sent to reviewers informing them that they are added as reviewers.

Add or Remove Reviewers

You can add reviewers when you create a merge request or while the review is open. You must be the requester or a reviewer to add or remove reviewers.

1. In the navigation bar, click **Merge Requests**  .
2. Click the merge request summary to open it.

You can manage the reviewers from the Review Status section available on the right side of the page.

Action	How To
Add a reviewer	1. In the **Review Status** section, click **Click to add a reviewer**. 2. In the Add Reviewer list, enter the project member name or select the member from the list. 3. Click **Save**  .

Action	How To
Add yourself as a reviewer	If you're a project member but not a reviewer, you can submit a request to add yourself as a reviewer to a merge request. Above the **Review Status** section, click **Add me**. If you're a project owner, you are added immediately to the merge request. If you're a project member, then enter a justification in the Request to be added as a reviewer dialog box, and click **OK**.
Approve a reviewer request	If you're the requester or a reviewer, you can approve requests of project users to join the merge request as reviewers. In the Conversation tab, click **Add User** in the **requested to be a reviewer** request.
Remove a reviewer	In the **Review Status** section, click **Remove** next to the reviewer you want to remove.

Link an Issue to a Merge Request

Linking issues to a merge request enables you to resolve them automatically when you merge or close a merge request.

1. Open the merge request.
2. Click the **Linked Issues** tab.

   The tab displays issues linked to the merge request.
3. To link an issue to the merge request, enter the issue summary text or the issue ID in the **Search and Link Issues** search box, select the issue from the drop-down list, and click **Save**.

If an external Jira server is linked with a project, you can also link Jira issues with the merge request. Select the Jira issues from the window that opens when you click the **Search and Link Issues** search box. Contact the project Owner to know more about the linked external Jira server.

Link a Build Job to a Merge Request

Linking build jobs to a merge request enables you to monitor them from the merge request and trigger them when a commit is pushed to the review branch. Depending on the build's status, reviewers can determine whether the merge request is ready to be merged with the target branch.

1. Configure the job to accept merge request parameters.

   See Use Build Parameters.
2. Open the merge request.
3. Click the **Linked Builds** tab.

   The tab displays linked jobs, if any.
4. In **Search and Link Build Jobs**, enter the job name and select it from the list.
5. Click **Save**.

After a job is linked to a merge request, a build automatically runs when the review branch is updated with a commit.

When a build of a linked job runs, a comment is automatically added to the **Conversation** tab. If the build succeeds, it auto-approves the merge request and adds itself to the **Approve** section of the Review Status list. If the build fails, it auto-rejects the merge request and adds itself to the **Reject** section of the Review Status list.

Watch a Merge Request

You can set up a watch on a merge request and get email notifications when a reviewer adds a comment, a user updates files of the review branch, or a reviewer shares a feedback.

Action	How To
Merge requests where you're a reviewer	By default, you get email notifications of merge requests where you're a reviewer. If you aren't getting the email notifications, select the **Merge Request updates and comments** check box in your user preferences page. 1. In the branding bar, click the user avatar, and select **Preferences**. 2. Click the **Notifications** tab. 3. Select the **Merge Request updates and comments** check box, if not selected. 4. To the left of the **User Preferences** title, click **Close** to return to the last opened page.
Merge requests where you're not a reviewer	1. Open the merge request. 2. Click **CC me**. To stop watching, remove your name from the **Watchers** list.

Merge Request Email Notifications

As the reviewer, the requester, or the subscriber (watcher) you receive email notifications when the merge request is created or updated. A notification of the event also appears in the Recent Activity feed on the **Project Home** page.

Some events that send notifications are:

- Merge request is created
- Additional source code changes are committed to the review branch and pushed to the upstream
- A general comment is added
- An inline comment is published
- Reviewers are added or removed
- Merge request is approved or rejected
- Merge request is closed or merged

Batch emails are sent when:

- A user submits multiple inline comments

- A user submits several private inline comments and publishes them later
- A user submits several general comments in a short duration
- Multiple users carry on multiple conversations at the same time in different inline comments
- Multiple users carry on multiple conversations in general comments

Batch emails are also sent for review events that occurred before the inactivity period, which is usually five minutes after users stop entering comments. Review activities, other than comments related activities, don't send email notifications in the inactivity period. A batch email is sent after the inactivity period listing all review activities that happened prior to the period of inactivity expires.

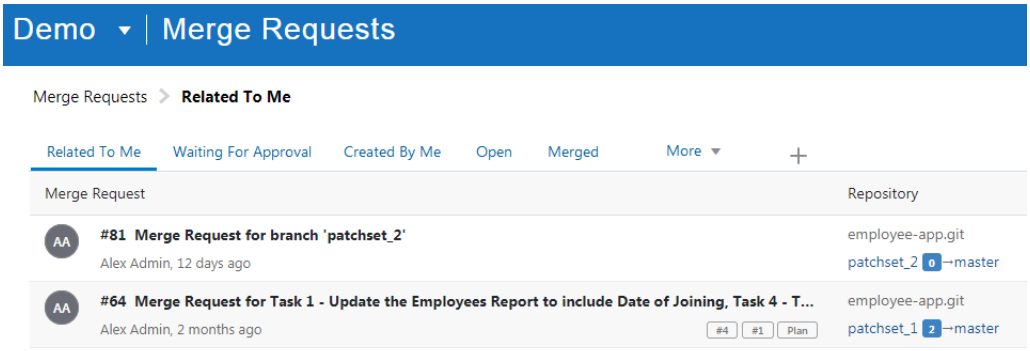
Review a Merge Request

To review a merge request, on the Merge Request page, click its summary. On the Review page, you can view the commits of the review branch, review changed files, add inline and general comments, and submit your feedback.

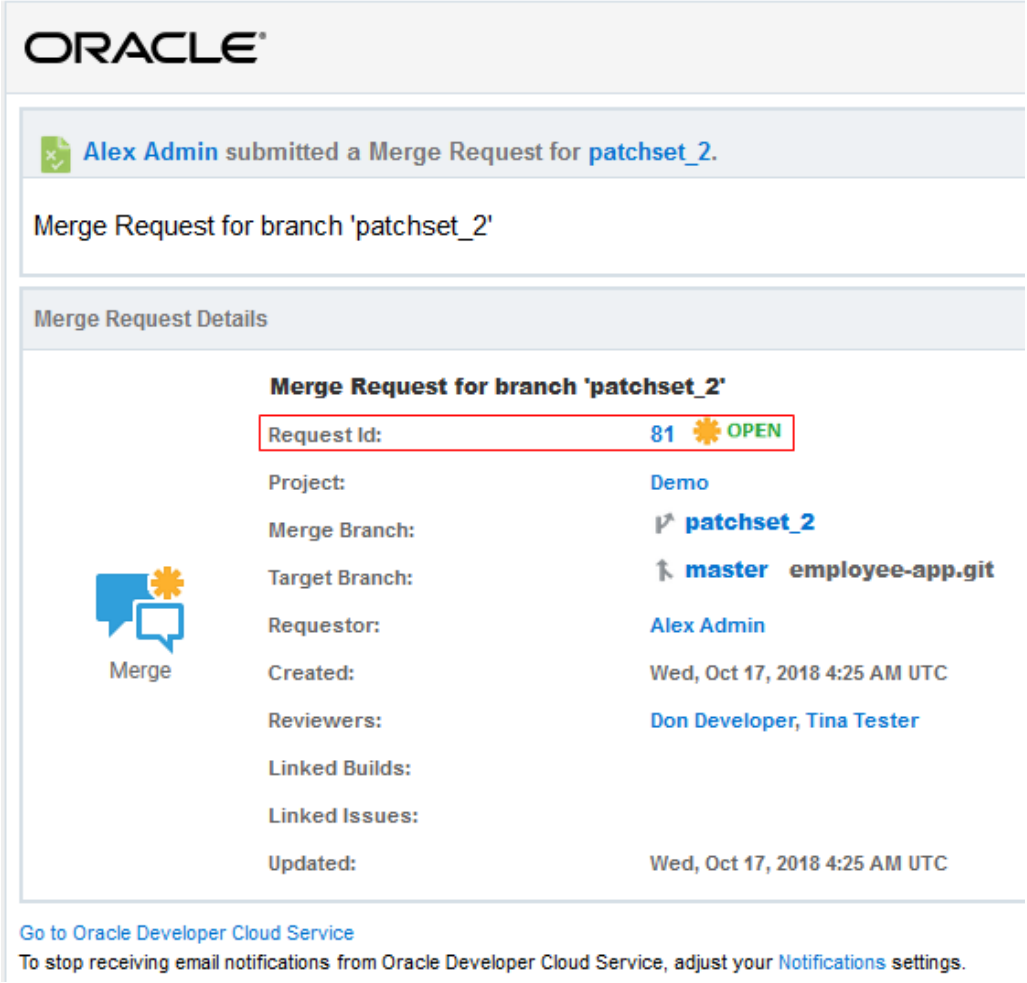
Open a Merge Request

To open a merge request, on the Merge Requests page, click its name.

Use the filter tabs to search for the merge request. By default **Related To Me**, **Waiting for Approval**, **Created By Me**, **Open**, and **Merged** filters are available. More filters are available in the **More** drop-down list.



If you're invited to a merge request, you can also click the request ID from the email notification.



If you still can't find the merge request through the available filters, use the search box at the top of the page or click **New Search** + to run an advanced search.

You can also save the advanced search for future use. In **Search Name**, enter a name and click **Save**. The saved searches are listed in the **More** drop-down list.

View Commits and Changed Files

You can view commits and changed files from the Commits and the Changed Files tabs.

The Commits tab shows all commits made to the review branch. This table lists the common actions you can perform from the Commits tab.

Action	How To
Compare the files of one commit with another	Click the button with the first seven characters of the commit's SHA-1 checksum hash. By default, the page compares the commit with the previous commit.

Action	How To
View all files of the repository when the commit was pushed to the branch	Click **Code**.
View files that were updated in the commit	Click **Show Details**. To compare a file with its parent commit, Click the file name to compare the file changes with its previous commit.

The Changed Files tab shows the files in the compare mode. This table lists the common actions you can perform from the Changed Files tab.

Action	How To
Open a tree view of changed files	Click **Changed Files Tree** .
View the compare options	Click **Diff Preferences** .
Add a comment to a code line or reply to one	Mouse-over the line number of the file and click **Add Comment** .

Add a General Comment

In the Conversation tab, you can view the ongoing conversation and add comments. The comment could be a generic comment, a question you want to ask reviewers, or a comment about an event such as a commit.

1. Open the merge request.
2. In the **Write** tab of the **Conversation** tab, enter your comment.

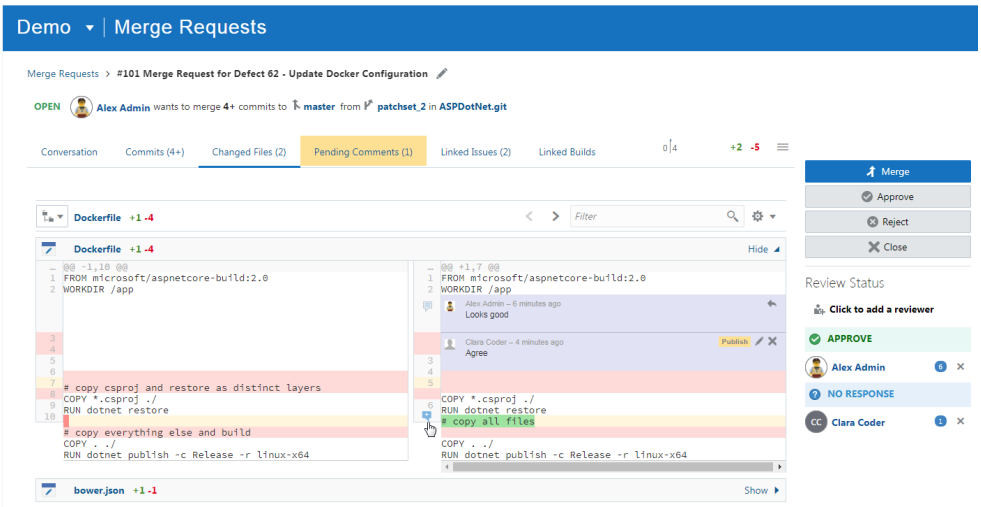
   You can use the project's wiki markup to format the comment. Click the **Preview** tab to preview the format.
3. Click **Submit**.

The comment adds in the **Conversation** tab along with icons to reply, edit, and delete your comment. Note that you can't edit or delete comments entered by other users.

Add an Inline Comment to a File

When a code review is in progress, you can add inline comments directly to a file's code lines. You can't add an inline comment after a merge request has been merged or closed.

1. Open the merge request.
2. Click the **Changed Files** tab.
3. Mouse-over the line number of the file and click **Add Comment** .



4. Add your comment in the comment box.
   - Use the project's wiki markup language to format the comment.
   - Click **Comment** to publish it and make it visible to all reviewers.

     You can't edit or delete a published comment.
   - Click **Save** to save the comment and publish it later. The comment isn't published and isn't visible to reviewers.
   - Click **Cancel** to cancel the comment.

To view your pending or unpublished comments, click the **Pending Comments** tab.

To reply to a published comment, click **Reply**, enter your comment, and click **Comment**. Comment replies to a published comment are published immediately and can't be edited or deleted.

Manage Unpublished Comments

The Pending Comments tab displays all pending comments with the code where these comments were added. The comments appear inline in the code.

- To edit a comment, click **Edit**.
- To publish a pending comment, click **Publish** to the right side of the comment header.
- To publish all pending comments, click **Publish All**.
- To discard all pending comments, click **Discard All**.
- To delete a comment, click **Delete**.

Approve or Reject a Merge Request

As a reviewer, after you review the source code, you can add a special comment that indicates whether you approve the code changes or reject them. Approving a merge request implies that you don't have any objections to changes made to the source

code. Similarly, rejecting a merge request implies that you've an objection and don't recommend merging branches.

Note that if you created the merge request, but didn't add yourself as a reviewer, you can't approve or reject the merge request. However, you can still close it or merge it with the target branch.

1. Open the merge request.
2. Click the **Approve** or **Reject** button at the right side of the page.
3. In the dialog box that appears, add your comment, and click **OK**.

   Use the project's wiki markup to format the comment.

You can see your feedback (approval or rejection) in the **Reviewers** list.

Merge Branches and Close the Merge Request

After addressing reviewers' comments, you can decide whether to merge the branches or cancel the request.

Before doing that, go to the Review Status section and check the review status for the reviewers and the status for linked build jobs. Depending on the number of Approves, Rejects, and No Response, you can decide whether you want to merge the review branch, wait for more approvals, or cancel the request.

Merge Branches

There are various ways to merge the review branch into the target branch. You can merge commits, squash and merge, rebase and merge, or merge the branches manually. To merge branches, you must be assigned either the reviewer role or the requester role of the merge request.

It isn't mandatory to get approval from all reviewers before you merge the review branch. Note that you can't merge the review branch if the target branch is locked and must contact the project owner to unlock the target branch.

1. Open the merge request.
2. On the right side of the page, click **Merge**.
3. In the Merge dialog box, click **Merge Options**, and select the merge type.

Use this merge type ...	To:
Create a merge commit	Merge all commits of the review branch to the target branch. The merge commits continue to show two parents.
Squash and Merge	Add the commit history of the review branch to the target branch as a single commit.
Rebase and Merge	Reapply the commits of the review branch and add them to the top of the target branch.
Manual Merge	Follow the on-screen commands to merge the branches using the Git CLI.

At the top of the dialog box, select the **Remember My Choice** check box to pre-select the current option next time you open the Merge dialog box.

4. If necessary, update the **Merge Summary** and **Merge Description**.

   The fields aren't available if you select **Rebase and Merge** or **Manual Merge**.

5. To delete the review branch after the commits are merged with the target branch, select the **Delete Branch**.

6. If there are any linked issues, deselect the check boxes of the issues that you don't want to mark as resolved after the commits are merged with the target branch. By default, check boxes of all linked issues are selected.

7. Submit the dialog box.

After the review branch has been merged, the merge request is automatically closed and no other action is allowed.
If you didn't select the **Delete branch** check box when you merged the review branch, note that the review branch wasn't removed from the Git repository. You can continue to make commits to the branch and create another merge request to review the new source code.

Resolve a Merge Conflict

Git can automatically resolve code conflicts when the review branch is merged with the target branch. In some cases, however, the conflicts must be resolved manually.

On the Merge Request page, if the **Merge** button is replaced by the **Show Merge Conflicts** button, it indicates a merge conflict.

Git automatically resolves conflicts if different files of the target and review branches are updated before both branches are merged. Merge conflicts are reported when the same lines of the same files are updated in the review branch and the target branch before both branches are merged. In such cases, you have to manually review each conflicting file in the review branch with the code of the same file in the target branch and resolve the conflicting code lines. To find and resolve the conflicting files, run the Git commands that the merge request page shows.

1. Open the merge request.

2. Click **Show Merge Conflicts**.

   A Merge Conflicts dialog box opens with the commands you need to run to resolve the conflict. It also lists the conflicting files.

3. On your computer, open the Git CLI.

4. If you've already cloned the project Git repository, navigate to its directory.

   If you haven't cloned the Git repository, clone it.

5. Run the commands shown on the Merge Conflicts dialog box.

   The commands help you resolve the conflict and mark the conflicting code lines in files.

6. Open each file with conflicts in a text editor or an IDE.

   The conflict content are marked with <<<<<<<, =======, and >>>>>>>. The lines between <<<<<<< and ======= show the code from the target branch. The lines between ======= and >>>>>>> show the code from the review branch.

7. Review the content and update it. Remember to remove the <<<<<<<, =======, and >>>>>>> from each conflicting file before saving it.

8. Save all files and commit them.

Run the `git status` command to view the status of conflicting files.

9. Push the commit to the target branch.

Conflicting files of the review branch are now merged with the target branch. No additional action is required on the Merge Request page.
If you want to delete the review branch, open or refresh the merge request page, and click **Delete Branch**.

Close a Merge Request

You must close a merge request after the review branch has been merged. To close a merge request, it isn't necessary to merge the review branch to the target branch. You can close a merge request if it was created by mistake or if you don't want to merge the review branch to the target branch.

Make sure that you perform any needed merge action before you close the request. Once a merge request is closed, you can't merge the review branch, add comments, or review the source code.

1. Open the merge request.
2. Click **Close**.
3. Complete the elements of the Close Merge Request dialog box.

   To change the review status to **Merged** and close the review, select the **Close as Merged** check box. You may choose the **Close as Merged** option if the review branch was merged through some other means (such as the Git CLI, IDE, or though the `git cherry-pick` command).

   If you don't select the **Close as Merged** check box, the Merge Request is closed without changing the review status to Merged. You may want to do this if the merge request was created on the wrong branch or created by mistake.
4. Click **OK**.

Merge Request and Branch Administration

On a Git repository branch, you can set some restrictions and assign some project users as default reviewers of the branch

For a branch, you can set rename, delete, push, and merge restrictions; or lock a branch if you don't want anyone to push commits to it or merge another branch with it. When a merge request is created with the branch as the target branch, the default reviewers of the branch are automatically added to the Reviewers list.

You must be assigned the project **Owner** role to assign default reviewers to a branch, or set push and merge restrictions on it. For a **Private** branch, a branch owner can also change its restrictions.

Set Review and Merge Restrictions on a Repository Branch

You can configure a branch to allow another branch to merge into it only through a merge request after the merge request has specified number of approvals.

The number of approvals ensures that specified reviewers of the merge request have reviewed the changes of the review branch. You can't merge a branch outside DevCS, such as using a Git client, without meeting the number of approvals requirement of the

merge request. You can set other review restrictions on a branch, such as whether the last build of the branch must be successful to merge it.

You must be assigned the project **Owner** role to set review restrictions on a branch.

1. In the navigation bar, click **Project Administration** .
2. Click **Branches**.
3. In **Repository** and **Branches**, select the Git repository and the branch.
4. Select the **Requires Review** option.

When you select the **Requires Review** option for branch, you can merge a branch after the branch's approvals requirement is met.

This table describes the other review restrictions you can set from the Branches page.

Action	How To
Assign default reviewers to a branch	A default reviewer is a project member who is automatically added as a reviewer when a merge request is created on a branch. In **Reviewers**, enter and select the users.
Set minimum number of approvals before a branch is merged to the selected branch	From the **Approvals** drop-down list, select the minimum number of reviewers who must approve the review branch of a merge request where the selected branch is the target branch.
Allow a review branch to be merged to the selected branch only if the last build of the linked job in Merge Request is successful	Select the **Require successful build** check box.
If a change is pushed to a branch after some reviewers have approved the merge request, merge only when they reapprove the merge request	Select the **Reapproval needed when branch is updated** check box.
Ensure changes pushed to the target branch match the contents of the review branch	Select the **Changes pushed to target branch must match review content** check box
Specify users who can bypass the branch restrictions and merge the review branch of a merge request outside DevCS or without required approvals	In **Merge Request Exempt Users**, specify the users. This is useful if you want to allow some users to merge the review branch irrespective of review conditions being met.

Collaborate on Documentation Using Wikis

You can use Wikis to collaborate on your projects' documentation.

Oracle Developer Cloud Service (DevCS) supports these Wiki markup languages:

- Confluence

  See `http://www.atlassian.com/software/confluence/`.

- Textile

  See `http://textilewiki.com`.

- Markdown

  See `http://daringfireball.net/projects/markdown/`.

Project users can use the project's Wiki markup language to format content in wiki pages, and in issue and merge request comments.

Create and Manage Wiki Pages

From the **Wiki** page, you can create and open wiki pages, add child wiki pages, add attachments, and delete or restore deleted wiki pages. From the Wiki Drafts page, you can open and edit saved drafts, publish drafts, and delete any drafts that are no longer needed.

In the navigation bar, click **Wiki** to create and manage wiki pages. Any project user can create a wiki page. For a page that you didn't create, depending on the edit and delete access set by the creator of the wiki page, you may or may not be able to edit or delete it. If you're a project owner, you can always do that.

Action	How To
Create a wiki page	1. On the **Wiki** page, click **+ Create Page**. 2. In the **Page Text** tab, enter the content. Use the project's wiki markup language to format the contents. To open the markup language's cheat sheet, click the reference link above the text area. 3. Click **Save** to publish the new page. Click **Close** and you'll be allowed to retain a draft of the page you were working on or delete the draft and discard your work.
Open a wiki page	To open a wiki page, click the wiki page title. If a wiki page has child wiki pages, click next to the wiki page title and select the child wiki page name to open it.
Edit a wiki page	To edit a page, click **Edit**. If a saved draft for the page exists, select **Resume editing** to display the draft with the last saved changes or select **New Edit** to start editing the published page. Use the wiki's markup language to format the contents. To open the markup language's cheat sheet, click the reference link above the text area. If the project is using the Markdown markup language: • Use # followed by the issue ID to add a reference to the issue. Example: #12 creates a link the issue ID 12. • Use ! followed by the merge request ID to add a reference to the merge request. Example: !34 creates a link to the merge request ID 34. • See `http://www.emoji-cheat-sheet.com/` for the complete list of supported emoticons and text patterns.

Action	How To
Edit a draft of a wiki page	1. On the **Wiki** page, click **My drafts**. 2. Locate the draft of the wiki page you want to edit and click **Edit** under **Actions**.
Create a child wiki page	1. Open the parent wiki page. 2. Click **New Child**. You can also create a child wiki page without opening or creating its parent wiki. On the **Wiki** page, click **New Page**. In **Page Title**, enter the path of the child wiki page. For example, to create a `HelloWorld` child page of the `Welcome` page, enter `Welcome/HelloWorld` in the title. If the `Welcome` page doesn't exist, DevCS creates both `Welcome` and `HelloWorld` pages. Open the empty `Welcome` wiki page and click **Create** to create the wiki page and add contents to it.
Add an attachment	You can attach files of any type to a wiki page, including images, videos, docs, spreadsheets, and archived files. 1. Open the wiki page or the draft in the edit mode. 2. Click the **Attachments** tab. 3. In the **Files to Attach or Update** section, click **Select**. Browse and select files to attach. You can also drag and drop files to the drop area. 4. Click **Attach**. 5. Click **Save**.
Publish a draft of a wiki page	1. On the Wiki Drafts page, locate the draft of the wiki page you want to publish. 2. Click **Publish** under **Actions**. A draft of a new page can only be published if it contains a path. A draft of an existing page can only be published if the draft contains changes from the underlying page.
Delete a wiki page	1. Open the wiki page 2. Click **Delete**. 3. In the Delete Wiki Page dialog box, select the **I understand that my wiki page will be permanently deleted** check box and click **Delete**. The wiki page along with its comments is deleted. An activity with a link to restore the wiki page (if necessary) is added to the recent activities feed of the **Project Home** page. If you delete a parent wiki page, its child wiki pages aren't deleted. The parent wiki page name continues to appear in the breadcrumb path. To delete a draft of a wiki page, from the Wiki Drafts page, click **Delete** under **Actions**. In the Delete Wiki Page dialog box, select the **I understand that my wiki draft will be permanently deleted** check box, and click **Delete draft**.

Action	How To
Restore a wiki page	1. Click the deleted wiki name in the recent activity feed or any other wiki page where it was referenced. 2. Click **Restore**. The wiki page along with comments and attachments restores to its original path. To restore a parent wiki page, open it and click **Restore**.

Add Comments

You can add general or block comments to a wiki page. You must be a project member to add comments. In a shared project, any organization user can add comments.

Action	How To
Add a generic comment	1. Open the wiki page. 2. Scroll down to the Comments section. In the **Write** tab, enter your comment, and click **Comment**. You can use the project's wiki markup language to format your comment. The comment appears in a conversation box along with icons to **Reply**  , **Edit**  , and **Delete**  . You can't edit comments entered by other users. You can't delete a parent comment if there are any child comments, or delete comments entered by other users.
Add a block comment	A block comment is a comment that you add to a content block, such as a paragraph or a table. 1. Open the wiki page. 2. Move the mouse pointer to the right edge of the block, and click **Add Comment**  . If you see a number on the right edge of the block, it indicates the number of existing comments of the block. Click the number to view the comments and then click **Leave a comment**. 3. In the popup, add your comment, and click **Post**. When you add a block comment, a watch is set on the wiki page and you automatically get email notifications of future comments and updates. To hide the comment, click the comment number or **Hide**  . To edit or delete the comment, click the number and then click **Edit** or **Delete**. If you edit, format, or move a content block, its comments move automatically. If you remove a content block, its comments are deleted too and can't be restored. If you merge a block with another, the comments of the source block will be hidden, but the comments of the target block remain visible. When you the split the source block from the target block and move it back its original position, the hidden comments reappear.

Watch a Wiki Page

You can set a watch on a wiki page and receive email notifications whenever someone updates the page or adds a comment.

To receive email notifications, in the User Preferences page, click the **Notifications** tab, and then select **Wiki page updates and comments**.

Action	How To
Watch a wiki page	1. Open the wiki page. 2. Click **Watch Wiki**. 3. In the menu, select the **Watch Page** check box. The button's label changes to **Watching**. To stop watching the page, click **Watching**, and deselect the **Watch Page** check box.
Watch all pages of the project	On the Wiki page, click **Watch Wiki**. The button label changes to **Watching**. To stop watching all pages, click **Watching**.

View History and Compare Versions of a Wiki Page

Each time you save a wiki page, Oracle Developer Cloud Service creates a version of it. You can view the contents of the wiki page of a previous version, compare any two versions, restore the wiki page to a particular version, or delete a particular version.

Action	How To
View the history of a wiki page and its versions	1. Open the wiki page. 2. Click the **Page History** tab.
View the contents of an old version	In the **Page History** tab, in the version number row, click **View**.
Compare two versions	In the **Page History** tab, select the check boxes of versions, and click **Compare selected versions**.
Restore the wiki to a version	In the **Page History** tab, in the version number row, click **Restore this version**. In the Revert Wiki Page dialog box, select the **I understand that my wiki page will be updated to this revisions' content** check box and click **Yes**. Restoring a page creates a new version of the page with the contents from the version you want to restore. You can't restore a wiki page if you don't have its edit access.

Wiki Administration

Wiki page creators, project owners, or Organization Administrators can make some administrative updates for wikis. These updates include changing edit and delete access rights for a wiki page, changing a project's wiki markup language, or changing an organization's default wiki markup language.

Configure Edit and Delete Rights for Wiki Pages

Edit access enables users to edit the wiki content, create child pages, or restore a deleted version. Delete access enables users to delete a wiki page, or delete a version from the Page History tab.

You can assign the access to a project role, not to a particular user. Project owners can always edit or delete a wiki page and assign access to other roles. If you created a wiki page, you can also assign its edit and delete access to other roles.

If you're a project member, note that for a wiki page that you create and allow other users to edit its access rights, you could lose your edit or delete rights for the page if another user changes the edit or delete rights to **Owners** only.

1. Open the wiki page.
2. Click **Edit**.
3. Click the **Access Rights** tab.
4. From the **Edit Access** list or the **Delete Access** list, select the project role. By default, it's set to **Members and Owners**.

Select this option ...	To assign the access to ...
All	All organization users. Use this option to enable all users to edit or delete the page.
Members and Owners	All project users. In a shared project, organization users can view the wiki page, but can't edit or delete it.
Owners	Project owners only. Project members can view the page, but can't edit or delete it. In a shared project, organization users can also view the wiki page, but can't edit or delete it.

5. Click **Save**.

Change a Project's Wiki Markup Language

The wiki markup language of a project is defined when the project is created. You can change the markup language for the new pages and comments, but note that the existing wiki pages and comments continue to use their original wiki markup language.

You must be assigned the project **Owner** role to change a project's wiki markup language.

1. In the navigation bar, click **Project Administration**.
2. Click **Properties**.
3. In the Properties page, from **Markup Language**, select the wiki markup language.

When you're finished, use the project navigation bar to switch to another page.

Set the Organization's Default Wiki Markup Language

When you create a new project, the project creation wizard displays the default wiki markup language of the organization. If you don't change the wiki markup language, the default markup language is used as the project's markup language. You can

change the default markup language for new projects, but note that existing projects continue to use their original markup language.

You must be the **Organization Administrator** to set or change the organization's default wiki markup language.

1. In the navigation bar, click **Organization** .
2. Click the **Properties** tab.
3. In **Markup Language**, select the default wiki markup language.

Share and Use Code Snippets

A Snippet hosts reusable code in files that you can use in the project and share with other project members. The file could include a small block of reusable source code or text that can be incorporated into larger modules.

Content in snippet files may not be code, but it should be useful. This content could be notes that you want to share with project members, or something you want to keep private, such as a reminder to yourself. If a snippet is shared, project members can copy or download the snippet files and then use them in their own applications.

A snippet can contain several files. When you create a snippet, you can add one file only, but you can add additional files after creating the snippet.

Create and Manage Snippets

You can create a snippet from the Snippets page or from a text selection in a code editor. You can only add one file when you create the snippet, but you can add more files later.

This table describes how you can create and manage a snippet.

Action	How To
Create a snippet	1. In the navigation bar, click **Snippets** . 2. Click **+ Create Snippet**. 3. On the Snippet Details page of the New Snippet wizard, enter the snippet name and description. 4. In **Visibility**, select **Private** if you don't want to share the snippet's files and keep them for personal use. Select **Shared** to share the snippet's files with project members. 5. To edit the snippet and add more files immediately after creating the snippet, select the **Edit snippet when finished** check box. 6. Click **Next**. 7. In the Snippet Content page of the New Snippet wizard, enter content for the default file of the snippet. If you don't enter content, an empty file `snippet1.txt` is added to the snippet. 8. Click **Finish**.

Action	How To
Create a snippet from a selection	You can create a snippet from a file that's open in the code editor. 1. In the open file, select the text. 2. Right-click and select **New Snippet from Selection**. 3. On the Snippet Details page of the New Snippet wizard, enter the snippet name and description. 4. In **Visibility**, select **Private** if you don't want to share it and keep for personal use. Select **Shared** to share the snippet with project members. 5. To edit the snippet and add more files immediately after creating the snippet, select the **Edit snippet when finished** check box. 6. In the Snippet Content page of the New Snippet wizard, enter content for the default file of the snippet. If you don't enter content, an empty file is added to the snippet. 7. Click **Finish**.
Share or stop sharing a snippet	After creating a snippet you may want to share it with your team members, or stop sharing it if it is already shared. You can set the share status of a snippet that you own. To share a snippet, in the **My Snippets** view of the Snippets page, click **Share**  . The icon changes to **Shared**  . To stop sharing a snippet, click **Shared**  . The icon changes to **Private**  .
Edit a snippet's title	You can edit the title of a snippet that you own. On the Snippets page, click the snippet name, and then click **Edit**  .
Delete a snippet	You can delete a snippet that you own. On the Snippets page, click the snippet name, and then click **Delete**. In the Delete Snippet dialog box, click **Yes** to confirm.

Add and Manage Files of a Snippet

You must be the creator of the snippet to add or manage its files.

Action	How To
Add a file	1. Open the snippet. 2. Scroll down and after the snippet's files, click **Add File**. 3. In the header, enter the file name with extension. In the editor, enter the file's content. The editor supports various code editing features such as autocomplete, indentation, syntax highlighting, code folding, and bracket matching. 4. After adding the content, scroll up and at the top of the page, click **Save**. To save the updates and stay on the Edit Snippet page, select **Save** To save the updates and exit, select **Save and Exit**.

Action	How To
Edit a file	1. Open the snippet. 2. If necessary, rename the file and configure its properties. 3. In the editor, update the file contents. 4. At the top of the page, click **Save**. To save the updates and stay on the Edit Snippet page, select **Save** To save the updates and exit, select **Save and Exit**.
Delete a file	1. Open the snippet. 2. For the file that you want to delete, on the right side of the file header, click **Remove File** . 3. In the Delete Snippet File dialog box, click **Yes** to confirm. 4. At the top of the page, click **Save**. To save the updates and stay on the Edit Snippet page, select **Save** To save the updates and exit, select **Save and Exit**.

Copy Contents of a Snippet File

You can copy contents of a snippet file manually from the Snippets page or insert it from the context menu in the code editor component.

The code editor is available in various editing pages and input fields such as the Edit Wiki page, edit mode of the readme file, snippet file editor, merge request comment box, and the shell command box of the Configure Build page.

Action	How To
Copy from the Snippets page	1. Open the snippet. 2. For the file whose contents you want to copy, click **Copy** . In some browsers, you must press Ctrl+C to copy the content to the clipboard after clicking **Copy** . 3. Paste the contents into the text field.
Insert from the context menu	1. In the code editor, right-click, and select **Insert from Snippet**. 2. In the Select Snippet page of the Insert From Snippet wizard, select the snippet, and click **Next**. 3. In the Select File page, click the file name whose contents you want to insert. 4. Click **Finish**.

Add a Comment to a Snippet

Add a comment to a snippet to share information with other developers.

To add a comment to a snippet:

1. Open the snippet.
2. Scroll down to the **Comments** section.
3. Enter the comment in the comment box.

   Use the project's wiki markup language to format the comment.
4. Click **Add**.

Use Git with Snippets

You can use Git to clone a snippet repository and manage its files. After you clone the snippet's repository, you can view the file history, and update and commit files locally. However, you can only push updates to repositories for snippets that you own.

1. Open the snippet.
2. On the Snippet Details page, at the top, click **Clone**, and then click **Copy** to copy the **HTTP** or the **SSH** URL of the snippet repository.
3. Use Git commands to clone the repository, update files, and push the commits to the project.

Download an Archive of the Snippet

You can download a zip or a tgz file of the snippet to your computer. You may want to do this if you back up the files of the snippet before deleting it.

1. Open the snippet.
2. On the Snippet Details page, at the top, click **Clone**, and the select **Download ZIP** or the **Download TGZ** option.

The downloaded file is an archive file that contains the latest content of all the files in the snippet. To view previous versions of the files, you must clone the snippet repository, and then use Git commands to show the history of the files.

6

Build Applications and Deploy to Oracle Cloud

After uploading the source code files to Git repositories, you'd want to run their builds and generate artifacts, and then deploy those artifacts to Oracle Cloud.

This table describes the Oracle Developer Cloud Service pages you'd use to build and deploy artifacts.

Use this page ...	To:
Builds	Create and configure build jobs and pipelines.
Deployments	Deploy build artifacts to Oracle Java Cloud Service, Oracle Application Container Cloud Service, and Oracle Java Cloud Service - SaaS Extension.

Configure and Run Project Jobs and Builds

Oracle Developer Cloud Service (DevCS) includes continuous integration services to build project source files. You can configure the builds from the **Builds** page.

The **Builds** page, also called the Jobs Overview page, displays information about all build jobs of the project and provides links to configure and manage them.

> **Note:**
>
> Compute-based builds are not available to DevCS users in the Traditional identity domain. The built-in shared build executor is available only, so there are no Build VMs, no Build VM templates, and no need to configure the OCI account. The ability to use compute-based builds is only available for DevCS users on OCI.

What is a Build VM and a Build VM Template?

A Build Virtual Machine (VM) is an OCI Compute or an OCI Compute Classic VM that runs builds of jobs defined in the DevCS projects. A Build VM Template defines the operating system and the software installed on Build VMs.

An Organization Administrator creates Build VM templates and Build VMs. If you're an Organization Administrator, you first create Build VM templates with software that your organization members use. After creating the templates, you allocate some Build VMs to each Build VM template. When your organization's members create jobs, they associate the Build VM template with each job.

To configure a job to use software, such as Node.js or Docker, assign the Build VM template to the job. If you're a project member, you assign the Build VM template to the job from the job's configuration page.

Read these steps to understand what happens when a build runs:

1. The build executor checks the Build VM template of the job and then looks for a VM that's allocated to the template.
   - If a VM is available, the build executor immediately runs the build on it.
   - If all VMs are busy running builds of other jobs using the same Build VM template, the build executor waits until a VM becomes available and then runs the current job's build on it.
2. If a build is running on a VM for the first time or a VM wakes up after its sleep timeout period, the build executor first installs the software defined in the Build VM template. This takes some time.
3. After installing the software, it clones the job’s Git repositories (if configured) to the VM, runs the defined build steps, creates artifacts (if configured), and performs post-build steps (if configured).
4. After the build is complete, the executor copies the artifacts (if generated) to the OCI Object Storage bucket or the OCI Object Storage Classic container, as defined by the Organization Administrator.
5. The Build VM waits for some time for any queued builds. If no builds run in the wait time period, the Build VM uninstalls its software and stops.

What is a Job and a Build?

A Job is a configuration that defines the builds of your application. A Build is a result of a job’s run.

A job defines where to find the source code files, how and when to run builds, the software and the environment required to run builds. When a build runs, it creates packaged archives of the application that you can deploy to a web server. A job can have hundreds of builds. Each build would’ve its artifacts, logs, and reports.

Build Concepts and Terms

Here are some terms that this documentation uses to describe the build features and components.

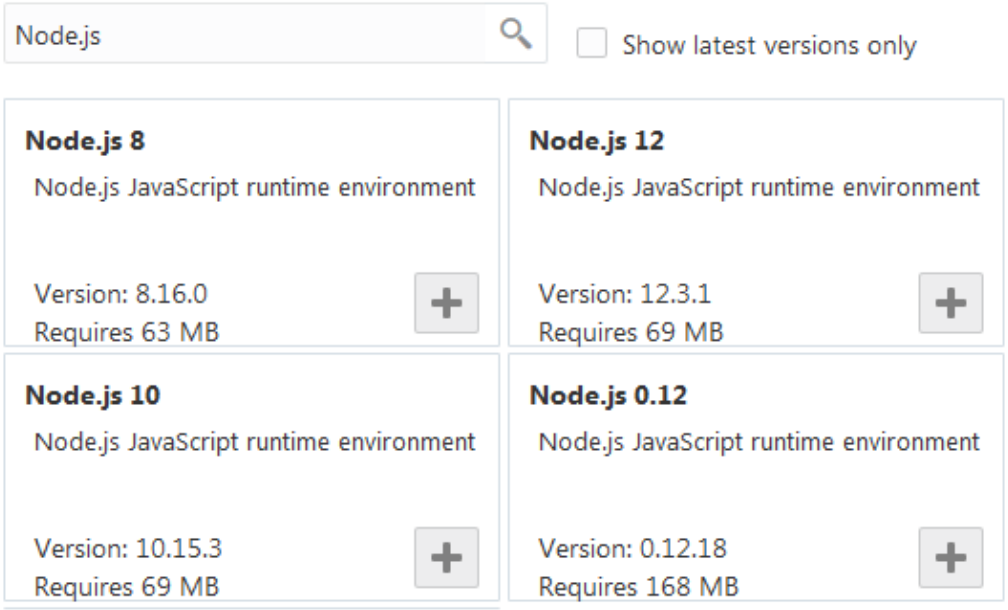
Term	Description
Build System	Software that automates the process of compiling, linking and packaging the source code into an executable form.
Build executor	A basic block that enables a build to run on a Build VM. Each build uses one build executor, or one Build VM.
Workspace	A temporary directory used by the build executor to download source code files and generate artifacts when a build runs.
Build artifact	A file generated by a build. It could be a packaged archive file, such as a .zip or .ear file, that you can deploy to a build server.
Trigger	A condition to run a build.

Software Versions in the Software Catalog

Some software, such as Node.js and Java, are available in multiple versions in the Software Catalog. Such software are called as Software Packages.

The version number of a package can be categorized into two: the major version and the minor version. If a software's version is `1.2.3`, then `1` is its major version and `2.3` is its minor version. In a software's tile, the major version number is displayed in the title of the package. In Configure Software page, the number shown in **Version** is the installed version, which includes both major and minor versions.

Here's an example. In this image, Node.js 0.12, 8, 10, and 12 are shown in the software catalog. In the Node.js 12 tile, `12` is the major version and `3.1` is its minor version. The installed version of the software is `12.3.1`.



When a new minor version of a software package is available in the Software Catalog, all VM templates using that software package are updated automatically. For example, assume that Node.js 10.13 is available in the Software Catalog for the Node.js 10 package. When Node.js 10.15 is made available in the Software Catalog, all VM templates using the Node.js 10 package update automatically to use Node.js 10.15. If there's an incompatibility between the upgraded software and other installed software of the VM template, an error is reported with suggestions about the cause of the error.

When a new major version of a software package is available in the catalog, VM templates using the older versions of the software package aren't updated automatically. The new major version of the software is added to the catalog as a separate package. For example, when Node.js 12 is available in the Software Catalog, all VM templates using Node.js 0.12, Node.js 8, or Node.js 10 aren't updated automatically. To use the new version, you must manually update the VM templates to use the new package.

When a major version of a software is removed from the catalog, all VM templates using that software version are updated automatically to use the next higher version. For example, when Node.js 8 phases out, VM templates using Node.js 8 will be automatically updated to use Node.js 10.

Set Up the Build System

Before your organization's members can run builds, you must create Build VM templates and allocate Build VMs to those templates.

You must be the **Organization Administrator** to create and manage Build VM templates and Build VMs.

> **Note:**
>
> Compute-based builds are not available to DevCS users in the Traditional identity domain. So, there are no Build VMs, no Build VM templates, and no need to configure the OCI account.

Create and Manage Build VM Templates

You can create and manage Build VM templates from the VM Templates page in Organization Administration.

Action	How To
Create a Build VM template	1. In the navigation bar, click **Organization**. 2. Click the **Virtual Machines Templates** tab. 3. Click **+ Create Template**. 4. In the New VM Template dialog box, enter a name and description of the VM template. In **Platform**, select the operating system to run on the VM. 5. Click **Create**.
Configure a VM template's software	1. In the navigation bar, click **Organization**. 2. Click the **Virtual Machines Templates** tab. 3. Select the template and click **Configure Software**. 4. If necessary, in **Filter Software Packages**, enter the search term and click the Search icon. To see the latest version of the software, select the **Show latest versions only** check box. This is helpful if multiple versions of the same software are available in the catalog. 5. In the Software Catalog, click **Add**. If a software is dependent on another software, a message informs you about the dependencies that must be added along to use the software. 6. Click **Done**.

Action	How To
Edit a VM template's name or description	1. In the navigation bar, click **Organization**  . 2. Click the **Virtual Machines Templates** tab. 3. Select the template and click **Edit**. 4. In the Edit VM Template dialog box, update the name or description. 5. Click **Save**.
Delete a VM template	When you delete a VM template, its Build VMs are deleted too. 1. In the navigation bar, click **Organization**  . 2. Click the **Virtual Machines Templates** tab. 3. Select the template and click **Delete**  . 4. In the Delete VM dialog box, click **Yes** to confirm.

Add and Manage Build VMs

When you add a Build VM, you allocate a VM on the linked OCI Compute or OCI Compute Classic to run builds of jobs.

> **Tip:**
>
> To minimize build execution delays, set the number of VMs of a specific Build VM template to the number of jobs that you expect to run in parallel using that template. If the VM quota is available, that number of Build VMs will be added to the **Build Virtual Machines** tab.
>
> You can always return to the **Build Virtual Machines** tab to add or remove VMs, based on your actual usage. Note that the more VMs you have running at a specific time, the higher the cost. To minimize the higher cost, use the Sleep Timeout setting on the Virtual Machines page to automatically shut down inactive VMs.

Each build runs in one build executor, or one VM. You can build up to 99 builds in parallel using the same Build VM template.

Action	How To
Add a Build VM	1. In the navigation bar, click **Organization**  . 2. Click the **Build Virtual Machines** tab. 3. In the **Build VMs** tab, click **+ Create VM**. 4. In the Add Build VM dialog box, in **Quantity**, specify the number of VMs you want to allocate. In **VM Template**, select the Build VM template. 5. In **Region**, specify the VM's region. A region is a localized geographic area, and an availability domain is one or more data centers located within a region. Remember, a Build VM is a VM on OCI Compute or OCI Compute Classic. To learn more about regions, see Regions and Availability Domains. 6. In **Shape**, select the VM's shape. A shape is a template that determines the number of CPUs, amount of memory, and other resources allocated to a newly created instance. To learn more about shapes, see VM Shapes. 7. Click **Add**.
View a Build VM's log	The Build VM's log has entries for all events along with information about when the events occurred, the type of event, and event details. 1. In the navigation bar, click **Organization**  . 2. Click the **Build Virtual Machines** tab. 3. In the **Build VMs** tab, select the VM, click **Actions**  , and select **Display Log**. 4. In the Provisioning Log window, review the log. To download the log file to your computer, click **Download Log**.
Start or stop a VM manually	When a build of a job triggers, its VM starts automatically if it was in the stopped state. It takes some time to start a VM and the user must wait for a VM to start before the job's build runs on it. Similarly, a VM stops automatically if no builds run on it during the sleep timeout period. At times, you may want to manually start a VM before triggering a job's build or stop it to free resources immediately. 1. In the navigation bar, click **Organization**  . 2. Click the **Build Virtual Machines** tab. 3. In the **Build VMs** tab, select the VM, click **Actions**  , and select **Start** or **Stop**.

Action	How To
Delete a Build VM	When you delete a Build VM, it waits for all builds running on it to complete and won't accept new builds while it's waiting for the running builds to complete. After all the builds are complete, it shuts down and is removed. 1. In the navigation bar, click **Organization** . 2. Click the **Build Virtual Machines** tab. 3. In the **Build VMs** tab, select the VM, click **Actions** , and select **Delete**. To delete multiple VMs, select them, click **Update Selected** , and select **Delete Selected VMs**. The deleted VMs are listed in the **Deleted VMs** tab.
Change the sleep timeout of all VMs	The sleep timeout defines the duration (in minutes) a VM would be available in the active state if no builds run on it. By default it is 1500 minutes. 1. In the navigation bar, click **Organization** . 2. Click the **Build Virtual Machines** tab. 3. Click **Sleep Timeout**. 4. In the Sleep Timeout dialog box, change the timeout duration, and click**Save**.

After adding Build VMs, you can configure the jobs to use the VM templates.

Create and Manage Jobs

To run builds and generate artifacts that you can deploy, you must create a job. You can create a job from the Build page.

Action	How To
Create a blank job	1. In the navigation bar, click **Builds** . 2. In the **Jobs** tab, click **+ Create Job**. 3. In the New Job dialog box, in **Name**, enter a unique name. 4. In **Description**, enter the job's description. 5. In **Template**, select the Build VM template. 6. Click **Create**.

Action	How To
Copy an existing job	Sometimes, you may want to copy parameters and configuration of a job to another. You can do that when you create a job. You can't copy the configuration of an existing job to another existing job. After you create the new job, you can change the copied parameters and configuration. 1. In the navigation bar, click **Builds**. 2. In the **Jobs** tab, click **+ Create Job**. 3. In the New Job dialog box, in **Name**, enter a unique name. 4. In **Description**, enter the job's description. 5. Select the **Copy From Existing** check box. 6. In **Copy From**, select the source job. 7. In **Template**, select the Build VM template. 8. Click **Create**.
Create a job to associate it with a merge request and accept build parameters	1. In the navigation bar, click **Builds**. 2. In the **Jobs** tab, click **+ Create Job**. 3. In the New Job dialog box, in **Name**, enter a unique name. 4. In **Description**, enter the job's description. 5. Select the **Use For Merge Request** check box. 6. In **Template**, select the Build VM template. 7. Click **Create**.
Create a job using YAML	In DevCS, you can use a YAML file (a file with `.yml` extension) to create a job and define its configuration. The file is stored in a project's Git repository. See Create and Configure Jobs and Pipelines Using YAML.
Configure a job	The job configuration page opens immediately after you create a job. You can also open it from the **Jobs** tab. Click **Configure**.
Run a build of a job	In the **Jobs** tab, click **Build Now**.
Delete a job	In the **Jobs** tab, click **Delete**.

Configure a Job

You can create, manage, and configure jobs from the Jobs tab of the **Builds** page.

To open a job's configuration page, in the **Jobs** tab of the **Builds** page, click the job's name. In the job's details page, click **Configure**.

You can also open a job's configure page from the **Builds** page. In the **Jobs** tab, for the job you want to configure, click **Configure**.

Configure the Privacy Setting of a Job

In a job, any project member can view its configuration, edit it, or run its build. To restrict access of members, mark the build job as Private and authorize some members who can view the job's configuration, edit it, or run its build.

You must be assigned the project **Owner** role to configure the privacy setting of a job. You can't change the privacy setting of a YAML job.

1. In the navigation bar, click **Project Administration**.
2. Click **Builds**.
3. Click the **Job Protection** tab.
4. From the jobs list, select the job.
5. Select the privacy option.
   - **Open**: Any team member can view the job's configuration, edit it, or run a build. This option is selected by default.
   - **Private**: Only authorized team members can view the job's configuration, edit it, or run a build.
     Note that a private job must be run manually. The build job won't run if a non-authorized user tries to run the build job via SCM/Periodic trigger, or a pipeline.
6. If you selected **Private**, in **Authorized Users**, specify the team members as its authorized users.
7. Click **Save**.

A private job shows a **Lock** icon in the jobs list on the left side of the Job Protection page, in the **Jobs** tab of the **Builds** page, and in the pipelines.
If a non-authorized user tries to view a private job, the message **This job is private. You need permission from your project owner o view this job.** is displayed.

Access Project Git Repositories

You can configure a job to access project's Git repositories and their source code files.

1. Open the job's configuration page.
2. Click **Configure**, if necessary.
3. Click the **Git** tab.
4. From the **Add Git** list, select **Git**.
5. In **Repository**, select the Git repository to track.
   When you created the job, if you selected the **Use for Merge Request** check box, the field is automatically populated with the `${GIT_REPO_URL}` value. Don't change it.
6. In **Branch**, select the branch name of the repository to track. By default, `master` is set.

When you created the job, if you selected the **Use for Merge Request** check box, **Branch** is automatically populated with the `${GIT_REPO_BRANCH}` value. Don't change it unless you don't want to link the job with a merge request.

7. Click **Save**.

You can specify multiple Git repositories. If you do, set **Local Checkout Directory** for all Git repositories.

Trigger a Build Automatically on SCM Commit

You can configure a job to monitor its Git repositories and trigger a build automatically after a commit is pushed to the Git repository.

1. Open the job's configuration page.
2. Click **Configure**  , if necessary.
3. Click the **Git** tab.
4. For the Git repository you want to monitor, select the **Automatically perform build on SCM commit** check box.
5. To specify a list of files and directories to track for changes in the repository, expand **Advanced Git Settings**, and enter the list in **Included Region**.

   You can use regular expressions (regex) or glob to specify fileset. Each entry must be separated by a new line. An empty list implies that all files and directories must be tracked.

   For example, this list configures the job to trigger a build when a user adds or updates an `.html`, `.jpeg`, or `.gif` file in the `myapp/src/main/web/` directory of the specified Git repository.

   ```
   myapp/src/main/web/.*\.html
   ```

   ```
   myapp/src/main/web/.*\.jpeg
   ```

   ```
   myapp/src/main/web/.*\.gif
   ```

6. To specify a list of files and directories not to track for changes in the repository, expand **Advanced Git Settings**, and enter the list in **Excluded Region**.

   You can use regular expressions (regex) or glob to specify fileset. Each entry must be separated by a new line. An empty list implies that all files and directories must be tracked. If a change occurs in any of the specified files or directories, a build won't run. If there's an overlap between included and excluded regions, exclusions take precedence over inclusion.

7. To exclude users whose commits to the repository don't trigger builds, in **Excluded User**, enter the list of user names.

Trigger a Build Automatically on SCM Polling

SCM polling enables you to configure a job to periodically check the job's Git repositories for any commits pushed since the job's last build. If updates are found, it triggers a build. You can configure the job and specify the schedule in Coordinated

Universal Time (UTC). UTC is the primary time standard by which the world regulates clocks and time.

You specify the schedule using Cron expressions. If you're not a Cron expert, use the novice mode and set the schedule by specifying values. If you're a Cron expert, use the Expert mode.

1. Open the job's configuration page.
2. In the **Git** tab, add the Git repository.
3. Click **Settings**  .
4. Click the **Triggers** tab.
5. Click **Add Trigger** and select **SCM Polling Trigger**.
6. To use the expert mode, select the **Expert mode** check box and enter the schedule in the text box.
   The default pattern is `0/30 * * * *`, which runs a build every 30 minutes.

   After you edit the expression, it's validated immediately when the cursor moves out of the text box. Note that other fields of the section aren't available when the check box is selected.
7. To use the novice mode, deselect the **Expert mode** check box and specify the schedule information. The page displays the generated Cron expression next to the **Expert mode** check box.
8. To use the novice mode, deselect the **Expert mode** check box and specify the schedule information in **Minute**, **Hour**, **Day of the Month**, **Month**, and **Day of the Week**.
   Click **Toggle Recurrence** to add or remove `0/` or `1/` at the beginning of the value in the Cron expression.

   The page displays the generated Cron expression next to the **Expert mode** check box.

   > **Tip:**
   >
   > To check the job's Git repositories every minute, deselect all check boxes. Remember that this may consume large amounts of system resources.
9. If necessary, in **Comment**, enter a comment.
10. To view and verify the build schedule of the next ten builds, from the timezone drop-down list, select your time zone and then click **View Schedule**.
11. Click **Save**.

To see the SCM poll log of the job after the build runs, in the job's details page or the build's details page, click **SCM Poll Log**  .

Generate Cron Expressions

You can use Cron expressions to define periodic build patterns.

For more information about Cron, see `http://en.wikipedia.org/wiki/Cron`.

You can specify the Cron schedule information in the following format:

```
MINUTE HOUR DOM MONTH DOW
```

where:

- `MINUTE` is minutes within the hour (0-59)
- `HOUR` is the hour of the day (0-23)
- `DOM` is the day of the month (1-31)
- `MONTH` is the month (1-12)
- `DOW` is the day of the week (1-7)

To specify multiple values, you can use the following operators:

- `*` to specify all valid values
- `-` to specify a range, such as `1-5`
- `/` or `*/X` to specify skips of X's value through the range, such as `0/15` in the `MINUTE` field for `0,15,30,45`
- `A,B,...,Z` can be used to specify multiple values, such as `0,30` or `1,3,5`

Use External Git Repositories

If you use an external Git repository to manage source code files, you can configure a job to access its files when a build runs.

If the external Git repository is a public repository, mirror it in the project or use its direct URL in the job configuration. If the external Git repository is a private repository, you must mirror it in the project. See Mirror an External Git Repository.

To configure a job to use an external Git repository:

1. Open the job's configuration page.
2. Click **Configure**  , if necessary.
3. Click the **Git** tab.
4. From the **Add Git** list, select **Git**.
5. If the external Git repository is mirrored with the project, in **Repository**, select the repository name. Note that the build executor uses the internal address URL of the mirrored repository.

   If the external Git repository isn't mirrored with the project, enter the repository's direct URL. Don't enter the direct URL of a private repository.
6. Configure other fields of the page and click **Save**.

To trigger a build on an update to the external Git repository, set up SCM polling according to the frequency of commits. DevCS can't trigger a build immediately on an update to the external Git repository.
Before you set SCM polling, note that if you use the internal address URL of a mirrored repository, there's a wait time of at least 5 minutes. If you use the external address URL or the direct URL of the repository, there's a wait time of at least 1 minute. Remember that polling every few minutes consumes large amounts of system resources.

Access Files of a Git Repository's Private Branch

To access a Git repository's private branch, configure the job to use SSH.

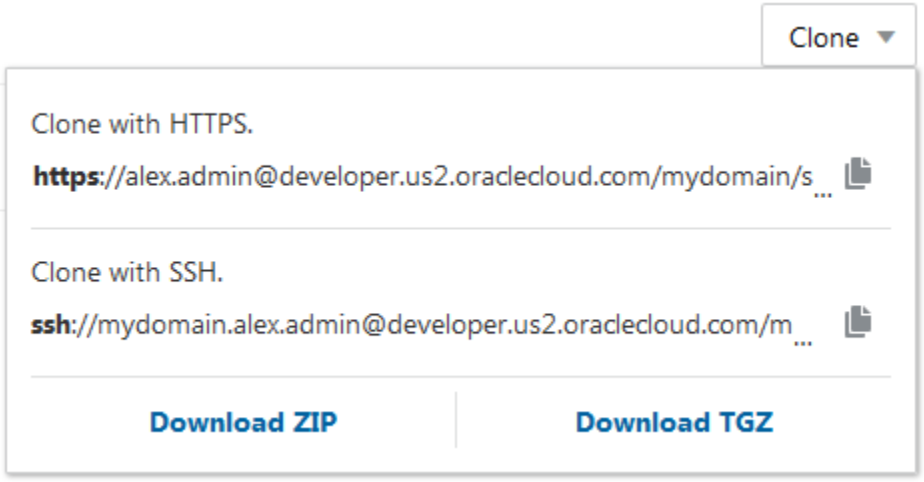
1. On the computer that you'll use to access the Git repository, generate a SSH key pair and upload its private key to DevCS. See Upload Your Public SSH Key. Make sure that the private key on your computer is accessible to the Git client.

   Ignore this step if you've already uploaded the SSH public key.

2. Copy the Git repository's SSH URL.

   On the Git page, from the **Repositories** drop-down list, select the Git repository.

   From the **Clone** drop-down list, click **Copy to clipboard** to copy the SSH URL.



3. Open the job's configuration page.
4. Click **Configure**, if necessary.
5. Click the **Git** tab.
6. From the **Add Git** list, select **Git**.
7. In **Repository**, paste the SSH URL of the Git repository.
8. In **Branch**, select the private branch.
9. Click the **Before Build** tab.
10. Click **Add Before Build Action** and select **SSH Configuration**.
11. In **Private Key** and **Public Key**, enter the private and the public key of your SSH Private-Public key pair.

    Leave the **Public Key** empty to use fingerprint.

12. In **Pass Phrase**, enter the pass phrase of your SSH Private-Public key pair. Leave it empty if the keys aren't encrypted with a pass phrase.
13. Continue to configure the job, as desired.
14. Click **Save**.

Publish Git Artifacts to a Git Repository

Git artifacts, such as tags, branches, and merge results can be published to a Git repository as a post-build action.

1. Open the job's configuration page.
2. Click **Configure**  .
3. In the **Git** tab, add the Git repository where you want to publish Git artifacts.
4. Click the **After Build** tab.
5. Click **Add After Build Action** and select **Git Publisher**.
6. To push Git artifacts to the Git repository only if the build succeeds, select the **Publish only if build succeeds** check box.
7. To push merges back to the target remote name, select the **Merge results** check box.
8. To push a tag to the remote repository, in **Tag to push**, specify the Git repository tag name. You can also use environment variables. In **Target remote name**, specify the target remote name of the Git repository where the tag is pushed. By default, `origin` is used.

   The push fails if the tag doesn't exist. Select the **Create new tag** check box to create the tag and enter a unique tag name.
9. To push a branch to the remote repository, in **Branch to push**, specify the Git repository branch name. You can also use environment variables. In **Target remote name**, specify the target remote name of the Git repository where the branch is pushed. By default, `origin` is used.
10. Click **Save**.

Advanced Git Options

When you configure the Git repositories of a job, you can also configure the job with some advanced Git options, such as change the remote name of the repository, set the checkout directory in the workspace, and whether to clean the workspace before a build runs.

You can perform these configuration actions from the **Git** tab of the job configuration page.

Action	How To
Change the remote name of a repository	For the Git repository, expand **Advanced Repository Options**, and specify the new name in **Name**. The default remote name is `origin`.
Specify a reference specification of a repository	A reference repository helps to speed up the builds of the job by creating a cache in the workspace and hence reducing the data transfer. When a build runs, instead of cloning the Git repository from the remote, the build executor clones it from the reference repository. To create a reference specification of a Git repository, expand **Advanced Repository Options**, and specify the name in **Ref Spec**. Leave the field empty to create a default reference specification.

Action	How To
Specify a local checkout directory	The local checkout directory is a directory in the workspace where the Git repository is checked out when a build runs. To specify the directory of a Git repository, expand **Advanced Repository Options**, and specify the path in **Local Checkout Directory**. If left empty, the Git repository is checked out on the root directory of the workspace.
Checkout the remote repository's branch and merge it into a local branch	Expand **Advanced Git Settings**, in **Merge another branch**, specify the branch name to merge to. If specified, the build executor checks out the revision to build as `HEAD` on this branch. If necessary, in **Checkout revision**, specify the branch to checkout and build as `HEAD` on the value of **Merge another branch**.
Configure Git user.name and user.email variables	Expand **Advanced Git Settings** and in **Config user.name** and **Config user.email**, specify the user name and the email address.
Merge to a branch before a build runs	Expand **Advanced Git Settings** and select the **Merge from another repository** check box. In **Repository**, enter or select the name of the repository to be merged. In **Branch**, enter or select the name of the branch to be merged. If no branch is specified, the default branch of the repository is used. The build runs only if the merge is successful.
Prune obsolete local branches before running a build	Expand **Advanced Git Settings** and select the **Prune remote branches before build** check box.
Skip the internal tag	When a build runs, the build executor checks out the Git repository to the local repository of the workspace and applies a tag to it. To skip this process, expand **Advanced Git Settings** and deselect **Skip internal tag** check box.
Remove untracked files before running a build	Expand **Advanced Git Settings** and select the **Clean after checkout** check box.
Retrieve sub-modules recursively	Expand **Advanced Git Settings** and select the **Recursively update submodules** check box.
Display commit's author in the log	By default, the Git change log shows the commit's `Committer` . To display commit's `Author`, expand **Advanced Git Settings** and select the **Use commit author in changelog** check box.
Delete all files of the workspace before a build runs	Expand **Advanced Git Settings** and select the **Wipe out workspace before build** check box.

View SCM Changes Log

The SCM Change log displays the files that were added, edited, or removed from the job's Git repositories before the build was triggered.

You can view the SCM changes log from the job's details page and a build's details page. The Recent SCM Changes page that you open from the job's details page displays SCM commits from last 20 builds in the reverse order. The SCM Changes page that you open from a build's details page displays SCM commits that happened after the previous build.

The log shows build ID, commit messages, commit IDs, and affected files.

Trigger a Build Automatically on a Schedule

You can configure a job to run builds on a specified schedule specified in Coordinated Universal Time (UTC). UTC is the primary time standard by which the world regulates clocks and time.

You can specify the schedule using Cron expressions. If you're not a Cron expert, use the novice mode and set the schedule by specifying values. If you're a Cron expert, use the Expert mode.

1. Open the job's configuration page.
2. Click **Settings**  .
3. Click the **Triggers** tab.
4. Click **Add Trigger** and select **Periodic Build Trigger**.
5. To use the expert mode, select the **Expert mode** check box, and enter the schedule in the text box.

   The default pattern is `0/30 * * * *`, which runs a build every 30 minutes.

   After you edit the expression, it's validated immediately when the cursor moves out of the text box. Note that other fields of the section aren't available when the check box is selected.
6. To use the novice mode, deselect the **Expert mode** check box and specify the schedule information. The page displays the generated Cron expression next to the **Expert mode** check box.
7. To use the novice mode, deselect the **Expert mode** check box and specify the schedule information in **Minute**, **Hour**, **Day of the Month**, **Month**, and **Day of the Week**.

   Click **Toggle Recurrence** to add or remove `0/` or `1/` at the beginning of the value in the Cron expression.

   The page displays the generated Cron expression next to the **Expert mode** check box.
8. If necessary, in **Comment**, enter a comment.
9. To view and verify the build schedule of the next ten builds, from the timezone drop-down list, select your time zone and then click **View Schedule**.
10. Click **Save**.

Use Build Parameters

Using build parameters, you can pass additional information when a build runs that is not available at the time of job configuration.

You can configure a job to use a parameter and its value as an environment variable or through variable substitution in other parts of the job configuration. When a build runs, a Configure Parameters dialog box opens so you can enter or change the default values of the parameters.

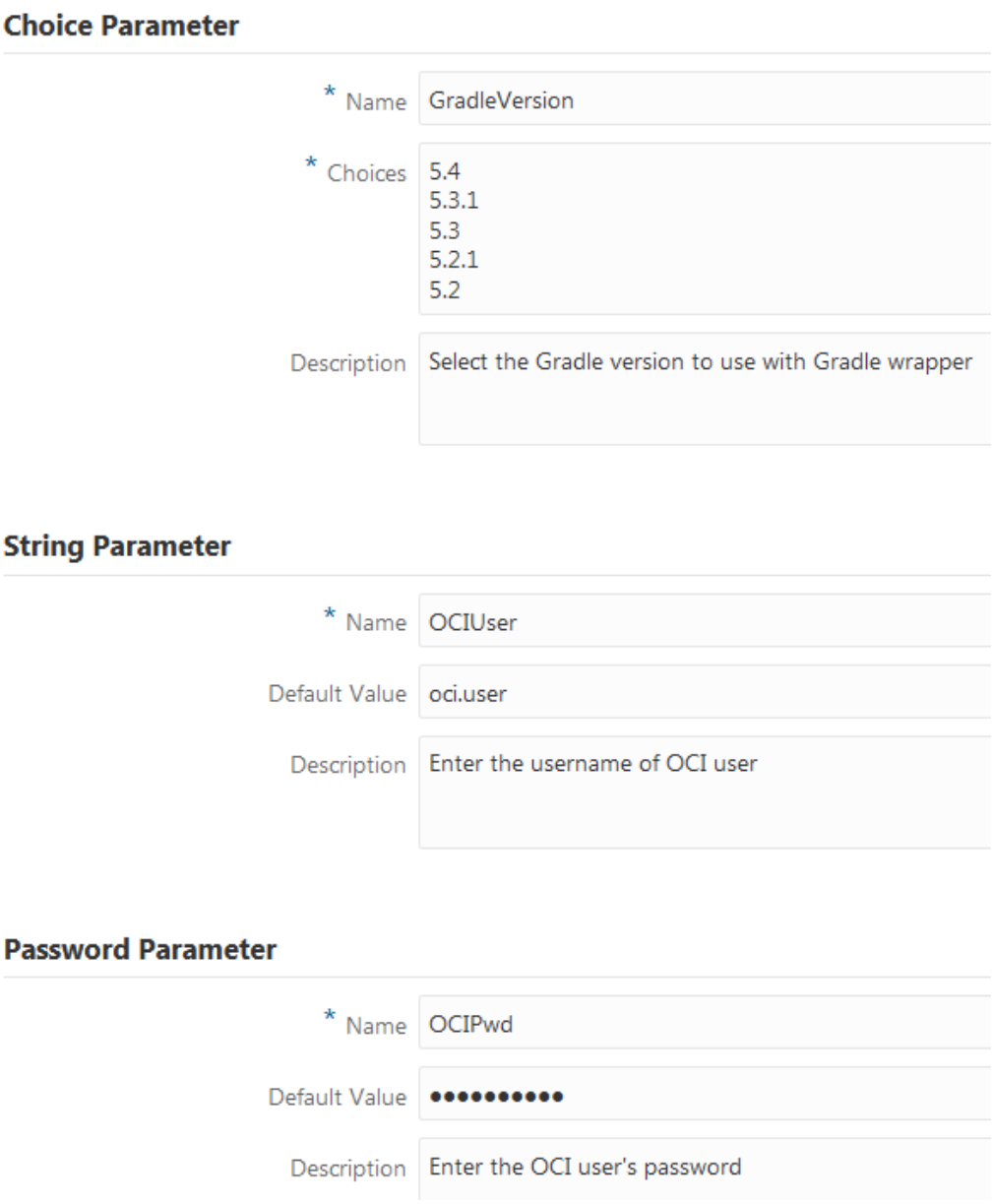
1. Open the job's configuration page.

2. Click **Configure**  .
3. Click the **Parameters** tab.
4. From the **Add Parameter** drop-down list, select the parameter type.
   You can add these types of build parameters:

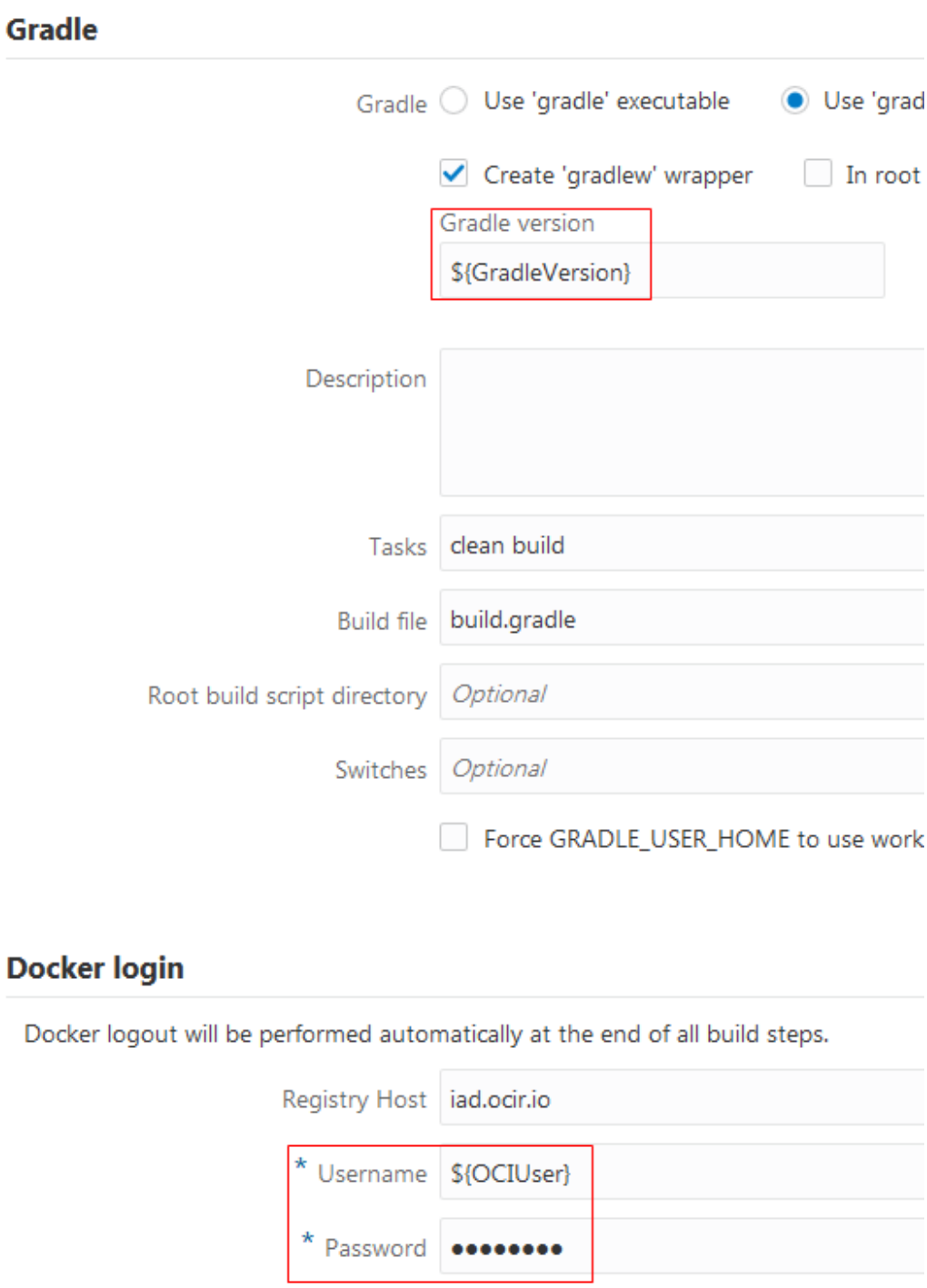
Use this parameter type ...	To:
String	Accept a string value from the user when a build runs. The parameter field appears as a text box in the Configure Parameters dialog box.
Password	Accept a password value from the user when a build runs. The parameter field appears as a password box in the Configure Parameters dialog box.
Boolean	Accept `true` or `false` as input from the user when a build runs. The parameter field appears as a check box in the Configure Parameters dialog box.
Choice	Accept a value from a list of values when a build runs. The parameter field appears as a drop-down list in the Configure Parameters dialog box. The first value is the default selected value.
Merge Request	Accepts string values for the Git repository URL, the Git repository branch name, and the merge request ID as input. The parameter fields appear as a text box in the Configure Parameters dialog box. Use this parameter if you want to link an existing job with a merge request.

5. Enter values, such as name, default value, and description.
6. Click **Save**.

For example, if you want a job to change the default values of Gradle's version, OCI username, and OCI user's password when a build runs, create the Choice, String, and Password build parameters to accept the values. The Password parameter's value isn't displayed in the input field.

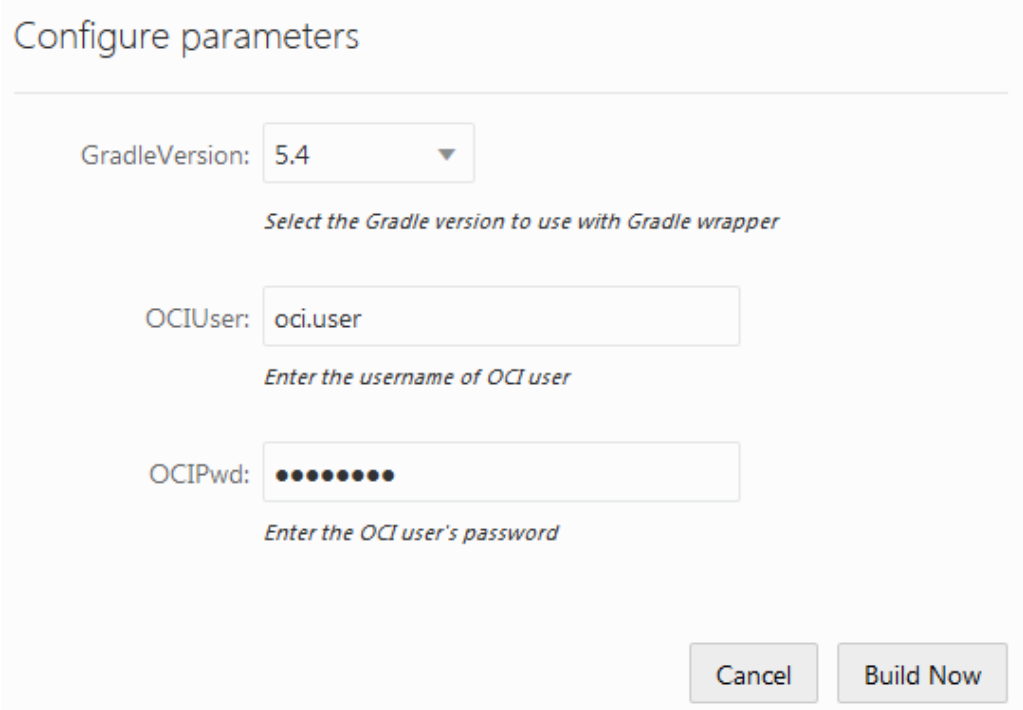


To use a build parameter, use it in the `${BUILD_PARAMETER}` format. For example, this image shows the Gradle version, OCI username, and OCI password parameters used in the build step fields of a job. Note that the password parameter's variable name isn't displayed.



When a build runs, the Configure Parameters dialog box opens where you can enter or change the default values of parameters. All parameter values, except the Password parameter's value,display as string in the dialog box and later in the build log.

Example:



If you selected the **Use for Merge Request** check box while creating the job, GIT_REPO_URL, GIT_REPO_BRANCH, and MERGE_REQ_ID Merge Request parameters are automatically added to accept the Git repository URL, the Git repository branch name, and the merge request ID as input from the merge request, respectively. The GIT_REPO_URL and GIT_REPO_BRANCH variables are automatically set in the **Repository** and **Branch** fields of the **Git** tab.

When a job in a pipeline runs, there is no way to enter or change the default values of the parameters. Job parameters in pipelines exhibit the following implicit behaviors:

- Upstream job parameters are passed to downstream jobs.
  For example, in a pipeline that flows from Job A to Job B to Job C, if parameter P1 is defined in Job A and parameter P2 is defined in Job B, then parameter P1 will be passed to Job B and parameters P1 and P2 will be passed to Job C.
- An upstream job with the same named parameter as a downstream job will overwrite the default value of the named parameter from the downstream job.
  For example, if parameters P1 and P2 are defined in Job A and parameters P2 and P3 are defined in Job B, then the value of parameter P2 from Job A will overwrite the default value of parameter P2 in Job B. If there was a Job C downstream from Job B, then the initial default value of P2 (from Job A) plus the values of P1 and P3 would be passed to Job C.
- When a build of the pipeline runs, the Configure Parameter dialog box displays all parameters of the jobs in the pipeline. Duplicate parameters are displayed once and its value is used by all jobs that use the parameter. The default value of a duplicate parameter comes from the first job in the pipeline where it is defined.
  For example, in a pipeline that flows from Job A to Job B to Job C, if parameter P1 is defined in Job A; parameters P2 and P3 are defined in Job 2; and parameters P1 and P4 are defined in Job C; then when the pipeline runs, it displays parameters P1, P2, P3, and P4 once in the Configure Parameter dialog box

though parameter P1 is defined in two jobs. The default value of P1 would come from Job 1 and is passed to subsequent jobs of the pipeline.

In the pipeline, if the **Auto start when pipeline jobs are built externally** is selected, then the Configure Parameter dialog box isn't displayed when a build of a pipeline's job runs. In the pipeline, the subsequent jobs of the job that trigger the build use the default values of their parameters. If a parameter is duplicate, then the job uses the default value of the first job where the parameter was defined.

For example, in a pipeline that flows from Job A to Job B to Job C, if parameter P1 is defined in Job A; parameters P2 and P3 are defined in Job B; and parameters P1 and P4 are defined in Job C; then when a build of Job A runs, it passes the default value of P1 to Job B and Job C, and overwrites the default of P1 in Job C. If a build of Job B runs, then the builds use the default values of P2, P3, P1 (defined in Job C) and P4.

To learn about how to use build parameters in a Shell build step, see the GNU documentation on Shell Parameter Expansion at `https://www.gnu.org/software/bash/manual/html_node/Shell-Parameter-Expansion.html`.

Access an Oracle Cloud Service Using SSH

You can configure a job to use SSH to access any Oracle Cloud service instances that has SSH enabled, such as Oracle Cloud Infrastructure Compute Classic VMs.

You can configure the job to use any of the following options, or both.

- Create an SSH tunnel to access a process running on a remote system, including an on-premise system, via the SSH port. The SSH tunnel is created at the start of the build job and is destroyed automatically when the job finishes.
- Set up the default `~/.ssh` directory with the provided keys in the build's workspace for use with the command-line tools. The modifications revert when the job finishes.

To connect to the Oracle Cloud service instance, you need IP address of the server, credentials of a user who can connect to the service instance, and local and remote port numbers.

1. Open the job's configuration page.
2. Click **Configure**  , if necessary.
3. Click the **Before Build** tab.
4. Click **Add Before Build Action** and select **SSH Configuration**.
5. In **Private Key** and **Public Key**, enter the private and the public key of your SSH Private-Public key pair.

   Leave the **Public Key** empty to use fingerprint.
6. In **Pass Phrase**, enter the pass phrase of your SSH Private-Public key pair. Leave it empty if the keys aren't encrypted with a pass phrase.
7. In **SSH Server Public Key**, enter the public key of the SSH server.

   If you're using a command-line SSH tool, note that the host name and the IP address must match. The host name and the IP address can be comma separated. Example: `ssh1.example.com,10.0.0.13 ssh-rss ...`.

Leave the field empty to skip host verification. For command-line tools, such as `ssh`, add the `-o StrictHostKeyChecking=no -o UserKnownHostsFile=/dev/null` option explicitly to skip host verification.

8. To use an SSH tunnel, select the **Create SSH Tunnel** check box.

   SSH tunnel provides an additional layer of security and can only be set up between trusted hosts. After you select the check box, enter the SSH server details.

   - **Username**: Name of the user who can connect to the SSH server.
   - **Password**: Password of the SSH user. Leave the field empty to use the key based authentication.
   - **Local Port**: Port number of the client used for local port forwarding.
   - **Remote Host**: Name of the remote host, or an interface on the SSH server.
   - **Remote Port**: Port number of the remote host or interface.
   - **SSH Server**: Name or IP address of the target SSH server.
   - **Connect String**: Displays the connect string to be used to set up the SSH tunnel.

9. To use command line tools (such as `ssh`, `scp`, or `sftp`), select the **Setup files in ~/.ssh for command-line ssh tools** check box.

   When a build runs, necessary files with the information that you've provided are created for you in the `known_hosts` file of the `~/.ssh` directory in the build system workspace. The files are removed automatically after the build is complete.

10. Click **Save**.

Access the Oracle Maven Repository

The Oracle Maven Repository contains artifacts, such as ADF libraries, provided by Oracle. You may require these artifacts to compile, test, package, perform integration testing, or deploy your applications. For more information about the Oracle Maven repository, see `https://maven.oracle.com/doc.html`.

To build your applications and access the Oracle Maven Repository, you configure the job and provide your credentials to access the repository.

1. Open `https://www.oracle.com/webapps/maven/register/license.html` in your web browser, sign-in with your Oracle Account credentials, and accept the license agreement.
2. Configure the POM file and add the Oracle Maven Repository details.
   a. Add a `<repository>` element to refer to `https://maven.oracle.com`.

      Example:

```
<repositories>
  <repository>
    <name>OracleMaven</name>
    <id>maven.oracle.com</id>
    <url>https://maven.oracle.com</url>
  </repository>
</repositories>
```

Depending on your application, you may also want to add the `<pluginRepository>` element to refer to `https://maven.oracle.com`.

Example:

```
<pluginRepositories>
  <pluginRepository>
    <name>OracleMaven</name>
    <id>maven.oracle.com</id>
    <url>https://maven.oracle.com</url>
  </pluginRepository>
</pluginRepositories>
```

b. Commit the POM file to the project Git repository.

3. If you're a project owner, set up Oracle Maven Repository connections for the project's team members.
4. Create and configure a job to access Oracle Maven Repository.

Create and Manage Oracle Maven Repository Connections

If your project users access the Oracle Maven Repository frequently, you can create a pre-defined connection for them. Project users can then configure a job and use the connection to access the artifacts of the Oracle Maven Repository while running builds.

To create a connection, you'd need the Oracle Technology Network (OTN) Single Sign-On (SSO) credentials of a user who has accepted the Oracle Maven Repository license agreement.

You must be assigned the project **Owner** role to add and manage Oracle Maven Repository connections.

Action	How To
Add an Oracle Maven Repository connection	1. In the navigation bar, click **Project Administration**. 2. Click **Build**. 3. Click the **Maven Connection** tab. 4. Click **Add Maven Connection**. 5. In the Create Maven Connection dialog box, in **Connection Name**, enter a unique name. 6. In **OTN Username** and **OTN Password**, enter the credentials of a user who has accepted the Oracle Maven Repository license agreement. 7. In **Server Id**, if necessary, enter the ID to use for the `<server>` element of the Maven `settings.xml` file or use the default `maven.oracle.com` ID. 8. Click **Create**.

Action	How To
Edit a connection	To change the connection's user credentials or provide another server ID, you can edit the connection. 1. In the navigation bar, click **Project Administration**  . 2. Click **Build**. 3. Click the **Maven Connection** tab 4. Click the connection name and then click the Edit icon. 5. In the Edit Maven Connection dialog box, if necessary, enter the credentials of a user with valid SSO user name. In **Server Id**, if necessary, enter the ID to use for the `<server>` element of the Maven `settings.xml` file. If not provided, the ID defaults to `maven.oracle.com`. 6. Click **Update**.
Delete the connection	1. In the navigation bar, click **Project Administration**  . 2. Click **Build**. 3. Click the **Maven Connection** tab 4. Click the connection name and then click  . 5. In the Delete Maven Connection dialog box, click **Delete**.

Configure a Job to Connect to the Oracle Maven Repository

1. Open the job's configuration page.
2. Click **Configure**  .
3. Click the **Before Build** tab.
4. Click **Add Before Build Action** and select **Oracle Maven Repository Connection**.
5. From **Use Existing Connection**, select a pre-defined connection. Your project owner has created a connection so that you don't have to worry about setting it up. If there's no pre-defined connection available or you want set up your own connection, click the toggle button. In **OTN Username** and **OTN Password**, enter the credentials of a user who has accepted the Oracle Maven Repository license agreement.
6. In **Server Id**, if required, enter the ID to use for the `<server>` element of the Maven `settings.xml` file, or use the default `maven.oracle.com` ID.
7. If you're using a custom `settings.xml` file, in **Custom settings.xml**, enter the file's path.
8. Click **Save**.

Run UNIX Shell Commands

You can configure a job to run a shell script or commands when a build runs.

1. Open the job's configuration page.
2. Click **Configure**  .
3. Click the **Steps** tab.
4. From **Add Step**, select **Unix Shell**.
5. In **Script**, enter the shell script or commands.

   The script runs with the workspace as the current directory. If there is no header line, such as `#!/bin/sh` specified in the shell script, then the system shell is used. You can also use the header line to write a script in another language, such as Perl (`#!/bin/perl`) or control the options that shell uses.

   You can also use Kubernetes, PSMcli, Docker, Terraform, Packer, and OCIcli commands in the Shell script. Make sure that you have the required software in the job's Build VM template before you run a build.

6. To show the values of the variables and hide the input-output redirection in the build log, select the **(-x) Expand variables in commands, don't show I/O redirection** option.

   To show the command as-it-is in the build log, select the **(-v) Show commands exactly as written** option.

7. Click **Save**.

Tip:

- By default, when a build runs, it invokes the shell with the `-ex` option. It prints all commands before they run. The build fails if any of the commands exits with a non-zero exit code. You can change this behavior by adding the `#!/bin/...` line in the shell script.
- Don't enter a long shell script. For a long script, create a script file, add it to the Git repository, and run the script with a command (such as `bash -ex /myfolder/myscript.sh`).
- To run Python 3, create an isolated environment using virtualenv. To learn more, see `https://virtualenv.pypa.io/en/stable/`.

  For example, add these commands as a Shell build step to create a virtual environment:

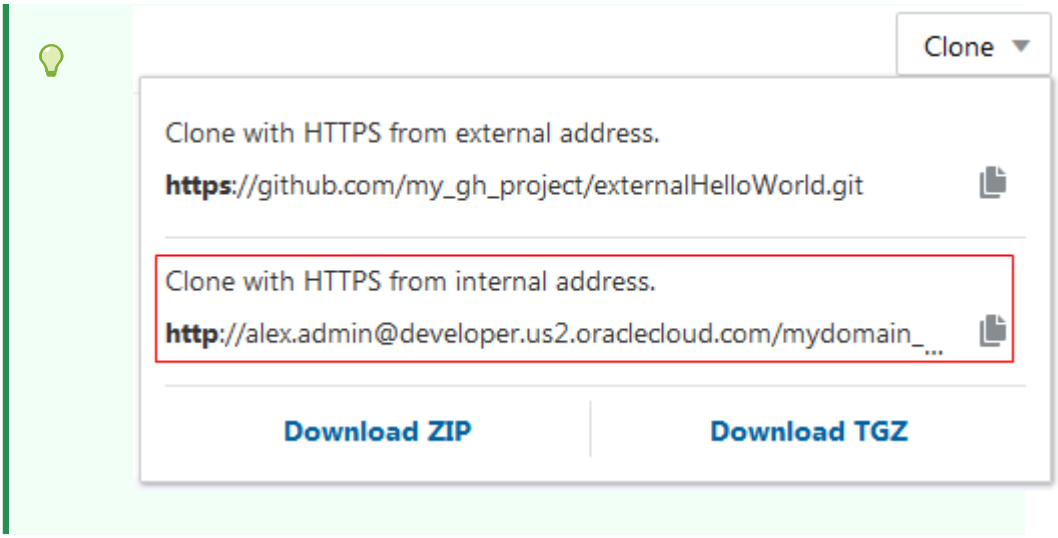
  ```
  pip3 list
  cd $WORKSPACE
  python3 -m venv mytest
  cd mytest/bin
  ./pip3 list
  ./pip3 install --upgrade pip requests setuptools selenium
  ./pip3 list
  ./python3 -c 'import requests; r=requests.get('\''https://
  www.google.com'\''); print(r.status_code)'
  ./pip3 uninstall -y requests
  ./pip3 list
  ```

- If both Python 2 and Python 3 are available in the job's Build VM template, to call Python, use these commands.

Command	Version
`python` `python2`	Python 2
`python3`	Python 3
`pip`	`pip` of Python 3
`pip3`	`pip` of Python 3

- To clone an external Git repository using a shell command, use the internal URL of the external Git repository. To copy the URL, open the **Git** page and, from the **Repositories** drop-down list, select the external Git repository. From the **Clone** menu, click **Copy to clipboard** of the **Clone with HTTPS from internal address URL**.

  If you're using an Oracle Linux 6 VM in your job, remove the username from the URL before using it in a shell command. For example, if the copied URL is `https://alex.admin@developer.us2.oraclecloud.com/mydomain-usoracle22222/s/developer1111-usoracle22222_myproject/scm/myextrepo.git`, then remove `alex.admin@` from the URL and use `https://developer.us.oraclecloud.com/mydomain-usoracle22222/s/mydomain-usoracle22222_myproject/scm/mydomain-usoracle22222_myproject.git` in your shell command.



Build Maven Applications

Using Apache Maven, you can automate your build process and download dependencies, as defined in the POM file.

1. Upload the Maven POM files to the project Git repository.
2. Open the job's configuration page.
3. Click **Configure** .
4. In the **Git** tab, add the Git repository where you uploaded the build files.
5. Click the **Steps** tab.
6. From **Add Step**, select **Maven**.
7. In **Goals**, enter Maven goals, or phases, along with their options. By default, `clean` and `install` goals are added.
   For more information about Maven goals, see the Maven Lifecycle Reference documentation at `http://maven.apache.org`.
8. In **POM file**, enter the Maven POM file name and path, relative to the workspace root. The default value is **pom.xml** at the Git repository root.
9. If necessary, specify the advanced Maven options.

Action	How To
Use a private repository for builds	Select the **Use Private Repository** check box. You may want to use it to make sure that other Maven build artifacts don't interfere with the artifacts of this job's builds. When a build runs, it creates a Maven repository `.maven/repo` directory in the build executor workspace. Remember selecting this option consumes more storage space of the workspace.
Use a private temporary directory for builds.	Select the **Use Private Temp Directory** check box. You may want to use it to create a temporary directory for artifacts or temporary files. When a build runs, it creates a `.maven/tmp` directory in the workspace. The temporary files may consume large amount of storage, so, remember to clean up the directory regularly.

Action	How To
Work offline and don't access remote Maven repositories	Select the **Offline** check box.
Activate Maven profiles	In **Profiles**, enter a list of profiles, separated by commas. For more information about Maven profiles, see the Maven documentation at `http://maven.apache.org`.
Set custom properties	In **Properties**, enter custom system properties in the `key=value` format, specifying each property on its own line. When a build runs, the properties are passed to the build executor in the standard way (example: `-Dkey1=value1 -Dkey2=value2`)
Set the Maven verbosity level	From **Verbosity**, select the level. You may want to use it to set the verbosity of the Maven log output to the build log.
Set the checksum mode	From **Checksum**, select the mode. You may want to use it to set the check-sum validation strictness when the build downloads artifacts from the remote Maven repositories.
Set handling of the SNAPSHOT artifacts	From **Snapshot**, select the mode.
Include other Maven projects to the reactor	In **Projects**, enter the comma or space separated list of Maven project jobs to include in the reactor. The reactor is a mechanism in Maven that handles multi-module projects. A project job can be specified by `[groupId]:artifactId` or by its relative path.
Resume a Maven project from the reactor	In **Resume From**, enter the Maven job project name from where you would like to resume the reactor. The Maven job project can be specified by `[groupId]:artifactId` or by its relative path.
Set the failure handling mode	From **Fail Mode**, select the mode. You may want to use it to set how the Maven build proceeds in case of a failure.
Set the Make-like reactor mode	From **Make Mode**, select the mode. You may want to use it enable Make-like build behavior.
Configure the reactor threading model	In **Threading**, enter the value for experimental support for parallel builds. For example, a value of 3 indicates three threads for the build.
Pass parameters to Java VM	In **JVM Options**, enter the parameters. The build passes the parameters as `MAVEN_OPTS`.

10. Click **Save**.

Use the WebLogic Maven Plugin

The WebLogic server includes a Maven plugin that you can use to perform various deployment operations against the server, such as deploy, redeploy, and update. The plugin is available in the DevCS build executor.

When a build runs, the build executor creates an empty Maven repository in the workspace. To install the WebLogic plugin every time a build starts, in the job configuration, add a shell command to install the plugin and then deploy it.
For more information about how to use the WebLogic Maven plugin, see *Fusion Middleware Deploying Applications to Oracle WebLogic Server* in *Oracle Fusion Middleware Online Documentation Library*.

1. Open the job's configuration page.

2. Click **Configure** .
3. Click the **Steps** tab.
4. From **Add Step**, select **Unix Shell**.
5. In **Script**, enter this command:

```
mvn install:install-file -Dfile=$WLS_HOME/server/lib/weblogic-maven-plugin.jar -
DpomFile=$WLS_HOME/server/lib/pom.xml
mvn com.oracle.weblogic:weblogic-maven-plugin:deploy
```

6. Click **Save**.

Upload to or Download Artifacts from the Project Maven Repository

To upload artifacts to the Maven repository, you'll use the `distributionManagement` snippet in the POM file. To download artifacts from the Maven repository, use the `repositories` snippet in the POM file.

1. To upload a build artifact to the Maven repository, copy the `distributionManagement` snippet of the project's Maven repository and add it to the POM file.
   a. In the navigation bar, click **Maven** .
   b. On the right side of the page, click **Browse**.
   c. In the **Artifact Details** section, expand **Distribution Management**.
   d. In the **Maven** tab, click **Copy** to copy the `<distributionManagement>` code snippet to the clipboard.
   e. Open the POM file of your project in a code editor (or a text editor) and paste the contents of the clipboard under the `<project>` element.

      Example:

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.example.employees</groupId>
    <artifactId>employees-app</artifactId>
    <packaging>war</packaging>
    <version>1.0-SNAPSHOT</version>
    <name>employees-app Maven Webapp</name>
    <url>http://maven.apache.org</url>
    <properties>
        <tomcat.version>7.0.57</tomcat.version>
    </properties>

  <distributionManagement>
    <repository>
      <id>Demo_repo</id>
      <name>Demo Maven Repository</name>
      <url>https://developer.us2.oraclecloud.com/profile/my-org/s/my-org_demo_12345/maven/</url>
    </repository>
    <snapshotRepository>
      <id>Demo_repo</id>
      <name>DemoMaven Repository</name>
      <url>https://developer.us2.oraclecloud.com/profile/my-org/s/my-org_demo_12345/maven/</url>
    </snapshotRepository>
  </distributionManagement>
```

2. To download an artifact from the Maven repository, use the `repositories` snippet of the project's Maven repository.
   a. In the navigation bar, click **Maven** .
   b. On the right side of the page, click **Browse**.
   c. In the **Artifact Details** section, expand **Distribution Management**.

d. In the **Maven** tab, copy the `<repository>` element of the **Distribution Management** to the clipboard.

e. Open the POM file of your project in a code editor (or a text editor) and paste the `<repository>` element in the`<repositories>` element under `<project>`.

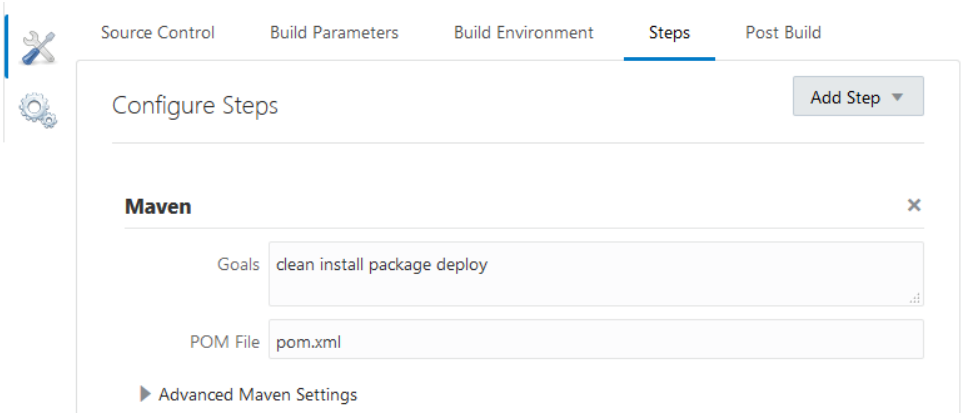
Example:

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/maven-v4_0_0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.example.employees</groupId>
    <artifactId>employees-app</artifactId>
    <packaging>war</packaging>
    <version>0.0.1-SNAPSHOT</version>
    <name>employees-app Maven Webapp</name>
    <url>http://maven.apache.org</url>

    <repositories>
      <repository>
           <id>Demo_repo</id>
           <name>Demo Maven Repository</name>
           <url>http://developer.us2.oraclecloud.com/profile/my-org/s/my-
org_demo_12345/maven/</url>
         </repository>
    </repositories>
.
.
.
</project>
```

3. Save the file, commit it to the Git repository, and then push the commit.
4. Configure the job to add a Maven step and add the required Maven goals.

Example:

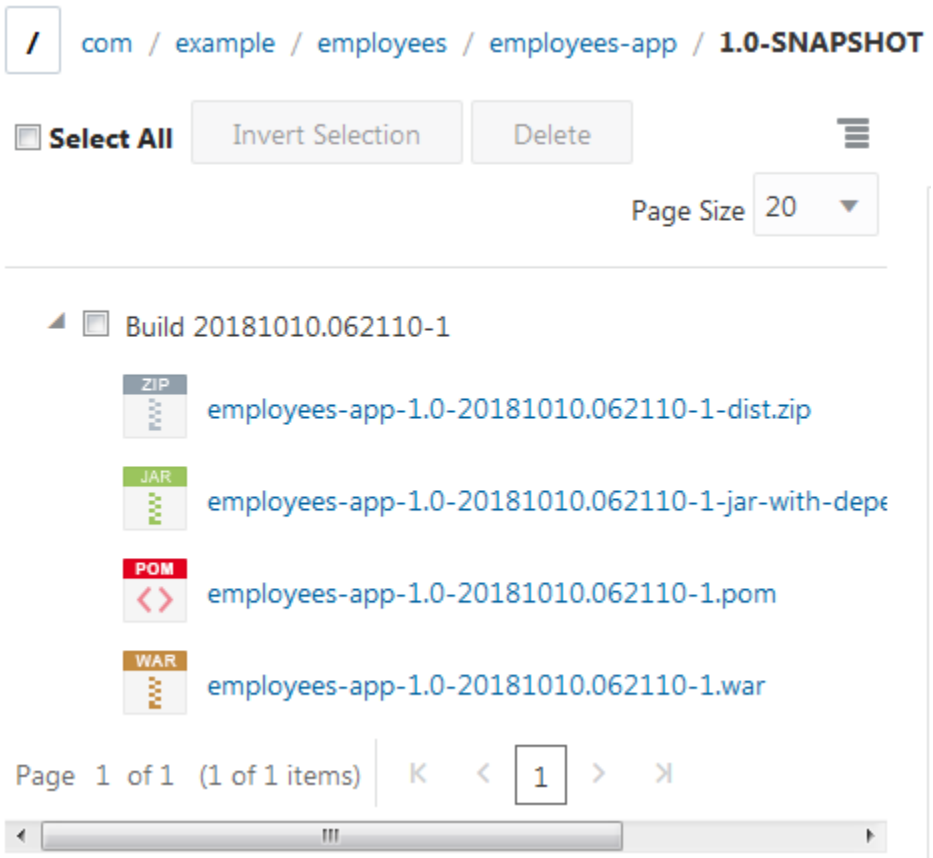


> **Tip:**
>
> Use the `deploy` goal to upload Maven artifacts to the project's Maven repository.

5. Run a build of the job.

6. If you configured the job to upload artifacts to the project's Maven repository, after the build is successful, verify the artifacts in the Maven page.

   Example:



Note that the credentials in `settings.xml` aren't required to access the project's Maven repository when running a build. The Build job has full access to the project's Maven repository for uploads and downloads.

Generate a Dependency Vulnerability Analysis Report

For a Maven application, you can configure a job to generate a Dependency Vulnerability Analysis (DVA) report, which helps you analyze any publicly known vulnerabilities in the application.

When a build runs, DevCS scans the job's POM file and checks the POM's direct and transitive dependencies against the National Vulnerability Database (NVD). For any vulnerabilities found, you can configure the job to mark the build as failed or file an issue. If email notifications are enabled or a Slack webhook is configured, you'd get a notification through email or Slack.
To find more about NVD, see `https://nvd.nist.gov/`.

1. Open the job's configuration page.
2. Click the **Before Build** tab.
3. From **Add Before Build Action**, select **Dependency Vulnerability Analyzer**.
4. In **Analyzer**, select **Enabled**.

5. In **Threshold at or above**, select the Common Vulnerability Scoring System (CVSS) score threshold.

   For example, if you select **Medium**, any vulnerability with a CVSS score of 4.0 or above is detected and reported.

   CVSS provides a way to capture the principal characteristics of a vulnerability and produce a numerical score reflecting its severity. For more details about how CVSS scores are calculated, see `https://nvd.nist.gov/vuln-metrics/cvss`.

6. In **CPE Confidence**, select the confidence rating the DVA has for the identified Common Platform Enumeration (CPE).

   CPE is a structured naming scheme for describing and identifying classes of applications, operating systems, and hardware devices present among an enterprise's computing assets. To find more about CPE, see `https://csrc.nist.gov/projects/security-content-automation-protocol/specifications/cpe/`.

   The CPE Confidence rating helps you filter the Common Vulnerabilities and Exposure (CVE) identifiers based on the confidence level. CVE is a list of common identifiers for publicly known cybersecurity vulnerabilities. To find more about CVE, see `https://cve.mitre.org/`.
   For example, select **Medium** to filter out the low confidence CVE identifiers from the report.

7. To fail the build if a vulnerability is detected, select the **Fail Build** check box.

8. To automatically file an issue for every POM file where a vulnerability is detected, select the **Create issue for every affected POM file** check box.

   In **Product** and **Component**, select the issue's product and component.

9. Click the **Steps** tab.

10. Add a **Unix Shell** step or a **Maven** step to build the POM file.

    For example, if you add a Shell step, enter the `mvn -install -fae -f app_dir/pom.xml -X` command to build the `pom.xml` file in the `app_dir` directory of the job's Git repository.

11. Click **Save**.

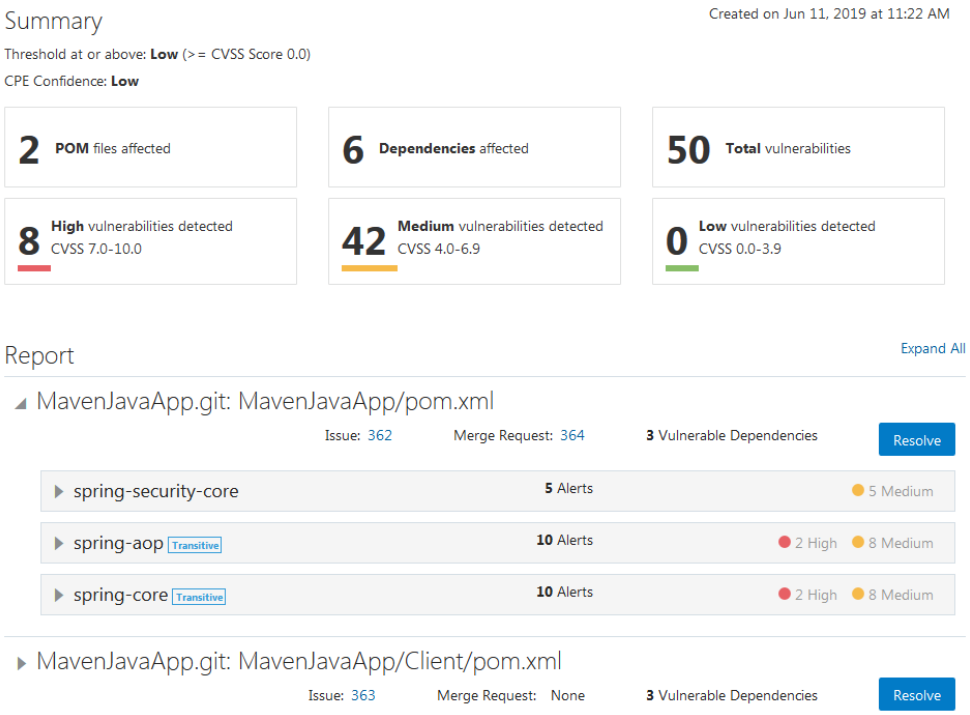
12. Run a build of the job.

    To trigger the build manually, in the Job Details page, click **Build Now**.

13. After the build is complete and a vulnerability is detected, click **Vulnerabilities**  to view the vulnerabilities report. If no vulnerability is detected, **Vulnerabilities**  is disabled.

14. On the Dependency Analyzer Summary page, review the affected POM files, dependencies, and the vulnerabilities detected.

    Expand the **Report** section to view the POM files of your application where vulnerabilities are found.
    Example:



After the DVA report is generated, expand each POM file in the **Report** section to view these details:

- Issue ID, if the **Create issue for every affected POM file** check box was selected. Click the issue link to open it.
  You can also open the **Project Home** page and check the recent activities feed about the issue's create notification. Example: **System created Defect 2: Vulnerabilities in -MavenJavaApp**. For a recurring issue, a comment is added to the issue and notification is added to the activities feed. Example: **System commented Defect 2: Vulnerabilities in - MavenJavaApp**.
- Merge request ID, if the **Resolve** button was clicked to resolve the vulnerabilities. Click the merge request link to open it.
- Number of vulnerabilities
- Name of each dependency where a vulnerability is found
- Each dependency's type (direct or transitive)
  If it's a transitive dependency, a **Transitive** label displays next to the name. If it's a direct dependency, no label is displayed.
- Number of alerts and alert categories of vulnerabilities (High, Medium, or Low)
- Expand each dependency to view its vulnerabilities
  If you want to mute alerts of a vulnerability, expand the vulnerability in the **Report** section, and click **Mute** in **Alerts**. In the Mute Vulnerability dialog box, review the details, and click **Mute**. For future builds, the alerts are muted in the job that generated the report.

  The **Mute** is available in the DVA report of the latest build only.

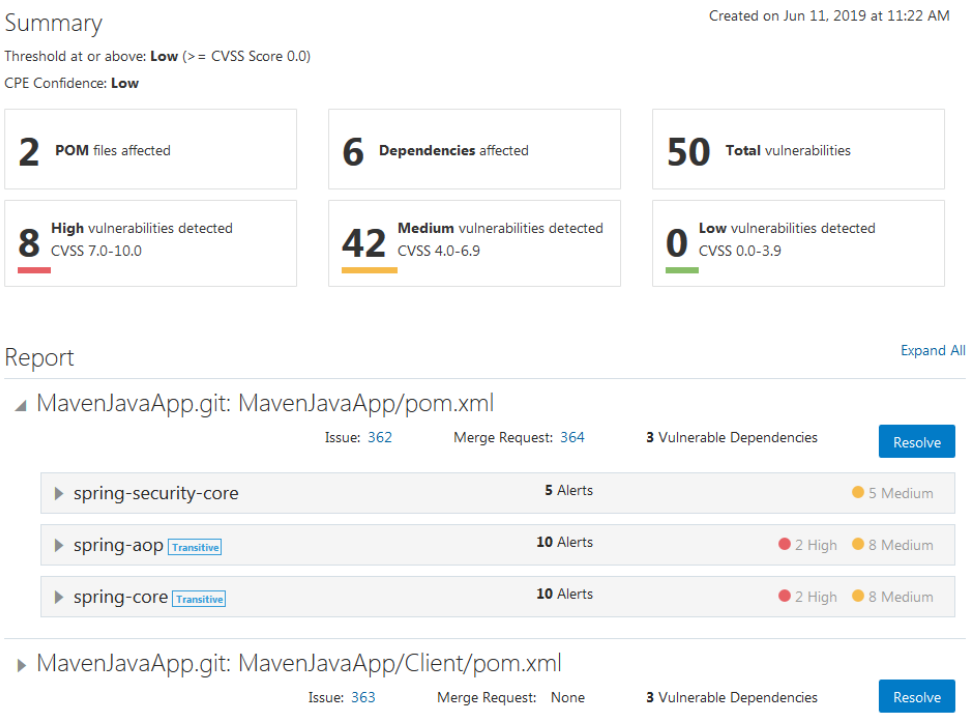
To fix a reported vulnerability, in the POM file, change the dependency's version number to a version where the vulnerability is fixed.

Resolve Reported Vulnerabilities Automatically

After the Dependency Vulnerability Analysis (DVA) report is generated, review the report to identify the vulnerabilities in the POM files, and click the **Resolve** button to resolve it.

The **Resolve** button helps you resolve vulnerabilities found in the direct dependencies of the POM file. The vulnerabilities found in the transitive dependencies must be resolved manually. If a POM file has transitive dependencies only and vulnerabilities are reported in it, the **Resolve** button is disabled. The button is not available in the DVA reports of older builds of the job, but only in the latest build of the job.

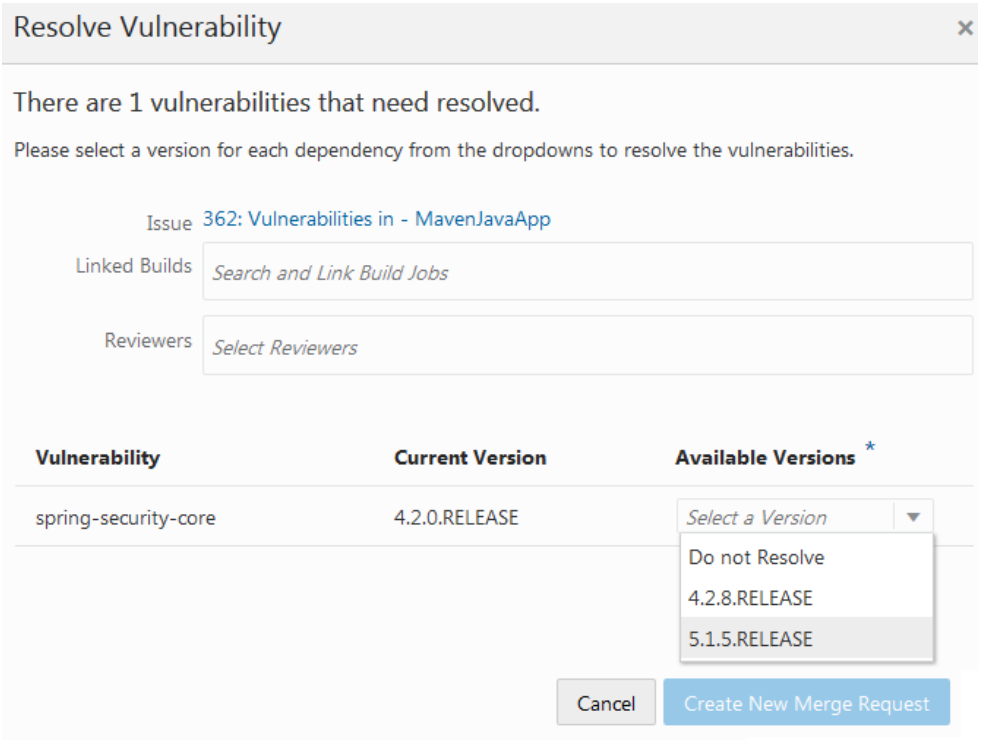
1. In the **Report** section of the vulnerability analysis report, expand the affected POM file.
   Example:



2. Click **Resolve**.
   If a merge request exists, you can cancel the dialog and use it or continue to create another merge request.
3. In the Resolve Vulnerability dialog box, review the reported vulnerabilities.
4. If an issue was created when the report was generated, its ID is displayed. If an issue wasn't created, select the **Create issue to track this resolution** check box to create it.
   In **Linked Builds**, add an existing build to link it to the merge request.

   In **Reviewers**, add team members to review the merge request.

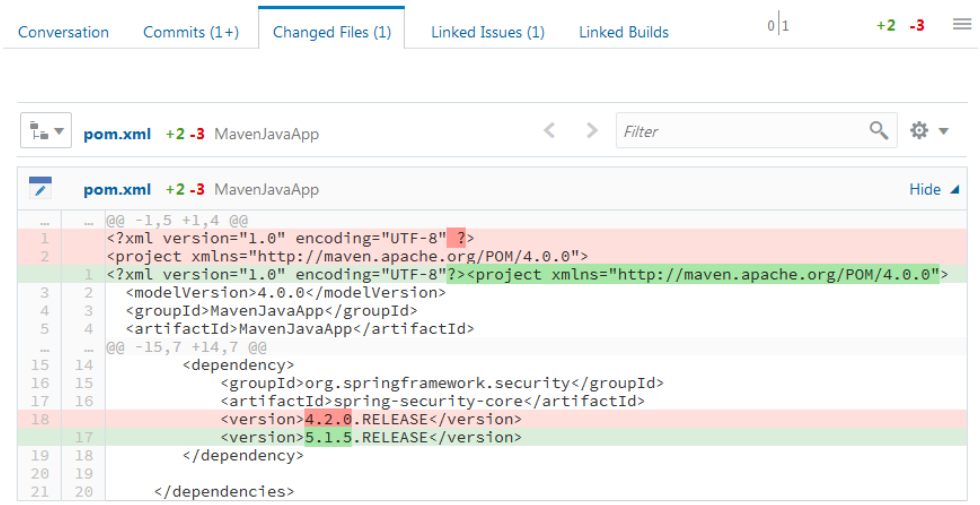
   Example:



5. For each vulnerability, in **Available Versions**, select a version of the dependency that doesn't have the reported vulnerability.
   If you don't want to resolve the dependency or no versions are available, select **Do Not Resolve**.
6. Click **Create New Merge Request**.
   When you click the button, DevCS does the following:
   a. Creates a merge request with details about the vulnerabilities found.
   b. Creates a branch with the job's Git repository branch as the base branch, and then sets it as the review branch of the merge request.
   c. Sets the job's Git repository branch as the target branch of the merge request.
   d. Updates the POM file of the review branch to use the specified versions of the dependencies.

   For example, if the job that generated the vulnerability report uses the `JavaMavenApp` Git repository and its `release1.1` branch, then a new branch is created in `JavaMavenApp` using `release1.1` as the base branch and is used as the review branch of the merge request. The `release1.1` branch is used as the target branch.

   If a merge request with same review and target branches was created in an older build of the job, DevCS uses the same merge request to merge the POM file updates.
7. Click the merge request link to open it in another tab or window of the browser, and click **OK**.
8. In the Merge Request, review the details of the vulnerabilities in the **Conversation** tab and the POM file changes in the **Changed Files** tab.
   Example:



9. If you've invited other reviewers, wait for their feedback.
10. If you've linked a build job to the merge request, in the **Linked Builds** tab, run a build and verify its stability.
11. When you're done and are ready to merge the POM file's updates, click **Merge**.
12. In the Merge dialog box, to delete the review branch, select the **Delete branch** check box. To resolve linked issues, select the **Resolve linked issues** check box and the check boxes of issues you want to resolve.
13. Click **Create a merge request**.
14. Run a build of the job that reported dependency vulnerabilities and verify that the POM file's update has fixed the vulnerability.
    If a vulnerability is still found, repeat the above steps to create another merge request and try another version of the dependency.

Build Ant Applications

Using Apache Ant, you can automate your build processes as described in its build files. For more information, see `https://ant.apache.org/`.

1. Upload the Ant build files (such as `build.xml` and `build.properties`) to the project Git repository.
2. Open the job's configuration page.
3. Click **Configure**  .
4. In the **Git** tab, add the Git repository where you uploaded the build files.
5. Click the **Steps** tab.
6. From **Add Step**, select **Ant**.
7. In **Targets**, specify the Ant targets or leave it empty to run the default Ant target specified in the build file.
8. In **Build File**, specify the path of the build file.
9. If necessary, in **Properties**, specify the values for properties used in the Ant build file.

Example:

```
# comment
name1=value1
name2=$VAR2
```

When a build runs, these are passed to Ant as `-Dname1=value1 -Dname2=value2`. You should always use `$VAR` for parameter references (don't use`%VAR%`). Use `\\`to escape a `\` and avoid using double-quotes ("). To define an empty property, use `varname=`in the script.

10. If your build requires a custom `ANT_OPTS`, specify it in **Java Options**. You may use it to specify Java memory limits (example: `-Xmx512m`). Don't specify other Ant options here (such as `-lib`), but specify them in **Targets**.
11. Click **Save**.

Build Gradle Applications

Using Gradle, you can automate your build processes as defined in its build script. For more information about Gradle, see `https://gradle.org/`.

In DevCS, Gradle 5 is available. To use another version of Gradle, use Gradle Wrapper in the Gradle build step. Gradle recommends using Gradle Wrapper as the preferred way to run a Gradle build. To learn more about using Gradle Wrapper, see `https://docs.gradle.org/current/userguide/gradle_wrapper.html`.

Set Up a Build VM and a Build VM Template with Gradle

Before you can use Gradle, you must create a Build VM template that includes Gradle and then add a Build VM that uses the VM template. If you don't want to create a template, you can add the software to an existing template. After adding a Build VM, create and configure a job to use the Build VM template and add Gradle commands.

 You must be the **Organization Administrator** to create a Build VM template and add a Build VM.

1. In the navigation bar, click **Organization**  .
2. Click the **Virtual Machines Templates** tab.
3. Click **+ Create Template**. Enter name and description, select the platform, and click **Create**.

   To use an existing template, click its name.
4. In the Software Catalog, search for `Gradle`.
5. Select the software tile and click **Done**.
6. Click **Build Virtual Machines** tab.
7. Click **+ Create VM**.
8. In the Add Build VM dialog box, in **Quantity**, specify the number of VMs you want to allocate. In **VM Template**, select the Gradle template.
9. Click **Add**.

Configure a Job to Run Gradle Commands

1. Upload the `build.gradle` file to a project's Git repository.
2. Open the job's configuration page.

   If you're creating a job, in **Software Template** of the New Job dialog box, select the Gradle template. Jump to step 5.
3. Click **Settings**  .
4. In the **Software** tab, select the Gradle template.
5. Click **Configure**  .
6. In the **Git** tab, add the Git repository where you uploaded the build file.
7. Click the **Steps** tab.
8. From **Add Step**, select **Gradle**.
9. To call the Gradle installation available on the build executor, select **Use 'gradle' executable**. To use Gradle wrapper, select **Use 'gradlew' wrapper**.

   If you selected **Use 'gradlew' wrapper**, deselect the **Create 'gradlew' wrapper** check box if you don't want to create a new Gradle wrapper when a build runs. If the check box isn't selected, make sure that the `gradlew` executable is in the `$WORKSPACE` directory. If the `gradlew` executable is in the root build script directory, select the **In root build script directory** check box.

   To use another version of Gradle, specify the version in **Gradle version**.

   > **Tip:**
   >
   > To change the Gradle version when a build runs, add a build parameter and use it here. When a build runs, user can change the default version of Gradle and specify another version.
10. In **Tasks**, enter Gradle tasks.
11. In **Build File**, enter the name and path of the Gradle `build.gradle` file. This path must be relative to the root build script directory, if specified, else relative to the `$WORKSPACE` directory.
12. In **Root build script directory**, enter the directory path that contains the top-level `build.gradle` file and serves as the project root. The path must be relative to the `$WORKSPACE` directory.

    If left empty, the path defaults to `build.gradle` in the root directory.
13. In **Switches**, enter Gradle switches.
14. If you're using a multi-executor slave, select the **Force GRADLE_USER_HOME to use workspace** check box to set the `GRADLE_USER_HOME` to the workspace and avoid collisions while accessing Gradle cache.

    By default, `GRADLE_USER_HOME` is set to `$HOME/.gradle`.
15. Click **Save**.

Build Node.js Applications

Using Node.js, you can develop applications that run JavaScript on a server. For more information, see `https://nodejs.org`.

Set Up a Build VM and a Build VM Template with Node.js

Before you can use Node.js, you must create a Build VM template that includes Node.js and then add a Build VM that uses the VM template. If you don't want to create a template, you can add the software to an existing template. After adding a Build VM, create and configure a job to use the Build VM template and add a Node.js script.

 You must be the **Organization Administrator** to create a Build VM template and add a Build VM.

1. In the navigation bar, click **Organization**  .
2. Click the **Virtual Machines Templates** tab.
3. Click **+ Create Template**. Enter name and description, select the platform, and click **Create**.

   To use an existing template, click its name.
4. In the Software Catalog, search for `Node.js`. To see the latest version only, select the **Show latest versions only** check box.
5. Select the software tile and click **Done**.
6. Click **Build Virtual Machines** tab.
7. Click **+ Create VM**.
8. In the Add Build VM dialog box, in **Quantity**, specify the number of VMs you want to allocate. In **VM Template**, select the Node.js template.
9. Click **Add**.

Configure a Job to Build a Node.js Application

1. If you have a Node.js script, upload it to the project Git repository.
2. Open the job's configuration page.

   If you're creating a job, in **Software Template** of the New Job dialog box, select the Node.js template. Jump to step 5.
3. Click **Settings**  .
4. In the **Software** tab, select the Node.js template.
5. Click **Configure**  .
6. In the **Git** tab, add the Git repository where you uploaded the script file.
7. Click the **Steps** tab.
8. From **Add Step**, select **Node.js**.

9. To specify the script file, in **Source**, select **NodeJS File**. In **NodeJS File Path**, specify the file path in the Git repository.

   To specify the script, in **Source**, select **Script**. In **NodeJS Script**, enter the script.

10. Click **Save**.

Access an Oracle Database Using SQLcl

Using SQLcl, you can run SQL statements from a build to connect and access an Oracle Database. You can use SQLcl to access any Oracle Database available on public internet that you can connect to using a JDBC connect string. You can run DML, DDL, and SQL Plus statements. You can also use SQLcl in a test scenario and run SQL scripts to initialize seed data or validate database changes.

SQLcl requires Java SE 1.8 or later. To learn more about SQLcl, see `http://www.oracle.com/technetwork/developer-tools/sqlcl/overview/index.html`. Also see Using the help command in SQLcl in *Using Oracle Database Exadata Express Cloud Service* and the *SQL Developer Command-Line Quick Reference* documentation to know more about using SQLcl supported commands.

To connect to Oracle Database Exadata Express Cloud Service, download the ZIP file that contains its credentials and upload it to the job's Git repository. You can download the ZIP file from the Oracle Database Cloud Service service console. See Downloading Client Credentials in *Using Oracle Database Exadata Express Cloud Service*.

Set Up a Build VM and a Build VM Template with SQLcl

Before you can use SQLcl, you must create a Build VM template that includes SQLcland then add a Build VM that uses the VM template. If you don't want to create a template, you can add the software to an existing template. After adding a Build VM, create and configure a job to use the Build VM template and add a SQLcl commands.

 You must be the **Organization Administrator** to create a Build VM template and add a Build VM.

1. In the navigation bar, click **Organization**  .
2. Click the **Virtual Machines Templates** tab.
3. Click **+ Create Template**. Enter name and description, select the platform, and click **Create**.

   To use an existing template, click its name.
4. In the Software Catalog, search for `SQLcl`. To see the latest version only, select the **Show latest versions only** check box.
5. Select the software tile and click **Done**.
6. Click **Build Virtual Machines** tab.
7. Click **+ Create VM**.
8. In the Add Build VM dialog box, in **Quantity**, specify the number of VMs you want to allocate. In **VM Template**, select the SQLcl template.
9. Click **Add**.

Configure a Job to Run SQLcl Commands

Before you configure the job, note these points:

- DevCS doesn't support SQL commands to edit buffer (such as `set sqlformat csv`) or edit console.
- DevCS doesn't support build parameters in the SQL file.
- If you are using Oracle REST Data Services (ORDS), some SQLcl commands, such as the BRIDGE command, requires a JDBC URL.
  Example: `BRIDGE table1 as "jdbc:oracle:thin:DEMO/demo@http://examplehost.com/ords/demo"(select * from DUAL);`
- To mark a build as failed if the SQL commands fail, add the `WHENEVER SQLERROR EXIT 1` line to your script.

1. Open the job's configuration page.

   If you're creating a job, in **Software Template** of the New Job dialog box, select the SQLcl template. Jump to step 5.

2. Click **Settings**  .
3. In the **Software** tab, select the SQLcl template.
4. From the **Java** drop-down list, select version `1.8.x`, or later.
5. Click **Configure**  .
6. In the **Git** tab, add the Git repository where you uploaded the script file.
7. Click the **Steps** tab.
8. From **Add Step**, select **SQLcl**.
9. In **Username** and **Password**, enter the user name and password of the Oracle Database account.

   You can also use build parameters in **Username** and **Password**.

10. To connect to Oracle Database Exadata Express Cloud Service, in **Credentials File**, enter the workspace path of the uploaded credentials zip file.
11. In **Connect String**, enter the JDBC or HTTP connection string of the Oracle Database account using any of the `host_name:port:SID` or `host_name:port/service_name` formats.

    JDBC Example: `test_server.oracle.com:1521:adt1100` where `adt1100` is the SID, and `test_server.oracle.com:1521/ora11g` where `ora11g` is the service name.

    HTTP Example: `http://test_server.oracle.com:8085/ords/demo`

    You can also use build parameters in **Connect String**.

12. If the SQL statements are available in a file uploaded to the project Git repository, in **Source**, select **SQL File**. In **SQL File Path**, enter the Git repository path of the SQL file. You can copy the file's path from the **Git** page.

    To enter SQL statements, in **Source**, select **Inline SQL**. In **SQL Statements**, enter the SQL statements. You can also use build parameters in **SQL Statements**.

13. In **Role**, if necessary, select the database role of the user.
14. In **Restriction Level**, if necessary, specify the restriction level on the type of SQL statements that are allowed to run.
15. Click **Save**.

When a build runs, DevCS stores your Oracle Database credentials in the Oracle Wallet. Check the build's log for the SQL output or errors.

Run Oracle PaaS Service Manager Commands Using PSMcli

Using Oracle PaaS Service Manager command line interface (PSMcli) commands, you can create and manage the lifecycle of various services in Oracle Public Cloud. You can create service instances, start or stop instances, or remove instances when a build runs.

For more information about PSMcli and its commands, see About the PaaS Service Manager Command Line Interface in *PaaS Service Manager Command Line Interface Reference*.

Set Up a Build VM and a Build VM Template with PSMcli

Before you can use PSMcli, you must create a Build VM template that includes PSMcli and then add a Build VM that uses the VM template. If you don't want to create a template, you can add the software to an existing template. After adding a Build VM, create and configure a job to use the Build VM template and add PSMcli commands.

 You must be the **Organization Administrator** to create a Build VM template and add a Build VM.

1. In the navigation bar, click **Organization**  .
2. Click the **Virtual Machines Templates** tab.
3. Click **+ Create Template**. Enter name and description, select the platform, and click **Create**.

   To use an existing template, click its name.
4. In the Software Catalog, search for `PSMcli`. To see the latest version only, select the **Show latest versions only** check box.
5. Select the software tile and click **Done**.
6. Click **Build Virtual Machines** tab.
7. Click **+ Create VM**.
8. In the Add Build VM dialog box, in **Quantity**, specify the number of VMs you want to allocate. In **VM Template**, select the PSMcli template.
9. Click **Add**.

Configure a Job to Run PSMcli Commands

1. Open the job's configuration page.

   If you're creating a job, in **Software Template** of the New Job dialog box, select the PSMcli template. Jump to step 5.

2. Click **Settings**  .
3. In the **Software** tab, select the PSMcli template.
4. Click **Configure**  .
5. In the **Git** tab, add the Git repository where you uploaded the script file.
6. Click the **Steps** tab.
7. From **Add Step**, select **PSMcli**.
8. In **Username** and **Password**, enter the user name and password of the Oracle Cloud account.
9. In **Identity Domain**, enter the identity domain.
10. In **Region**, select your identity domain's region.
11. In **Output Format**, select the preferred output format: **JSON** (default) or **HTML**.
12. Scroll up and from **Steps**, select **Unix Shell**.
13. In **Script**, enter the PSM commands on separate lines.
14. Click **Save**.

You can add multiple shell steps to run different group of commands. Don't add the PSMcli build step again.

Access Oracle Cloud Infrastructure Services Using OCIcli

Using Oracle Cloud Infrastructure command line interface (OCIcli) commands, you can create and manage Oracle Cloud Infrastructure objects and services when a build runs.

For more information about OCIcli and its commands, see the Oracle Cloud Infrastructure Command Line Interface documentation.
To configure the job, you need the User OCID, private key, and fingerprint of a user who can create and access the resources, and the tenancy name. Contact the OCI administrator and get the required OCI input values. See Get the Required OCI Input Values to learn where to find the input values.

Set Up a Build VM and a Build VM Template with OCIcli

Before you can use OCIcli, you must create a Build VM template that includes OCIcli and then add a Build VM that uses the VM template. If you don't want to create a template, you can add the software to an existing template. After adding a Build VM, create and configure a job to use the Build VM template and add OCIcli commands.

 You must be the **Organization Administrator** to create a Build VM template and add a Build VM.

1. In the navigation bar, click **Organization**  .
2. Click the **Virtual Machines Templates** tab.
3. Click **+ Create Template**. Enter name and description, select the platform, and click **Create**.

   To use an existing template, click its name.

4. In the Software Catalog, search for `OCIcli`. To see the latest version only, select the **Show latest versions only** check box.
5. Select the software tile and click **Done**.
6. Click **Build Virtual Machines** tab.
7. Click **+ Create VM**.
8. In the Add Build VM dialog box, in **Quantity**, specify the number of VMs you want to allocate. In **VM Template**, select the OCIcli template.
9. Click **Add**.

Configure a Job to Run OCIcli Commands

1. Open the job's configuration page.

   If you're creating a job, in **Software Template** of the New Job dialog box, select the PSMcli template. After creating the job, jump to step 4.

2. Click **Settings**  .
3. In the **Software** tab, select the OCIcli template.
4. Click **Configure**  .
5. Click the **Steps** tab.
6. From **Add Step**, select **OCIcli**.
7. In **User OCID**, enter the OCID of the user who can access or create OCI resources.
8. In **Fingerprint**, enter the public key fingerprint of the user.
9. In **Tenancy**, enter the tenancy OCID.
10. In **Private Key**, enter the private key of the user.
11. In **Region**, select the Oracle Cloud Infrastructure tenancy's region.
12. Scroll up and from **Add Step**, select **Unix Shell**.
13. In **Script**, enter the OCIcli commands on separate lines.
14. Click **Save**.

You can add multiple **Unix Shell** steps to run different group of commands. Don't add the **OCIcli** build step again.

Run Docker Commands

You can configure a job to run Docker commands on a Docker container when a build runs.

You should use the Docker container for short tests and builds. Don't run a Docker container for long tests or builds, else the builds might not finish. For example, if you use a Docker image that's listening on a certain port and behaves as a web server, the build won't exit.

For more information about Docker commands, see `https://docs.docker.com/`.

> **Tip:**
>
> If you face a network issue while running a Docker command, add the `HTTP_PROXY` and `HTTPS_PROXY` environment variables in the Docker file.

Set Up a Build VM and a Build VM Template with Docker

Before you can use Docker, you must create a Build VM template that includes Docker and then add a Build VM that uses the VM template. If you don't want to create a template, you can add the software to an existing template. After adding a Build VM, create and configure a job to use the Build VM template and add Docker commands.

 You must be the **Organization Administrator** to create a Build VM template and add a Build VM.

1. In the navigation bar, click **Organization**  .
2. Click the **Virtual Machines Templates** tab.
3. Click **+ Create Template**. Enter name and description, select the platform, and click **Create**.

   To use an existing template, click its name.
4. In the Software Catalog, search for `Docker`. To see the latest version only, select the **Show latest versions only** check box.
5. Select the software tile and click **Done**.
6. Click **Build Virtual Machines** tab.
7. Click **+ Create VM**.
8. In the Add Build VM dialog box, in **Quantity**, specify the number of VMs you want to allocate. In **VM Template**, select the Docker template.
9. Click **Add**.

Configure a Job to Run Docker Commands

1. Open the job's configuration page.

   If you're creating a job, in **Software Template** of the New Job dialog box, select the Docker template. Jump to step 4.
2. Click **Settings**  .
3. In the **Software** tab, select the Docker template.
4. Click **Configure**  .
5. Click the **Steps** tab.
6. From **Add Step**, select **Docker**, and then select the Docker command.

Use this command ...	To ...
`login`	Log in to the Docker registry. In **Registry Host**, select a pre-linked Docker registry, or enter the Docker registry's host name where the images are stored. Leave it empty to use Docker Hub. In **Username** and **Password**, enter the credentials of the user who can access the Docker registry.
`build`	Build Docker images from a Dockerfile. Specify the registry host name, Docker image name, its version tag, Docker options, and name and the source of the Dockerfile. You can upload the Dockerfile in the Git repository and provide its path, add the Dockerfile code manually, or provide its URL if it's available on an external source. To specify an external source, include the protocol (example, `http`) in the URL if you're referencing a remote tar file (example: `http://55.555.555.555/me/mydocker.tar.gz`). Ignore the protocol if you're referencing a remote repository (example: `git://github.com/me/my.git#mybranch:myfolder`). To learn more about Docker build command options, see `https://docs.docker.com/engine/reference/commandline/build/`.
`tag`	Create a target image tag that refers to the source image. Specify the registry host name, Docker image name, and its version tag name for the source and target images.
`push`	Push an image to the Docker registry. To learn more about push options, see `https://docs.docker.com/engine/reference/commandline/push/`.
`images`	List available images. To learn more about images, see `https://docs.docker.com/engine/reference/commandline/images/`.
`save`	Save an image to a `.tar` archive file. In **Output File**, specify the relative path and name of the output `.tar` file in the workspace.
`load`	Load an image from a `.tar` archive file. In **Output File**, specify the relative path and name of the output `.tar` file in the workspace.
`rmi`	Remove an image. You can remove new images, a specific image, or all images. To remove a specific image, enter the host name of the registry where the Docker images are stored. Remember that the images are stored in the registry if they are pushed there. Until the images are pushed, the Registry Host is used to form the fully qualified name of the Docker image on the computer where the image is being created.
`version`	View the version of Docker on the build executor.

7. Click **Save**.

The Docker `logout` command runs automatically after all Docker commands have run.

Trigger a Wercker Pipeline

Wercker is a Docker-based platform that provides continuous integration and continuous delivery (CI/CD) capabilities.

To learn more about Wercker, see `https://devcenter.wercker.com/`.

Get Wercker Authentication Token

To trigger the Wercker pipeline from a job's build, you need your Wercker account's token.

1. In a web browser, open `https://app.wercker.com/sessions/new` and log in using your Wercker or GitHub account.
2. In the top-right corner of the page, click the user icon and select **Settings**.
3. In the left navigation bar, click **Personal Tokens**.
4. In **Token Name**, enter a name and click **Generate**.

   This is required if you create a new build.
5. Copy the generated token value and keep it safe.

   It is important that you save the generated token because you can't view or copy the token later.

Configure a Job to Trigger the Wercker Pipeline

1. Open the job's configuration page.
2. Click **Configure**  .
3. Click the **Steps** tab.
4. From **Add Step**, select **Wercker**.
5. In **Token**, paste the copied Wercker's authentication token.
6. Verify the values of **Application**, **Pipeline**, and **Branch**.

   The fields are automatically populated.
7. In **Message**, enter a message to be passed to Wercker. When a build runs, the message is displayed in the Runs tab of Wercker.
8. Click **Save**.

When a build of the job runs, it triggers the specified Wercker pipeline.

Run Fn Commands

Fn, or Fn Project, is an open-source, container-native, serverless platform for building, deploying, and scaling functions in multi-cloud environments. To run Fn commands when a build runs, you must have access to a Docker container that has a running Fn server.

For more information about Fn, see `https://fnproject.io/`.

Set Up a Build VM and a Build VM Template with Fn

Before you can use Fn, you must create a Build VM template that includes Fn and then add a Build VM that uses the VM template. If you don't want to create a template, you can add the software to an existing template. After adding a Build VM, create and configure a job to use the Build VM template and add Fn commands.

 You must be the **Organization Administrator** to create a Build VM template and add a Build VM.

1. In the navigation bar, click **Organization**  .
2. Click the **Virtual Machines Templates** tab.
3. Click **+ Create Template**. Enter name and description, select the platform, and click **Create**.

   To use an existing template, click its name.
4. In the Software Catalog, search for `Fn`. To see the latest version only, select the **Show latest versions only** check box.
5. Select the software tile and click **Done**.
6. Click **Build Virtual Machines** tab.
7. Click **+ Create VM**.
8. In the Add Build VM dialog box, in **Quantity**, specify the number of VMs you want to allocate. In **VM Template**, select the Fn template.
9. Click **Add**.

Configure a Job to Run Fn Commands

1. Open the job's configuration page.

   If you're creating a job, in **Software Template** of the New Job dialog box, select the Fn template. Jump to step 4.
2. Click **Settings**  .
3. In the **Software** tab, select the Fn template.
4. Click **Configure**  .
5. Click the **Steps** tab.
6. From **Add Step**, select **Fn**, and then select the command.

Use this option ...	To ...
`Fn Build`	Build a new function. Specify the relative path of the working directory to build the function, Fn build arguments, Docker registry host, and its user name. If you don't want to use Docker registry's cache, deselect the **Use Docker Cache** check box. To display the command's log in the build's log, select the **Verbose Output** check box.

Use this option ...	To ...
`Fn Push`	Push the image to the Docker registry. Specify the relative path of the working directory, Docker registry host, and its user name. To display the command's log in the build's log, select the **Verbose Output** check box.
`Fn Bump`	Bump the version of the `func.yaml` file. Specify the relative path of the working directory and the bump type (`Major`, `Minor`, or `Patch`). To display the command's log in the build's log, select the **Verbose Output** check box.
`Fn Test`	The `Fn Test` command isn't supported. It'll be removed from the list of available Fn commands in an upcoming release.
`Fn Deploy`	Deploy functions to the function server. Using the deploy command, you can bump, build, push and update a function. In **Deploy to App**, specify the Fn app name to deploy to. In other fields, specify the working directory, build arguments, Docker registry host, user name, API URL, and the Call URL. Select the desired check boxes, if necessary.

7. Click **Save**.

Use SonarQube

SonarQube is an open source quality management software that enables you to continuously analyze your application. When you configure a job to use SonarQube, the build generates an analysis summary that you can view from the job or the build details page.

To learn about SonarQube, see its documentation at `https://docs.sonarqube.org`.

Set Up SonarQube

To set up a SonarQube system for your project users, you can create a pre-defined SonarQube connection for your users.

To create the connection, you need the URL of a SonarQube Server available on the public internet.

You must be assigned the project **Owner** role to add and manage SonarQube connections.

Action	How To
Add a SonarQube connection	1. In the navigation bar, click **Project Administration** . 2. Click **Build**. 3. Click the **SonarQube Server** tab. 4. Click **Add SonarQube Server Connection**. 5. In the Create SonarQube Server dialog box, enter a name for the server, provide the SonarQube server's URL, and the credentials of a user who has access to the server. 6. Click **Create**.

Action	How To
Edit a connection	To change the connection's user credentials or provide another server ID, you can edit the connection. 1. In the navigation bar, click **Project Administration**  . 2. Click **Build**. 3. Click the **SonarQube Server** tab 4. Click the connection name and then click the Edit icon. 5. In the Edit SonarQube Server dialog box, as necessary, update the SonarQube server's URL and the credentials of a user who can access the server. 6. Click **Update**.
Delete the connection	1. In the navigation bar, click **Project Administration**  . 2. Click **Build**. 3. Click the **SonarQube Server** tab 4. Click the connection name and then click  . 5. In the Delete SonarQube Server dialog, click **Delete**.

Configure a Job to Connect to SonarQube

You can configure a job to use SonarQube from the Before Build tab and add a post-build action to publish its reports.

1. Open the job's configuration page.
2. Click the **Before Build** tab.
3. From **Add Before Build Action**, select **SonarQube Settings**.
4. From **Sonar Server**, select the pre-configured SonarQube server.

   The **Username**, **Password**, and **SonarQube Server URL** display the details of the selected user. To add a server, contact the organization administrator.
5. To provide the SonarQube project name and the SonarQube project key, expand **Advanced SonarQube Settings**, and update the values. Make sure that the SonarQube project key is unique.

   By default, the project key is set to `<organization>_<projectname>.<jobname>`and the project name is set to `<projectname>.<jobname>`.
6. Click the **After Build** tab.
7. From **Add After Build Action**, select **SonarQube Result Publisher**.
8. To use the SonarQube Quality Gate status as the build status, select the **Apply SonarQube quality gate status as build status** check box.

   If the SonarQube Quality Gate status is `Passed`, the build is marked as successful. If the SonarQube Quality Gate status is `Failed`, the build is marked as failed. To learn about SonarQube Quality Gates, see `https://docs.sonarqube.org/display/SONAR/Quality+Gates`.

9. To create an archive file of the SonarQube analysis files, select the **Archive Analysis Files** check box.
10. Click **Save**.

To view the SonarQube analysis summary after a build, from the job's details page, click **SonarQube Analysis Summary**  . The SonarQube Analysis Summary displays SonarQube server URL of the job and the analysis summary.

Use Named Passwords

A named password is a variable that users can use across a project's build job configurations. Named passwords can be used in any password field in the job configuration, such as external Git repositories, SQLcl, PSMcli, and Docker configurations.

When the password changes, change the value of the variable and the new password is applied to all jobs and configurations where the variable is used. Note that the named password is not an environment variable. To use a named password as an environment variable, create a **Password** build parameter and set it to use the named password.

Create a Named Password

 You must be assigned the project **Owner** role to create a named password.

Action	How To
Create a named password	1. In the navigation bar, click **Project Settings**  . 2. Click **Build**. 3. Click the **Named Passwords** tab. 4. Click **+ Create**. 5. In the Create New Named Password dialog box, in **Name**, enter a name for the variable. In **Password**, enter the password. 6. Click **Create**. After creating the named password, share its name with your project users.
Edit a named password	1. In the navigation bar, click **Project Settings**  . 2. Click **Build**. 3. Click the **Named Passwords** tab 4. Click the password name and then click  . 5. In the Edit Named Password dialog box, update the password. You can't change the named password's name. 6. Click **Update**.

Action	How To
Delete the named password	1. In the navigation bar, click **Project Settings**  . 2. Click **Build**. 3. Click the **Named Passwords** tab 4. Click the named password name and then click  . 5. In the Delete Named Password dialog box, click **Delete**. After deleting the named password, let your project users know that it's no longer available.

Configure a Job to Use a Named Password

1. Open the job's configuration page.
2. In the Password field of the component you want to configure, enter the named variable as `#{password_name}`.

   For example, if the name of the named password is `my_password`, enter `#{my_password}`.
3. Click **Save**.

Publish JUnit Results

If you use JUnit in your application to run test scripts, you can configure your job to publish JUnit test reports and get useful information about test results, such as historical test result trends, failure tracking, and so on.

1. Upload your application with test script files to the Git repository.
2. Open the job's configuration page.
3. Click the **After Build** tab.
4. From **Add After Build Action**, select **JUnit Publisher**.
5. In **Include JUnit XMLs**, specify the path and names of XML files to include. You can also use wildcards.

   For example, if you're using Ant, specify the path as `**/build/test-reports/*.xml`. If you're using Maven, specify the path as `target/surefire-reports/*.xml`. Make sure that you don't include any non-report files into this pattern.
6. In **Exclude JUnit XMLs**, specify the path and names of XML report files to exclude. You can also use wildcards.
7. To see and retain the standard output and errors in the build log, select the **Retain long standard output/error** check box.

   If you don't select the check box, the build log is saved, but the build executor truncates it to save space. If you select the check box, every log message is saved, but this might increase memory consumption and can slow the performance of the build executor.
8. To combine all test results into a single table of results, select the **Organize test output by parent location** check box.

If you use multiple browsers, then the build executor categorizes results by browsers.

9. To mark the build as failed if the JUnit tests fail, select the **Fail the build on fail tests** check box.
10. To archive videos and image files, select the **Archive Media Files** check box.
11. Click **Save.**

After a build runs, you can view its test result.

View Test Results

You can view the JUnit test results of a build from the Test Results page.

Action	How To
View test results of the last build	1. Open the job's details page. 2. Click **Tests**  .
View test results of a particular build	1. Open the job's details page. 2. In the Build History table, click the build number. 3. Click **Tests**  .
View test suite details	On the Test Results page, click the **All Tests** toggle button. From the **Suite Name**, click the suite name.
View details of a test	Open the test suite details page and click the test name. To view details of a failed test, on the Test Results page, click the **All Failed Tests** toggle button, and then click the test name.
View test results history	On the Test Results page, click **View Test Results History**.

If you configure the job to archive videos and image files, click **Show**  to download the test image and click **Watch**  to download the test video file.

The supported image formats are `.png`, `.jpg`, `.gif`, `.tif`, `.tiff`, `.bmp`, `.ai`, `.psd`, `.svg`, `.img`, `.jpeg`, `.ico`, `.eps`, and `.ps`.

The supported video formats are `.mp4`, `.mov`, `.avi`, `.webm`, `.flv`, `.mpg`, `.gif`, `.wmv`, `.rm`, `.asf`, `.swf`, `.avchd`, and `.m4v`.

Use the Xvfb Wrapper

Xvfb is an X server that implements the X11 display server protocol and can run on machines with no physical input devices or display.

Set Up a Build VM and a Build VM Template with Xvfb

Before you can use Xvfb, you must create a Build VM template with minimum required software and then add a Build VM that uses the VM template. You can use any

existing Oracle Linux 7 VM template to run Xvfb, or create a VM template if you don't want to use an existing VM template. Xvfb isn't available on an Oracle Linux 6 VM template.

 You must be the **Organization Administrator** to create a Build VM template and add a Build VM.

1. In the navigation bar, click **Organization**  .
2. Click the **Virtual Machines Templates** tab.
3. Click **+ Create Template**. Enter name and description, select **Oracle Linux 7** as the platform, and click **Create**.

   To use an existing template, click its name.
4. Click **Build Virtual Machines** tab.
5. Click **+ Create VM**.
6. In the Add Build VM dialog box, in **Quantity**, specify the number of VMs you want to allocate. In **VM Template**, select the Xvfb template.
7. Click **Add**.

Configure a Job to Run Xvfb

1. Open the job's configuration page.

   If you're creating a job, in **Software Template** of the New Job dialog box, select the Xvfb template. Jump to step 4.
2. Click **Settings**  .
3. In the **Software** tab, select the Xvfb template or any minimum required software template.
4. Click **Configure**  .
5. Click the **Before Build** tab.
6. From **Add Before Build Action**, select **Xvfb Wrapper**.
7. In **Display Number**, specify the ordinal number of the display the Xvfb server is running on. The default value is 0. If left empty, a random number is chosen when the build runs.
8. In **Screen offset**, specify the offset for display numbers. The default value is 0.
9. In **Screen Size (WxHxD)**, specify the resolution and color depth of the virtual frame buffer in the WxHxD format. The default value is `1024x758x24`.
10. In **Additional options**, specify additional Xvfb command line options, if necessary. The default options are `-nolisten inet6 +extension RANDR -fp /usr/share/X11/fonts/misc`.
11. In **Timeout in seconds**, specify the timeout duration for the build to wait before returning control to the job. The default value is 0.
12. If you don't want to log the Xvfb output in the build log, deselect the **Log Xvfb output** check box. The check box is selected by default.

13. If you don't want to keep the Xvfb server running for post-build steps, deselect the **Shutdown Xvfb with whole job, not just with the main build action** check box. The check box is selected by default.
14. Click **Save**.

Publish Javadoc

If your application source code files are configured to generate Javadoc, then you can configure a job to publish Javadocs when a build runs.

1. Open the job's configuration page.
2. Click the **After Build** tab.
3. From **Add After Build Action**, select **Javadoc Publisher**.
4. In **Javadoc Directory**, specify the workspace path where the build executor would publish the generated Javadoc. By default, the path is set to `target/site/apidocs` .
5. To configure the build executor to retain Javadoc for each successful build, select the **Retain Javadoc for each build** check box.

   You may want to do this to browse Javadoc of older builds, but remember that retaining old Javadoc would consume more disk space. By default, the check box isn't selected.
6. Click **Save**.

Archive Artifacts

If you want builds of a job to archive artifacts, you can do so as a after build action. Once archived, you can download the artifacts manually and deploy them. By default, artifacts of a build are kept as long as the build log is kept.

1. Open the job's configuration page.
2. Click **Configure**  .
3. Click the **After Build** tab.
4. Click **Add After Build Action** and select **Artifact Archiver**.
5. In **Files to archive**, enter comma separated list of files (with path) to archive. You can use wildcards too. Example: `module/dist/**/*.zip`.
6. In **Files to exclude**, enter comma separated list of files (with path) to exclude.

   A file that matches the exclude pattern isn't archived even if it matches the pattern specified in **Files to archive**.
7. If your application is a Maven application and you want to archive Maven artifacts, select the **Archive Maven Artifacts** check box.

   To archive the Maven POM file along with Maven artifacts, select the **Include POM.xml** check box.
8. Click **Save**.

Discard Old Builds and Artifacts

To save storage space, you can configure a job to discard its old builds and artifacts. The old builds are discarded after the job configuration is saved and after a job is built.

1. Open the job's configuration page.
2. Click **Settings**  .
3. Click the **General** tab, if necessary.
4. If not selected, select the **Discard Old Builds** check box.
5. Configure the discard options.
6. Click **Save**.

Copy Artifacts from Another Job

If your application depends on another job's artifacts, you can configure the job to copy those artifacts when a build runs.

1. Open the job's configuration page.
2. Click **Configure**  .
3. Click the **Before Build** tab.
4. Click **Add Before Build Action** and select **Copy Artifacts**.
5. In **From Job**, select the job whose artifacts you want to copy.
6. In **Which Build**, select the build that generated the artifacts.
7. In **Artifacts to copy**, specify the artifacts to copy. When a build runs, the artifacts are copied with their relative paths.

   If you specify no value, the build copies all artifacts. The `archive.zip` file is never copied.
8. In **Target Directory**, specify the workspace directory where the artifacts will be copied.
9. To flatten the directory structure of the copied artifacts, select the **Flatten Directories** check box.
10. By default, if a build can't copy artifacts, it's marked as failed. If you don't want the build to be marked as failed, select the **Optional (Do not fail build if artifacts copy failed)** check box.
11. Click **Save**.

Deploy Build Artifacts to Oracle Cloud Services

The recommended way to deploy your project's build artifacts to Oracle Cloud Services, including Oracle Java Cloud Service (JCS) and Oracle Application Container Cloud Service (ACCS), is to do so in an Oracle Deploy build step. To deploy build artifacts to Oracle Java Cloud Service - SaaS Extension (JCS-SX) from Oracle Developer Cloud Service (DevCS), you need to create and use a deployment configuration from the Deployments page.

Before. you create a build step for deployment, you need to create an environment and add the JCS and ACCS instances to it. These instances will be used for your deployment targets. If you do not add them in the Environments page, you will not be able to select them in the build step. See Set Up an Environment for information about creating an environment and adding instances to it.

You can either add a build step that deploys the build artifact to the job that creates and packages the artifact or create a separate job for each task. If you choose to use separate jobs, you can create a pipeline that begins with a job that builds and packages the application, followed by a job that deploys the build artifact to the desired target environment. Using pipelines gives you the flexibility of adding testing and other tasks to the flow.

Create a Build Step to Deploy an Application to JCS

You can create a separate job that copies build artifacts generated by another job and deploys those artifacts to a JCS deployment target.

1. In the navigation bar, click **Builds**  .
2. In the **Jobs** tab, click **+ Create Job**.
3. In the New Job dialog box, in **Name**, enter a unique name.
4. In **Description**, enter the job's description.
5. In **Template**, select the Build VM template.
6. Click **Create**.
7. In the Job Configuration page, in the **Before Build** tab, select **Copy Artifacts** from the **Add Before Build Action** dropdown.
8. Select the job that produces the artifact from the **From job** dropdown and the Last successful build from the **Which build** dropdown, then click **Save**.
9. In the **Steps** tab, click the **Add Step** dropdown and select **Oracle Deployment**.
10. In the **Application Name** field, enter the name that will be used by the target service to identify your application.
11. In the **Deployment Target** dropdown, select the environment where you want to deploy the application.

    If you need to create a new connection, define a new instance in an environment and it will show up as a target in the dropdown.
12. In the **Deploy to Java** dialog, specify the desired version and protocol, then enter the HTTPS port number, the username, and the password.
13. Click **Find Targets** and select the server you want to deploy to from the list of available servers or clusters.
14. Click **OK**.
15. In the **Artifact** field, enter the path to the artifact that you want to deploy.
16. Click **Save**.
17. To run the job, from the **Builds** page, click **Build Now** and the job will execute, first copying the artifact over, then deploying it to the selected deployment target.

At this point, you probably want to create a pipeline that flows a series of jobs that builds and packages the artifact, then deploys it to the desired deployment target.

Create a Build Step to Deploy an Application to ACCS

You can create a separate job that copies build artifacts generated by another job and deploys those artifacts to an ACCS deployment target.

1. In the navigation bar, click **Builds**  .
2. In the **Jobs** tab, click **+ Create Job**.
3. In the New Job dialog box, in **Name**, enter a unique name.
4. In **Description**, enter the job's description.
5. In **Template**, select the Build VM template.
6. Click **Create**.
7. In the Job Configuration page, in the **Before Build** tab, select **Copy Artifacts** from the **Add Before Build Action** dropdown.
8. Select the job that produces the artifact from the **From job** dropdown and the Last successful build from the **Which build** dropdown, then click **Save**.
9. In the **Steps** tab, click the **Add Step** dropdown and select **Oracle Deployment**.
10. In the **Application Name** field, enter the name that will be used by the target service to identify your application.
11. In the **Deployment Target** dropdown, select the environment where you want to deploy the application.

    If you need to create a new connection, define a new instance in an environment and it will show up as a target in the dropdown.
12. In the **ACCS Properties**:
    a. select the desired **Runtime** radio button (Java, Java EE, Node, or PHP).
    b. Select the **Subscription** radio button (Hourly or Monthly).
    c. To override the commands of the manifest file in the artifact zip, select the **Include ACCS Manifest** check box. For example, you can override the deployed application's version number at the time of deployment.

       In **ACCS Manifest**, enter the contents of the manifest file. The field is a code editor component and you can use code editor features.
    d. To enter the contents of the deployment descriptor, select the **Include ACCS Deployment** check box and enter the commands in **ACCS Deployment**.

       You can also enter commands to override the deployed application's container's configuration (such as RAM) at the time of deployment. The field is a code editor component and you can use code editor features.
       For more information about the ACCS metadata files, see Creating Metadata Files in *Developing for Oracle Application Container Cloud Service*.
13. Click **OK**.
14. In the **Artifact** field, enter the path to the artifact that you want to deploy.
15. Click **Save**.
16. To run the job, from the **Builds** page, click **Build Now** and the job will execute, first copying the artifact over, then deploying it to the selected deployment target.

At this point, you probably want to create a pipeline that flows a series of jobs that builds and packages the artifact, then deploys it to the desired deployment target.

Configure General and Advanced Job Settings

You can configure several general and advanced job settings, such as name and description, the JDK version used in the build, discarding old builds, running concurrent builds, adding timestamps to the build log, and more.

Action	How To
Update the job's name and description	1. Open the job's configuration page. 2. Click **Settings**  . 3. Click the **General** tab. 4. In **Name** and **Description**, update the job name and description. 5. Click **Save**.
Check the software available on the job's Build VM template	1. Open the job's configuration page. 2. Click **Settings**  . 3. Click the **Software** tab. 4. See the software and their versions. You can change versions of some software, such as Java SE. Select the version from the drop-down list. 5. Click **Save**.
Run concurrent builds	By default, DevCS runs one build of a job at a time. The next build runs after the running build finishes. To configure the job to run concurrent builds, do this: 1. Open the job's configuration page. 2. Click **Settings**  . 3. Click the **General** tab. 4. Select the **Execute concurrent builds if necessary** check box. 5. Click **Save**.
Set a quiet period	You can specify a period (in seconds) before a new scheduled build of the job should wait before it runs. If the build server is busy with too many builds, setting a longer quiet period can reduce the number of builds. 1. Open the job's configuration page. 2. Click **Settings**  . 3. Click the **Advanced** tab. 4. Select the **Quiet period** check box and specify the value in seconds.. 5. Click **Save**.

Action	How To
Set a retry count	If the builds of a job fail, by default, the build executor tries five times to run the build. You can increase or decrease the count. 1. Open the job's configuration page. 2. Click **Settings**  . 3. Click the **Advanced** tab. 4. Select the **Retry Count** check box. 5. In **Build Retries** specify the number of times the build executor tries the build. In **SCM Retries** specify the number of times the build executor tries the build to checkout files from the Git repository. 6. Click **Save**.
Abort a build if it's stuck for some duration	1. Open the job's configuration page. 2. Click **Settings**  . 3. Click the **Advanced** tab. 4. Select the **Abort the build if it is stuck** check box. 5. In **Hours** and **Minutes**, specify the duration. If a build doesn't complete in the specified amount of time, the build is terminated automatically and marked as aborted. Select the **Fail the build on abort** check box to mark the build as failed, rather than aborted. 6. Click **Save**.
Remove timestamps from the build log	By default, build logs are timestamped, however you can configure a job to remove them from the log. 1. Open the job's configuration page. 2. Click **Settings**  . 3. Click the **Advanced** tab. 4. Deselect the **Add Timestamps to the Console Output** check box. 5. Click **Save**.
Set the maximum size of the console log	1. Open the job's configuration page. 2. Click **Settings**  . 3. Click the **Advanced** tab. 4. In **Max Log Size (MB)**, set the size. The default value is 50 MB and the maximum value is 1000 MB. 5. Click **Save**.

Change a Job's JDK Version

1. Open the job's configuration page.
2. Click **Settings**  .
3. Click the **Software** tab.
4. In **Available Software**, from the **Java** drop-down list, change the JDK's version number.

   You can also select GraalVM instead of selecting JDK. GraalVM is a universal virtual machine for running applications written in JavaScript, Python, Ruby, R, JVM-based languages like Java, Scala, Groovy, Kotlin, Clojure, and LLVM-based languages such as C and C++. To learn more, see `https://www.graalvm.org/docs/`.
5. Click **Save**.

Change a Job's Build VM Template

You can change a job's Build VM template after you create the job.

Contact the Organization Administrator to create a Build VM template or install software to a template.

1. Open the job's configuration page.
2. Click **Settings**  .
3. Click the **Software** tab.
4. In **Software Template**, select the Build VM template that you want to use for your builds.
5. Click **Save**.

Run a Build

You can run a job's build manually or configure the job to trigger it automatically on an SCM commit or according to a schedule.

Action	How To
Run a build manually	Open the job's details page and click **Build Now**. You can also run a job's build from the Jobs Overview page. In the jobs table, click **Build Now**  .
Run a build on SCM commit	See Trigger a Build Automatically on SCM Commit.
Run a build on a schedule	See Trigger a Build Automatically on a Schedule.

View a Job's Builds and Reports

To view a job's builds, reports, build history and perform actions such as run a build or configure the job, on the Build page, click the job name to open its details page.

View a Build's Logs and Reports

A build generates various types of reports and logs such as SCM Changes, test results, and action history. You can open these reports from the Job Details page or the Build Details page.

The following table lists the various types of reports generated by a build. On the Job Details page or the Build Details page, click the report icon to view its details.

Log/Report	Description
Build Log 	View the last build's log. In the log page, review the build log. If the log is displayed partially, click the **Full Log** link to view the entire log. To download the log as a text file, click the **Download Console Output** link.
SCM Changes 	View all files that have changed in the build. When a build is triggered, the build system checks the job's Git repositories for any changes to the SCM. If there are any updates, the SCM Change log displays the files that were added, edited or removed.
Javadoc 	View Javadoc of the build. The report is available only if the application's build generated a Javadoc.
SCM Poll Log 	View the Git SCM polling log of the builds that displays the log of builds triggered by SCM polling. The log includes scheduled uilds and builds triggered by SCM updates. In the Job Details page of a job, click **Latest SCM Poll Log**  to view the Git SCM polling log of the last build.
Test Results 	View the log of build's JUnit test results. To open the Test Suite details page, on the Test Results page, click the **All Tests** toggle button and click the suite name in the **Suite Name** column. To view details of a test, on the Test Results page, click the **All Failed Tests** toggle button and then click the test name link in the **Test Name** column. You can also click the **All Tests** toggle button, open the test suite details page, and then click the test name link in the **Test Name** column.
Audit	View the Audit log of user actions. You can use the Audit log to track the user actions on a build. Use the log to see who performed particular actions on the job. For example, you can see who cancelled a build of the job, or who disabled the job and when was it disabled.
SonarQube Analysis Summary 	View the SonarQube analysis report of the job.

View a Project's Build History

The Recent Build History page displays builds of all jobs of the project.

To view the build history, in the **Build Queue** box of the **Builds** page, click the **View Recent Build History** link. The history page shows the last 50 builds of the project. Click a job name to open its details page. Click a build number to open its details page. Click to open the build's console and view the console log output.

> **Tip:**
>
> To sort the table data by a column, right-click inside the build history table column and select the sort order from the **Sort** context menu.

View a Job's Build History

A job's build history can be viewed in the Build history section of the Job Details page. It displays the status of the running builds, and completed job builds in descending order (recent first) along with their build numbers, date and time, and a link to the console output of the build.

The build history shows information abut how the build was triggered, its status, build number, date-time stamp, a **Console** icon to open the build's console, and a **Delete** icon to delete the build.

To review the build history, note these points:

- In the **By** column, the icons indicate the following:

This icon ...	Indicates:
User	The build was initiated by a user.
SCM Change	The build was triggered by an SCM change.
Pipeline	The build was initiated by a pipeline. Click to open the build's pipeline instance.
Periodic Build Trigger	The build was triggered by a periodic build trigger.
Build System	The build was started or rescheduled by the build system.

- In the **Build** column, an * in the build number indicates the build is annotated with a description. Mouse over the build number to see the description.
- The list doesn't show the discarded and deleted jobs.
- If a running build is stuck in the `Queued` status for a long time, mouse over the `Queued` status to see a message about the problem.

  If the build is using a Build VM, you may also contact the Organization Administrator to check the VM's status.
- To sort the table data in ascending or descending order, click the header column name and then click the Previous or Next icon in the column header.

You can also right-click inside table column and then select the sort order from the **Sort** context menu.

- A project non-member can't delete a build.

View a Job's User Action History

You can use the Audit log to track a job's user actions. For example, you can see who cancelled a build of the job, or who disabled the job and when it was disabled.

To open the Audit log, from the job's details page, click **Audit**  .
The log displays information about these user actions:

- Who created the job
- Who started a build or how a build was triggered (followed by the build number), when the build succeeded or failed, and the duration of the build

  A build can also be triggered by a timer, a commit to a Git repository, or an upstream job.
- Who aborted a build
- Who changed the configuration of the job
- Who disabled a job
- Who enabled a job

View a Build's Details

To open the a build's details page, click the build number in the Build History.

A build's details page shows its status, links to open build reports, download artifacts, and logs. You can perform these common actions from a build's details page.

Action	How To
Keep a build forever	A build that's marked as 'forever' isn't removed if a job is configured to discard old builds automatically. You can't delete it either. To keep a build forever, click **Configure**, select the **Keep Build Forever** check box, and click **Save**.
Add a name and description to a build	Adding a description and a name is especially helpful if you mark a particular build to keep it forever and not get discarded automatically. When you add a description to a build, an * is added to the build number in the Build History table. To keep a build forever, click **Configure**. In **Name** and **Description**, enter the details, and click **Save**.
Open a build's log	Click **Build Log**.
Delete a build	Click **Delete**.

Download Build Artifacts

If the job is configured to archive artifacts, you can download them to your computer and then deploy to your web server.

The build artifacts are displayed in a directory tree structure. You can click the link to download parts of the tree including individual files, directories, and subdirectories.

1. Open the job's details page.
2. Click **Artifacts**.

   To download artifacts of a particular build, in the Build History, click the build number, and then click **Artifacts**.
3. Expand the directory structure and click the artifact link (file or directory) to download it.

   To download a zip file of all artifacts, click **All files in a zip**.
4. Save the file to your computer.

Watch a Job

You can subscribe to email notifications when a build of a job is successful or fails.

To get email notifications, enable them in your user preferences, and then set up a watch on the job.

Action	How To
Enable your email notifications preference	In your user preferences page, select the **Build Activities** check box.
Watch a job	1. Open the job's details page. 2. Click the **On** toggle button, if necessary. 3. Click **CC Me**. 4. In the CC Me dialog box, to receive email when the build is successful, select the **Successful Builds** check box. Select **Failed Builds** to receive email when the build fails. 5. Click **OK**.
Disable email notifications of the job to all subscribed members	1. Open the job's details page. 2. Click the **Off** toggle button, if necessary.

Build Executor Environment Variables

When you run a build job, you can use the environment variables in your shell scripts and commands to access the software on the build executor.

To use a variable, use the `$VARIABLE_NAME` syntax. Example: `$BUILD_ID`.

Common Variables

This table describes some common environment variables.

Environment Variable	Description
BUILD_ID	The current build's ID.
BUILD_NUMBER	The current build number.
BUILD_URL	The full URL of the current build.
BUILD_DIR	The build output directory.
JOB_NAME	The name of the job.
EXECUTOR_NUMBER	The unique number that identifies the current executor (among executors of the same machine) that's running the current build.
HTTP_PROXY	The HTTP proxy for outgoing connections.
HTTP_PROXY_HOST	The HTTP proxy host for outgoing connections.
HTTP_PROXY_PORT	The HTTP proxy port for outgoing connections.
HTTPS_PROXY	The HTTPS proxy for outgoing connections.
HTTPS_PROXY_HOST	The HTTPS proxy host for outgoing connections.
HTTPS_PROXY_PORT	The HTTPS proxy port for outgoing connections.
JOB_NAME	The name of the current job.
JOB_URL	The full URL of the current job.
NO_PROXY	Specify the comma separated list of domain names or the IP addresses for which the proxy should not be used. You can also specify port numbers.
NO_PROXY_ALT	Specify the pipe (|) separated list of domain names or the IP addresses for which the proxy should not be used. You can also specify port numbers.
PATH	The PATH variable set in the build executor specifying the path of executables in the build executor.
WORKSPACE	The absolute path of the build executor's workspace.

Software Variables

Environment Variable	Description
DYNAMO_HOME	The path of the Oracle ATG home directory.
DYNAMO_ROOT	The path of the Oracle ATG root directory.
GRADLE_HOME	The path of the Gradle directory.
JAVA_HOME	The path of the directory where the Java Development Kit (JDK) or the Java Runtime Environment (JRE) is installed. If your job is configured to use a specific JDK, the build executor sets the variable to the path of the specified JDK. When the variable is set, PATH is also updated to have $JAVA_HOME/bin.
NODE_HOME	The path of the Node.js home directory.
NODE_PATH	The path of the Node.js modules directory.

To access JDeveloper or SOA, use these variables.

- Use `JAVACLOUD_HOME` variables to access the Java SDK
- Use `ORACLE_HOME` variables to access JDeveloper
- Use `MIDDLEWARE_HOME` variables to access Oracle Fusion Middleware. The `MIDDLEWARE_HOME` directory includes the JDeveloper installation directory, WebLogic Server installation directory, and the Oracle Common library dependencies.
- Use `WLS_HOME` variables to access the WebLogic server binary directory bundled with JDeveloper

Make sure that you have the right software available in the Build VM Template of your job.

Software	Variables

JDeveloper Studio 12	`JAVACLOUD_HOME=/opt/Oracle/Middleware_12.2.1.3.0/jdeveloper/cloud/oracle-javacloud-sdk/lib`
	`MIDDLEWARE_HOME=/opt/Oracle/Middleware_12.2.1.3.0`
	`ORACLE_HOME=/opt/Oracle/Middleware_12.2.1.3.0/jdeveloper`
	`WLS_HOME=/opt/Oracle/Middleware_12.2.1.3.0/wlserver`

JDeveloper 11*g*	`JAVACLOUD_HOME_11_1_1_7_1=/opt/Oracle/Middleware_11.1.1.7.1/jdeveloper/cloud/oracle-javacloud-sdk/lib`
	`JAVACLOUD_HOME_11G=/opt/Oracle/Middleware_11.1.1.7.1/jdeveloper/cloud/oracle-javacloud-sdk/lib`
	`JAVACLOUD_HOME=/opt/Oracle/Middleware_11.1.1.7.1/jdeveloper/cloud/oracle-javacloud-sdk/lib`
	`MIDDLEWARE_HOME_11_1_1_7_1=/opt/Oracle/Middleware_11.1.1.7.1`
	`MIDDLEWARE_HOME_11G=/opt/Oracle/Middleware_11.1.1.7.1`
	`MIDDLEWARE_HOME=/opt/Oracle/Middleware_11.1.1.7.1`
	`ORACLE_HOME_11_1_1_7_1=/opt/Oracle/Middleware_11.1.1.7.1/jdeveloper`
	`ORACLE_HOME_11G=/opt/Oracle/Middleware_11.1.1.7.1/jdeveloper`
	`ORACLE_HOME=/opt/Oracle/Middleware_11.1.1.7.1/jdeveloper`
	`WLS_HOME_11_1_1_7_1=/opt/Oracle/Middleware_11.1.1.7.1/wlserver_10.3`
	`WLS_HOME_11G=/opt/Oracle/Middleware_11.1.1.7.1/wlserver_10.3`
	`WLS_HOME=/opt/Oracle/Middleware_11.1.1.7.1/wlserver_10.3`

Software	Variables
SOA 12.2.1.3	JAVACLOUD_HOME_SOA_12_2_1=/opt/Oracle/MiddlewareSOA_12.2.1.3.0/jdeveloper/cloud/oracle-javacloud-sdk/lib JAVACLOUD_HOME_SOA=/opt/Oracle/MiddlewareSOA_12.2.1.3.0/jdeveloper/cloud/oracle-javacloud-sdk/lib MIDDLEWARE_HOME_SOA_12_2_1=/opt/Oracle/MiddlewareSOA_12.2.1.3.0 MIDDLEWARE_HOME_SOA=/opt/Oracle/MiddlewareSOA_12.2.1.3.0 ORACLE_HOME_SOA_12_2_1=/opt/Oracle/MiddlewareSOA_12.2.1.3.0/jdeveloper ORACLE_HOME_SOA=/opt/Oracle/MiddlewareSOA_12.2.1.3.0/jdeveloper WLS_HOME_SOA_12_2_1=/opt/Oracle/MiddlewareSOA_12.2.1.3.0/wlserver WLS_HOME_SOA=/opt/Oracle/MiddlewareSOA_12.2.1.3.0/wlserver
SOA 12.2.1.2	JAVACLOUD_HOME_SOA_12_2_1=/opt/Oracle/MiddlewareSOA_12.2.1.2/jdeveloper/cloud/oracle-javacloud-sdk/lib JAVACLOUD_HOME_SOA=/opt/Oracle/MiddlewareSOA_12.2.1.2/jdeveloper/cloud/oracle-javacloud-sdk/lib MIDDLEWARE_HOME_SOA_12_2_1=/opt/Oracle/MiddlewareSOA_12.2.1.2 MIDDLEWARE_HOME_SOA=/opt/Oracle/MiddlewareSOA_12.2.1.2 ORACLE_HOME_SOA_12_2_1=/opt/Oracle/MiddlewareSOA_12.2.1.2/jdeveloper ORACLE_HOME_SOA=/opt/Oracle/MiddlewareSOA_12.2.1.2/jdeveloper WLS_HOME_SOA_12_2_1=/opt/Oracle/MiddlewareSOA_12.2.1.2/wlserver WLS_HOME_SOA=/opt/Oracle/MiddlewareSOA_12.2.1.2/wlserver

Software	Variables
SOA 12.2.1.1	JAVACLOUD_HOME_SOA_12_2_1=/opt/Oracle/MiddlewareSOA_12.2.1.1/jdeveloper/cloud/oracle-javacloud-sdk/lib JAVACLOUD_HOME_SOA=/opt/Oracle/MiddlewareSOA_12.2.1.1/jdeveloper/cloud/oracle-javacloud-sdk/lib MIDDLEWARE_HOME_SOA_12_2_1=/opt/Oracle/MiddlewareSOA_12.2.1.1 MIDDLEWARE_HOME_SOA=/opt/Oracle/MiddlewareSOA_12.2.1.1 ORACLE_HOME_SOA_12_2_1=/opt/Oracle/MiddlewareSOA_12.2.1.1/jdeveloper ORACLE_HOME_SOA=/opt/Oracle/MiddlewareSOA_12.2.1.1/jdeveloper WLS_HOME_SOA_12_2_1=/opt/Oracle/MiddlewareSOA_12.2.1.1/wlserver WLS_HOME_SOA=/opt/Oracle/MiddlewareSOA_12.2.1.1/wlserver
SOA 12.1.3	JAVACLOUD_HOME_12C3=/opt/Oracle/MiddlewareSOA_12.1.3/jdeveloper/cloud/oracle-javacloud-sdk/lib JAVACLOUD_HOME_SOA_12_1_3=/opt/Oracle/MiddlewareSOA_12.1.3/jdeveloper/cloud/oracle-javacloud-sdk/lib JAVACLOUD_HOME_SOA=/opt/Oracle/MiddlewareSOA_12.1.3/jdeveloper/cloud/oracle-javacloud-sdk/lib MIDDLEWARE_HOME_12C3=/opt/Oracle/MiddlewareSOA_12.1.3 MIDDLEWARE_HOME_SOA_12_1_3=/opt/Oracle/MiddlewareSOA_12.1.3 MIDDLEWARE_HOME_SOA=/opt/Oracle/MiddlewareSOA_12.1.3 ORACLE_HOME_12C3=/opt/Oracle/MiddlewareSOA_12.1.3/jdeveloper ORACLE_HOME_SOA_12_1_3=/opt/Oracle/MiddlewareSOA_12.1.3/jdeveloper ORACLE_HOME_SOA=/opt/Oracle/MiddlewareSOA_12.1.3/jdeveloper WLS_HOME_12C3=/opt/Oracle/MiddlewareSOA_12.1.3/wlserver WLS_HOME_SOA=/opt/Oracle/MiddlewareSOA_12.1.3/wlserver

Tip:

- You can run the `env` command as a Shell build step to view all environment variables of the build executor.
- Some Linux programs, such as curl, only support lower-case environment variables. Change the build steps in your job configuration to use lower-case environment variables.

  Example:

```
export http_proxy="$HTTP_PROXY"
export https_proxy="$HTTPS_PROXY"
export no_proxy="$NO_PROXY"
curl -v http://www.google.com
```

Software Installed on the Build Executor

Various software are available in the Software Catalog of Build VM templates. Some of them are available by default when a VM template is created.

Executables from the software bundles are available on the builder's `PATH` variable, which is set to`/usr/bin`, and can be invoked directly from the **Unix Shell**. You should use the `PATH` variable and other environment variables to access the installed software.

This table lists software available in the software catalog, in addition to the software installed by default.

Software	Version	Installed by default?
Ant	1.9.6	Yes
C++ Compiler	4.4.x	Yes
Docker	1.12.x (Oracle Linux 6) 17.12.x (Oracle Linux 6 and 7)	No
Findbugs	3.0.1	No
Firefox	60.3.x	Yes
Fn	0.5.x (Oracle Linux 7)	No
Git	1.7.x (Oracle Linux 6) 1.8.x (Oracle Linux 7)	Yes 1.7.x (Oracle Linux 6) 1.8.x (Oracle Linux 7)
GraalVM	1.0.0-14 (Oracle Linux 7)	Yes
Gradle	5.x	No
Groovy	2.5.x	No
Java	12.x 11.x 1.8.x 1.7.x	Yes 1.8.x (by default)

Software	Version	Installed by default?
Kubectl	1.8.4	No
Maven	3.3.3	Yes
Node.js	0.12.x (Oracle Linux 6 and 7) 8.x (Oracle Linux 6 and 7) 10.x (Oracle Linux 6 and 7) 12.x (Oracle Linux 7)	No
Node.js Driver for Oracle Database	12.1.0.2	No
OCIcli	2.5.x	No
Oracle ATG 11	11.1.x	No
Oracle Developer Studio 12c 12.5	12.5	No
Oracle Forms Developer 12	12.2.1.3.0	No
Oracle Instant Client 12c	12.1.0.2.0	No
Oracle JDeveloper Studio 12	12.2.1.3.0	No
Oracle JDeveloper Studio 11	11.1.1.7.1	No
Oracle SOA Suite 12	12.2.1.3.0 12.2.1.2.0 12.2.1.1.0 12.1.3.0.0	No
Packer	1.3.x	No
PSMcli	1.1.x	No
Python	3.6.x 3.5.x 2.6.x (Oracle Linux 6) 2.7.x (Oracle Linux 7)	Yes 2.6.x (Oracle Linux 6) 2.7.x (Oracle Linux 7)
Python – pip3	9.0.x	No
Ruby	1.9.3p488	Yes
SQLcl	18.x 17.x 4.x	No
Terraform	0.11.x	No
Xvfb	1.15.0 (Oracle Linux 7)	Yes

To know about the environment variables that you can use to access the software, see Build Executor Environment Variables.

Monitor Jobs and Builds from IDEs

You can monitor jobs and builds from IDEs such as OEPE, NetBeans IDE, and JDeveloper.

See these topics for more information:

- Monitor a Project's Builds in Eclipse IDE
- Monitor a Project's Builds in NetBeans IDE
- Monitor a Project's Builds in JDeveloper

Design and Use Job Pipelines

You can create, manage, and configure job pipelines from the **Pipelines** tab of the **Builds** page.

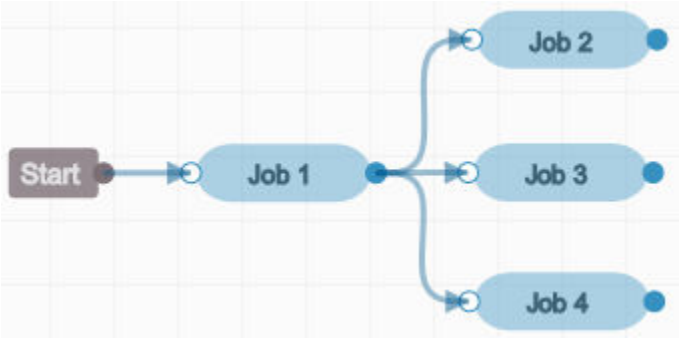
What Is a Pipeline?

A Pipeline lets you define dependencies of jobs and create a path or a chain of builds. A pipeline helps you in running continuous integration jobs and reduce network traffic.

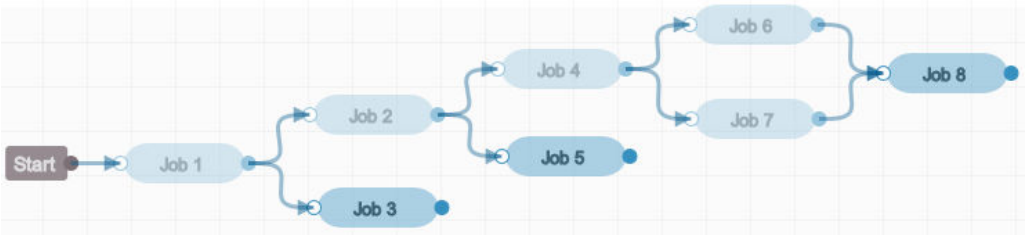
To create a pipeline, you design a pipeline diagram where you define the dependencies of jobs. When you create a dependency of a job over another, you define the order of automatic builds of the dependent jobs. If required, the dependent jobs can be configured to use artifacts of the parent job too.

For example, in this diagram, Job 2 depends on Job 1 and runs after Job 1 is successful.

In this diagram, Job 2, Job 3, and Job 4 depend on Job 1 and run after Job 1 is successful.



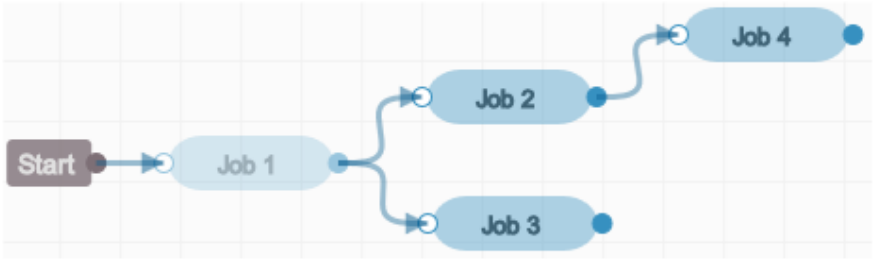
This diagram shows a complex example.



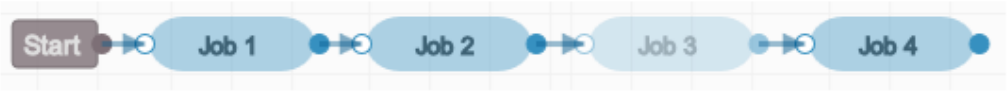
The above diagram defines these dependencies:

- Job 2 and Job 3 depend on Job 1 and run after Job 1 is successful
- Job 4 and Job 5 depend on Job 2 and run after Job 2 is successful
- Job 6 and Job 7 depend on Job 4 and run after Job 4 is successful
- Job 8 depends on Job 6 and Job 7 and runs after Job 6 and Job 7 are successful
- Job 1 is the master job. Running Job 1 triggers a chain and all jobs from Job 1 through Job 8 run automatically one after the other.

You can create multiple pipeline diagrams of jobs. If multiple pipelines have some common jobs, then multiple builds run of those jobs. For example, in this figure, Pipeline 1 and Pipeline 2 have common jobs.



Pipeline 1



Pipeline 2

Let's assume that Pipeline 1 is defined first and Pipeline 2 is defined second. If both pipelines are triggered, the builds run in this order:

1. A build of Job 1 runs.
2. Builds of Job 2 and Job 3 of Pipeline 1 get in the build executor queue after Job 1 is successful. A build of Job 2 of Pipeline 2 also gets in the build executor queue after Job 1 is successful.
3. Builds of jobs in build executor queue run on first-come first-served basis. So, Job 2 and Job 3 of Pipeline 1 run first. Let's call the build as Build 1 of Job 2 and Job 3. Then, another build of Job 2 of Pipeline 2 runs. Let's call it Build 2 of Job 2.
4. A build of Job 4 of Pipeline 1 joins the build executor queue as soon as Job 2 is successful. A build of Job 3 of Pipeline 2 also joins the queue when Job 2 is successful.
5. As soon as the build executor is available, Build 1 of Job 4 runs and Build 2 of Job 3 also runs. Remember that Build 1 of Job 3 ran in Pipeline 1.
6. After a build of Job 3 of Pipeline 2 is successful, a build of Job 4 of Pipeline 2 joins the queue and runs when the build executor is available. Remember that this is Build 2 of Job 4 as Build 1 ran in Pipeline 1.

While creating multiple pipeline diagrams with common jobs, be careful if a job is dependent on artifacts of the parent job.

Set Up a Pipeline

You configure a job pipeline from the **Pipelines** tab of the **Builds** page. To set up a pipeline, design a pipeline diagram.

Create a Pipeline

1. In the navigation bar, click **Builds** .
2. Click the **Pipelines** tab.
3. Click **+ Create Pipeline**.
4. In the Create Pipeline dialog box, in **Name** and **Description**, enter a unique name and description.
5. To trigger the pipeline build if a job of the pipeline is triggered externally (outside the pipeline), select the **Auto start when pipeline jobs are build externally** check box.

   In the pipeline, builds of jobs following the triggered job run as per the diagram, but builds of jobs preceding the triggered job don't run.
6. To disable manual or automatic builds of the jobs that are part of the pipeline when the pipeline is running, select the **Disallow pipeline jobs to build externally when the pipeline is building** check box.
7. Click **Create**.
8. In the Designing Pipeline page, design the pipeline, and click **Save**.

Use the Pipeline Designer

You use the pipeline designer to create a pipeline diagram, that defines dependencies between jobs and the order of their builds.

The **Jobs** list shows all jobs of the project on the left side of the page. Drag and drop jobs to the designer area to design the pipeline diagram. Click **Configure** to configure the dependency condition between the parent and the child job.

Create a One-to-One Dependency

A one-to-one dependency is formed between a parent and a child job. When a build of the parent job is successful, a build of the child job runs automatically.

To create a one-to-one dependency of a child job to its parent job:

1. From the **Jobs** list, drag-and-drop the parent job to the designer area.
2. From the **Jobs** list, drag-and-drop the dependent (or child) job to the designer area.
3. To indicate the parent job, the job that triggers the pipeline build, mouse over the handle of the **Start** node. The cursor icon changes to the + cursor icon.

Example:

In the above example, the **Start** node indicates the starting point of the pipeline. The **Start** node is available in all pipelines and can't be removed. `Job 1` is the parent job and `Job 2` is the dependent job.

4. Drag the cursor from the **Gray circle** handle to the **White circle** handle of the job. An arrow line appears.

   Example:

5. Similarly, mouse-over the **Blue circle** handle of the parent job and drag-and-drop the arrow head over the **White circle** of the child job.

A dependency is now formed. In the above example, `Job 2` is now dependent on `Job 1`. A build of `Job 2` will run automatically after every `Job 1` build is successful.

To delete a job node or a dependency, click to select it, and then click **Delete**.

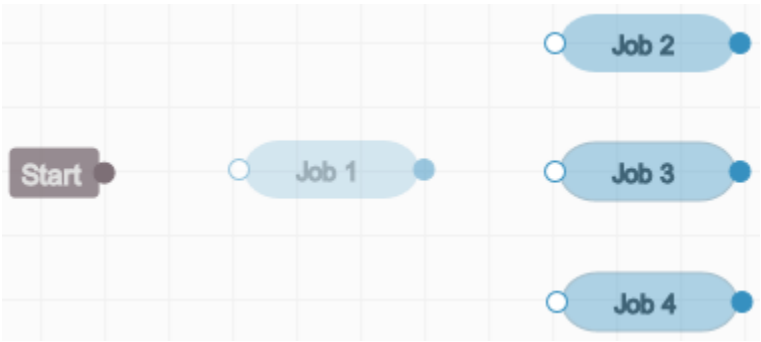
Create a One-to-Many Dependency

A one-to-many dependency is formed between one parent job and multiple child jobs. When a build of the parent job is successful, builds of child jobs run automatically.

To create a one-to-many dependency between jobs:

1. From the **Jobs** list, drag-and-drop the parent job to the designer area.
2. From the **Jobs** list, drag-and-drop all dependent (or child) jobs to the designer area.

   Example:

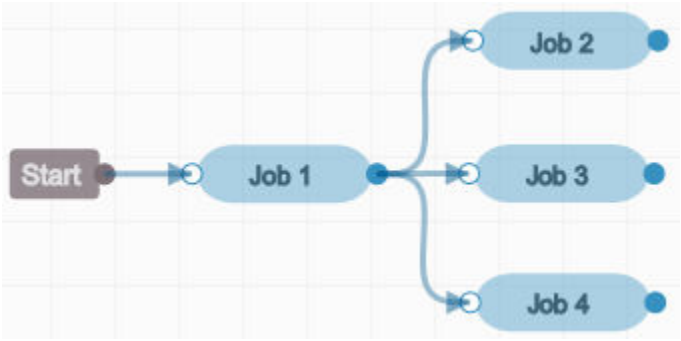


Here, `Job 1` is the parent job and `Job 2`, `Job 3`, and `Job 4` are the dependent jobs.

3. To indicate the parent job, the job that triggers the pipeline build, mouse over the **Gray circle** handle of the **Start** node. The cursor icon changes to the + cursor icon.
4. Drag the cursor from the **Gray circle** handle to the **White circle** handle of the job. An arrow line appears.

   Example:

5. Similarly, mouse-over the **Blue circle** handle of the parent job and drag-and-drop the arrow head over the **White circle** of the child jobs.



A dependency is now formed. In the above example, `Job 2`, `Job 3`, and `Job 4` are now dependent on `Job 1`. A build of `Job 2`, `Job 3`, and `Job 4` runs automatically after every `Job 1` build is successful.

To delete a job node or a dependency, click to select it, and then click **Delete**.

Create a Many-to-One Dependency

A many-to-one dependency is formed between multiple parent jobs and one child job. When builds of all parent jobs are successful, a build of the child job runs automatically.

To create a many-to-one dependency on parent jobs with a child job:

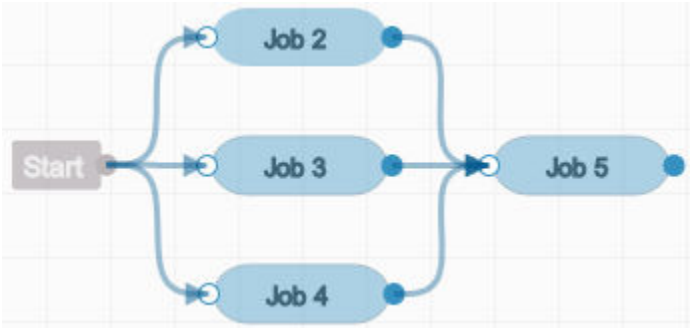
1. From the **Jobs** list, drag-and-drop all parent jobs to the designer area.
2. From the **Jobs** list, drag-and-drop the dependent (or child) jobs to the designer area.

   Example:

Here, `Job 2`, `Job 3`, and `Job 4` are the parent jobs and `Job 5` is the dependent job.

3. To indicate the parent job, the job that triggers the pipeline build, mouse over the **Gray circle** handle of the **Start** node. The cursor icon changes to the + cursor icon.
4. Drag the cursor from the **Gray circle** handle to the **White circle** handle of the parent job. Repeat the steps for all parent nodes.
5. Drag the cursor from the **White circle** handle of the parent job. An arrow line appears. Repeat the steps for all parent nodes.
6. Similarly, mouse over the **Blue circle** handle of the parent jobs and drag-and-drop the arrow head over the **White circle** handle of the dependent job.

   Example:



A dependency is now formed. In the above example, `Job 5` is dependent on `Job 2`, `Job 3`, and `Job 4`. A build of `Job 5` will run automatically after `Job 2`, `Job 3`, and `Job 4` are successful.

To delete a job node or a dependency, click to select it, and then click **Delete**.

Configure the Dependency Condition

When you create a dependency between a parent and a child job, by default, a build of the child job runs after the parent job's build is successful. You can configure the dependency to run a build of the child job after the parent job's build fails.

1. In the pipeline designer, click to select the dependency condition arrow.
2. In the pipeline designer toolbar, click **Configure**.

3. In the Pipeline config flow editor, in **Result Condition**, select **Successful**, **Failed**, or **Test Failed**.

   You can also double-click the dependency arrow to open the Pipeline config flow editor. You can't configure the dependency condition from the **Start** node.

4. Click **Apply**.

Manage Pipelines

You can manage a pipeline by editing the pipeline diagram from the Configure Pipeline page.

Action	How To
Design the pipeline diagram	In the **Pipelines** tab, for the pipeline whose diagram you want to edit, **Configure**  . On the Configuring Pipeline page, click **Configure**  .
Run a pipeline	To run all jobs of a pipe in the defined order, in the **Pipelines** tab, click **Build**  .
View a pipeline's instances	When you trigger a pipeline, an instance of the pipeline is created. To view the instances, in the **Pipelines** tab, click the pipeline name.
Edit a pipeline	In the **Pipelines** tab, for the pipeline you want to edit, **Configure**  . On the Configuring Pipeline page, click **Edit Pipeline Settings**  .
Delete a pipeline	Deleting a pipeline deleted the dependency or the order of job builds. The jobs aren't deleted. In the **Pipelines** tab, for the pipeline you want to delete, click **Delete**  .

View a Pipeline's Instances

When a pipeline build is triggered, an instance of the pipeline is created and is available in the pipeline's instances page.

To view instances of a pipeline, click its name in the **Pipelines** tab. The Pipeline Instances page displays the history of the pipeline run. For each pipeline instance, the page shows the pipeline diagram and its status.
A pipeline's status is determined by the builds of its jobs.

- **Success**  : Indicates that all builds of the pipeline are successful.
- **Failed**  : Indicates that a build of a pipeline job failed, hence the pipeline has failed too.
- **Canceled**  : Indicates the pipeline was canceled.
- **In Progress**  : Indicates a pipeline is in progress.

In the pipeline diagram, the color of job nodes indicates the job's status.

- Green: The last build of the job was successful
- White: A build of the job is running or hasn't run yet
- Red: The last build of the job failed

Deploy Your Application to Oracle Cloud

You can deploy your project's build artifacts to Oracle Java Cloud Service (JCS), to Oracle Application Container Cloud Service (ACCS), and to Oracle Java Cloud Service - SaaS Extension (JCS-SX) from Oracle Developer Cloud Service (DevCS) without leaving its web interface.

> **Note:**
>
> The recommended way to deploy build artifacts to JCS and ACCS is to do so in a build step. You still need to use a deployment configuration to deploy build artifacts to JCS-SX. See Deploy Build Artifacts to Oracle Cloud Services for information about using a build step to deploy applications.

Deployment Concepts and Terms

Here are some concepts and terms that this documentation uses to describe deployment functions and components in DevCS.

Term	Description
Deployment configuration	Defines how to deploy a build artifact to a target Oracle Cloud service.
Deployment target	An instance of the target Oracle Cloud service.
Continuous delivery	A method to automatically deploy a build artifact to the target service.

Deploy an Application to Oracle Java Cloud Service

You can deploy your application to a publicly available JCS instance and make it publicly accessible.

You can deploy your application from DevCS to JCS using the Oracle WebLogic RESTFul Management Interface or the SSH tunnel. You can use either the RESTful interface or SHH to deploy to the Oracle WebLogic Server 12*c* of JCS, but you can only use SSH to deploy to Oracle WebLogic Server 11*g* (10.3.x) of JCS.

Before creating a deployment configuration, enable the HTTPS or the SSH access rule in the JCS console. See Enabling Console Access in an Oracle Java Cloud Service in *Administering Oracle Java Cloud Service*.

Use the Oracle WebLogic RESTFul Management Interface

Deploying to JCS using the Oracle WebLogic RESTFul Management Interface is easy and doesn't require any additional configuration.

1. In the navigation bar, click **Deployments** .
2. Click **+ Create Configuration**.
3. In **Configuration Name**, enter a name to identify the deployment configuration. By default, the same name is used by **Application Name**, however you can modify it.
4. From the **Deployment Target** drop-down list, select the JCS target that uses the Oracle WebLogic RESTFul Management Interface.

   To create a target, see Add an Oracle Java Cloud Service Deployment Target.
5. To deploy the artifact manually, from the **Type** options, select **On Demand**. To deploy the artifact automatically every time the specified job's build is successful, select **Automatic**.
6. In **Job**, **Build**, and **Artifact**, specify the job, build, and artifact to deploy to the target service.

   Build isn't available if you selected **Automatic** as **Type**.
7. If you selected **On Demand** as **Type**, click **Save and Deploy**. If you selected **Automatic** as **Type**, click **Save**.

Use SSH

To deploy to a JCS instance using SSH, you must set up an authenticated connection between DevCS and JCS.

To set up the connection, get the DevCS SSH public key and append it to the `authorized_keys` file of the JCS instance. Then, create a deployment configuration to deploy the artifact.

For more information about accessing the services and resources of a JCS instance via SSH, see Accessing a Node with a Secure Shell (SSH) in *Administering Oracle Java Cloud Service*.

Upload the DevCS public key to the JCS instance

This is a one-time step. After a connection is authenticated, you don't have to repeat the steps to upload the public key.

1. Get the DevCS SSH public key through the REST API.

   Use this URL syntax to get the key:

   `https://<host-name>/<organization-name>/api/deployment/sshkey`

   If the base URL of your DevCS instance is `https://developer.us2.oraclecloud.com/my-org/`, then your host name is `developer.us2.oraclecloud.com` and the organization name is `my-org`.

   Don't enter the URL in the web browser address bar. Instead, use cURL or a browser REST plugin to get the SSH key. For more information about cURL, see `http://curl.haxx.se/`.

   cURL example:

   ```
   curl -u alex.admin:my_password https://
   developer.us2.oraclecloud.com/my-org/api/deployment/sshkey
   ```

Use the same user name and the password that you use to access DevCS. When you have the key, copy the key string value and save it in a `.pub` file (for example, `odcskey.pub`).

2. In the JCS console, open the Overview page of the service instance.

   To know more about the Overview page, see Exploring the Oracle Java Cloud Service Instance Overview Page in *Administering Oracle Java Cloud Service*.

3. In the header of the Overview page, click  next to the **Oracle Java Cloud Service** link and select **SSH Access**.

4. In the Add New Key dialog box, with **Upload a new SSH Public Key value from file** selected, click **Browse**, and select the DevCS SSH public key file that you saved in Step 1.

   You can also copy the key value to the **Key Value** text box.

5. Click **Add New Key**.

6. In the Add New Key dialog box, click **Submit**.

The DevCS SSH public key is added to the `authorized_keys` file of the JCS instance. After adding the public key, the instance restarts automatically. Wait for some time. The icon in the header section also changes. You can track the restart activity in the **Activity Summary** section of the Overview page. When the instance has restarted and the activity is complete, you see the `Add SSH Key is Completed` message in **Activity Summary**.

Create the Deployment Configuration

1. Open DevCS.

2. In the navigation bar, click **Deployments**  .

3. Click **+ Create Configuration**.

4. In **Configuration Name**, enter a name to identify the deployment configuration. By default, the same name is used by **Application Name**, however you can modify it.

5. From the **Deployment Target** drop-down list, select the JCS target that uses SSH.

   To create a target, see Add an Oracle Java Cloud Service Deployment Target.

6. To deploy the artifact manually, from the **Type** options, select **On Demand**. To deploy the artifact automatically every time the specified job's build is successful, select **Automatic**.

7. In **Job**, **Build**, and **Artifact**, specify the job, build, and artifact to deploy to the target service.

   Build isn't available if you selected **Automatic** as **Type**.

8. If you selected **On Demand** as **Type**, click **Save and Deploy**. If you selected **Automatic** as **Type**, click **Save**.

Add an Oracle Java Cloud Service Deployment Target

To deploy to JCS, you need the credentials of a user with the `JaaS_Administrator` (Java Administrators) identity domain role, public IP address of the JCS instance, and the port number of the WebLogic Server.

You can find the IP address and the port number from the software development environment in the Environments page, if configured. You can also find the details from Overview page of the JCS instance. See Exploring the Oracle Java Cloud Service Instance Overview Page and Understanding the Default Access Ports in *Administering Oracle Java Cloud Service*.

1. On the Deployment Configuration page, from the **New** drop-down list of **Deployment Target**, select **Java Cloud Service**.
2. In the Deploy to Java Cloud Service dialog box, select the WebLogic Server version and the protocol (**Oracle WebLogic RESTFul Management Interface** or **SSH**).

   You can't deploy to Oracle WebLogic Server 11g (10.3.x) using the Oracle WebLogic RESTFul Management Interface.
3. In **Host**, enter the public IP address of the JCS instance.
4. In **HTTPS Port**, enter the HTTPS port number of the WebLogic Server that runs on the JCS instance. By default, it's 7002.

   If you've selected the SSH protocol, in **Administration Port**, enter the admin port number of the WebLogic Admin Server that runs on the JCS instance. By default, it's 9001.
5. In **Username** and **Password**, enter the credentials of the user with the `JaaS_Administrator` (Java Administrators) identity domain role.
6. Click **Find Targets**.
7. In the Available Targets dialog box, select the JCS servers or clusters you want to deploy the application to. You can select more than one option.

   To know more about servers and clusters, see the *Targeting Deployments to Servers, Clusters, and Virtual Hosts* topic in *Fusion Middleware Deploying Applications to Oracle WebLogic Server*.
8. Click **OK**.
9. Continue filling in the details in the deployment configuration page.

Deploy an Application to Oracle Application Container Cloud Service

DevCS provides tools to build and deploy your Java, Java EE, PHP, and Node.js applications to ACCS.

You can deploy the application to the ACCS instance of any identity domain and data center. To deploy your application to ACCS, configure a job to create an archive of the application that includes the application, any dependent libraries, and the `manifest.json` file. See Packaging Your Application in *Developing for Oracle Application Container Cloud Service*.

You can deploy the application, undeploy it, and start and stop the application on the target ACCS instance. Other operations, such as scaling up and down of applications,

can be managed from the ACCS console. See About Your Application and Oracle Application Container Cloud Service in *Developing for Oracle Application Container Cloud Service*.

1. In the navigation bar, click **Deployments**  .
2. Click **+ Create Configuration**.
3. In **Configuration Name**, enter a name to identify the deployment configuration. By default, the same name is used by **Application Name**, however, you can modify it.
4. From the **Deployment Target** drop-down list, select the target deployment service.

   To create a target, see Add an Oracle Application Container Cloud Service Deployment Target.
5. In **ACCS Properties**, select the application type and subscription type.
6. To deploy the artifact manually, from the **Type** options, select **On Demand**. To deploy the artifact automatically every time the specified job's build is successful, select **Automatic**.
7. In **Job**, **Build**, and **Artifact**, specify the job, build, and artifact to deploy to the target service.

   Build isn't available if you selected **Automatic** as **Type**.
8. To override the commands of the manifest file in the artifact zip, select the **Include ACCS Manifest** check box. For example, you can override the deployed application's version number at the time of deployment.

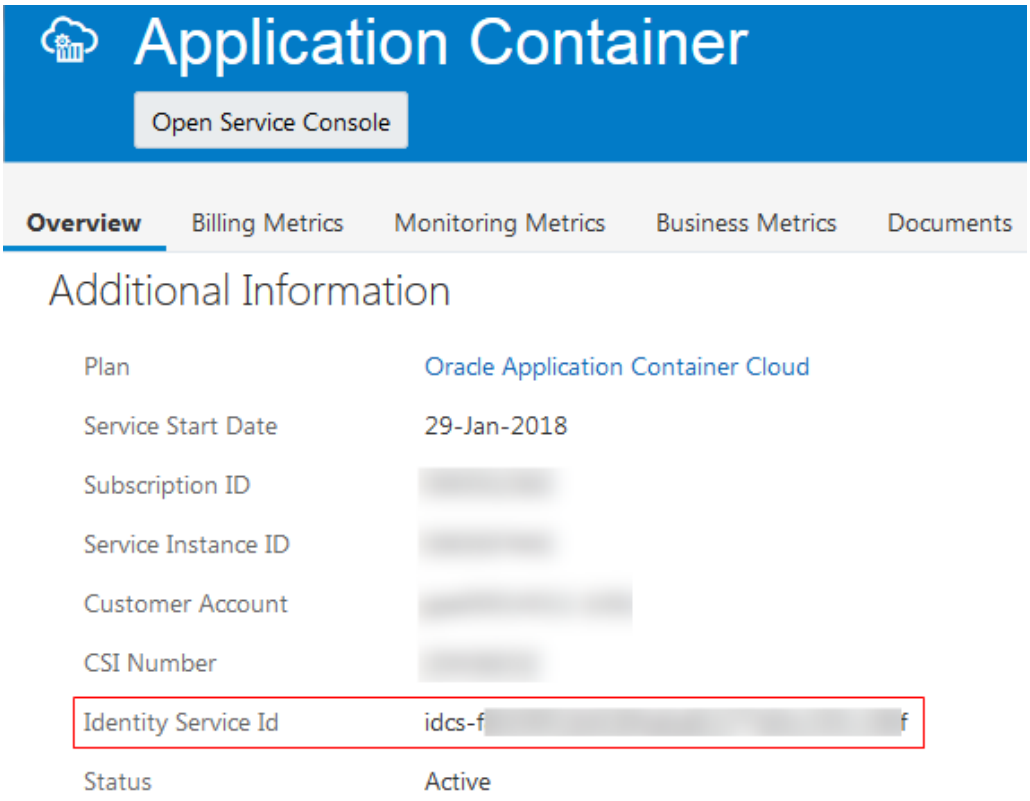
   In **ACCS Manifest**, enter the contents of the manifest file. The field is a code editor component and you can use code editor features.
9. To enter the contents of the deployment descriptor, select the **Include ACCS Deployment** check box and enter the commands in **ACCS Deployment**.

   You can also enter commands to override the deployed application's container's configuration (such as RAM) at the time of deployment. The field is a code editor component and you can use code editor features.
   For more information about the ACCS metadata files, see Creating Metadata Files in *Developing for Oracle Application Container Cloud Service*.
10. If you selected **On Demand** as **Type**, click **Save and Deploy**. If you selected **Automatic** as **Type**, click **Save**.

Add an Oracle Application Container Cloud Service Deployment Target

To deploy to ACCS, you need credentials of a user with the `APaaS_Administrator` (APaaS Administrator) identity domain role and the identity domain of the target service.

If you have an IDCS account, enter the value of **Identity Service Id** in the identity domain field. The **Identity Service Id** is displayed in the ACCS service details page.



1. On the Deployment Configuration page, from the **New** drop-down list of **Deployment Target**, select **Application Container Cloud** .
2. In the Deploy to Application Container Cloud dialog box, in **Data Center**, select the data center of the target ACCS instance.
3. In **Identity Domain**, enter the traditional or IDCS identity domain name.
4. In **Username** and **Password**, enter the credentials of the user with the `APaaS_Administrator` (APaaS Administrator) identity domain role.
5. Click **Test Connection**.
6. After the connection is successful, click **Use Connection**.
7. Continue filling in the details in the deployment configuration page.

Deploy an Application to Oracle Java Cloud Service - SaaS Extension

You can deploy your application artifacts to the JCS-SX instance of the current identity domain and to a service instance of another data center or identity domain,.

To prepare your application to deploy to JCS-SX, see Preparing Applications for Oracle Java Cloud Service - SaaS Extension Deployment in *Using Oracle Java Cloud Service - SaaS Extension*.

1. In the navigation bar, click **Deployments** .
2. Click **+ Create Configuration**.

3. In **Configuration Name**, enter a name to identify the deployment configuration. By default, the same name is used by **Application Name**, however you can modify it.
4. From the **Deployment Target** drop-down list, select the target deployment service.

   To create a target, see Add an Oracle Java Cloud Service - SaaS Extension Deployment Target.
5. To deploy the artifact manually, from the **Type** options, select **On Demand**. To deploy the artifact automatically every time the specified job's build is successful, select **Automatic**.
6. In **Job**, **Build**, and **Artifact**, specify the job, build, and artifact to deploy to the target service.

   Build isn't available if you selected **Automatic** as **Type**.
7. If you selected **On Demand** as **Type**, click **Save and Deploy**. If you selected **Automatic** as **Type**, click **Save**.

See the deployment logs for the status of the deployment.

Add an Oracle Java Cloud Service - SaaS Extension Deployment Target

To deploy to JCS-SX, you need credentials of a user with the `JaaS_Administrator` (Java Administrators) identity domain role and identity domain of the service.

1. On the Deployment Configuration page, from the **New** drop-down list of **Deployment Target**, select **Java Cloud Service - SaaS Extension**.
2. In the Deploy to Java Cloud Service - SaaS Extension dialog box, in **Data Center**, select the data center of the target JCS-SX instance.
3. In **Identity Domain**, enter the traditional or the IDCS identity domain name.
4. In **Username** and **Password**, enter the credentials of the user with the `JaaS_Administrator` (Java Administrators) identity domain role.
5. Click **Find Targets**.
6. In the Available Targets dialog box, click the JCS-SX instance name.
7. Continue filling in the details in the deployment configuration page.

Automatically Deploy a Build Artifact

You can configure a deployment configuration to automatically deploy new version of a build artifact as soon as it becomes available. You can also configure a job to trigger a deployment configuration and deploy artifact as a post-build action.

Action	How To
Configure a deployment configuration to auto-deploy its artifact	You can configure the deployment configuration to automatically deploy an artifact when you create the configuration, or later when you edit the configuration. 1. Create or edit a deployment configuration. 2. In **Configuration Name**, enter the configuration name. 3. Specify the deployment target. 4. In **Type**, select **Automatic**. 5. To deploy an artifact of a build only if it's stable, select **Deploy stable builds only**. 6. If necessary, specify the build job and artifact in **Job** and **Artifact**. 7. Click **Save**. Whenever the job of the deployment configuration runs a build, DevCS deploys the specified artifact of the build immediately.

Action	How To
Configure a job to trigger a deployment action of a deployment configuration	You can configure a job of a deployment configuration to trigger its deployment actions (such as deploy, start, stop, or undeploy) when a build of the job runs. The build is marked as successful if the deployment actions are successful. This is useful if you want to configure a job to run tests on its deployed artifacts. When you create the job and the deployment configuration, you must use the same name for both. 1. On the **Builds** page, create a job and configure it to generate the artifacts that you want to deploy. 2. On the **Deployments** page, click **+ Create Configuration**. 3. In **Configuration Name**, enter the configuration name. Ensure that its name is the same as the job's name. 4. Specify the deployment target. 5. In **Type**, select **On Demand**. 6. Click **Save and Deploy**. 7. Open the job's configuration page again. 8. Click the **Post Build** tab. 9. From **Add Post Build Action**, select **Oracle Cloud Service Deployment**. 10. From **Add Deploy Task**, select the deployment action. For example, select **Deploy** to deploy the artifact of the deployment configuration. 11. In **Deployment Configuration**, select the deployment configuration. 12. If required, add additional deployment actions. For example, you can add an undeploy action first and then add a deploy action of the same deployment configuration. This is useful if you want to make sure that the application is undeployed before it's redeployed. 13. Click **Save**. Now, when a build of the job runs, it automatically triggers the deployment actions specified in the deployment configuration.

Manage Deployment Configurations and Deployments

After you've created a deployment configuration, you can edit its properties, start and stop the deployment, redeploy an application, view deployment logs, and delete a deployment configuration.

Action	How To
Edit a deployment configuration	On the **Deployments** page, in the deployment configuration tile, mouse over **Settings**  and select **Edit Configuration**. You can't change the application name and the deployment target of a deployment configuration.

Action	How To
Start or stop the application	You can start or stop the deployed application on the target service from the deployment configuration. you don't need to open the target service's console to do that. In the deployment configuration tile, mouse over **Settings** and select **Start** or **Stop**.
Redeploy the application	If you've made changes to the source code or the build generated a new artifact, you can manually redeploy the application to the target service. In the deployment configuration tile, mouse over **Settings** and select **Redeploy**. You'll be prompted to specify the build and the artifact to deploy.
View deployment logs	Select the deployment configuration tile and click the log's link on the right side of the page.
Delete a deployment configuration	In the deployment configuration tile, mouse over **Settings** and select **Delete Configuration**. In the Confirm Delete dialog box, select the **Also undeploy** check box to undeploy the application. Click **Delete**.

Access a Deployed Application

To access a deployed application, in the deployment configuration tile, click the Application Name link. Enter your identity domain name and your credentials, if you're prompted to do so.

You can also access the deployed application from the console of the target service. Here are some ways to get the deployed application's URL .

Action	How To
Access an application deployed to JCS-SX	1. From the Oracle Cloud Dashboard, open the Oracle Java Cloud Service - SaaS Extension Control. You can also open it from the deployment configuration tile. In the tile, click the Java Service name link. 2. Click the application name link under **Applications**. 3. Click the application URL under **Application URLs** to access the deployed application. To learn more about accessing JCS-SX, see Accessing Oracle Java Cloud Service - SaaS Extension in *Using Oracle Java Cloud Service - SaaS Extension*.

Action	How To
Create the application's URL that's deployed to JCS	1. Use the JCS View a Service Instance API to get the Content URL and examine the response body output to find the `content_url`. Example: `curl -i -X GET -u jdoe@example.com:my_password -H "X-ID-TENANT-NAME:exampleidentitydomain" https://jaas.oraclecloud.com/jaas/api/v1.1/instances/exampleidentitydomain/exampleservice` For more information about the REST API, see REST API for Oracle Java Cloud Service in *Using Oracle Java Cloud Service*. You must use basic authentication to call the REST API. You can use cURL or a browser REST add-on, such as Postman for Google Chrome to make the call. 2. Get the context root of the application from the `application.xml` deployment descriptor for EAR deployments or from the `web.xml` deployment descriptor for WAR deployments. If there is no such descriptor, you'll need to get the context root from the WebLogic Console. a. Open the WebLogic Console of the JCS instance. You can access the console from the **Java Service** link of the JCS deployment configuration. b. Click **Deployments** in the Domain Structure pane. c. Click the deployed application name in the Deployments table. d. In the Overview tab, copy the value displayed by **Context Root**. Note that the `<host>:<port>` referenced in the WebLogic Console is local to the JCS instance, so you'll need the externally available IP address or the host name of the JCS instance VM to access the deployed application. 3. Join the content URL and the context root of the application to construct the application URL. For example, if the content URL is `http://129.130.131.132` and the context root is `/deploy4214351085908057349`, the application's URL would be `http://129.130.131.132/deploy4214351085908057349`. For more information, see Accessing an Application Deployed to an Oracle Java Cloud Service Instance in *Using Oracle Java Cloud Service*.

Create and Configure Jobs and Pipelines Using YAML

YAML (YAML Ain't Markup Language) is a human-readable data serialization language that is commonly used for configuration files. To find more about YAML, see `https://yaml.org/`.

In DevCS, you can use a YAML file (a file with `.yml` extension) to store a job or a pipeline configuration in any of the project's Git repositories. The build system

constantly monitors the Git repositories and, when it detects a YAML file, creates or updates a job or a pipeline with the specified configuration.

Here's an example of a YAML configuration for a job:

```
job:
  name: MyFirstYAMLJob
  vm-template: Basic Build VM Template
  git:
  - url: "https://mydevcsinstance-mydomain.developer.ocp.oraclecloud.com/
mydevcsinstance-mydomain/s/mydevcsinstance-mydomain_my-project_902/scm/
employee.git"
    branch: master
    repo-name: origin
  steps:
  - shell:
      script: "echo Build Number: $BUILD_NUMBER"
  - maven:
      goals: clean install
      pom-file: "employees-app/pom.xml"
  after:
  - artifacts:
      include: "employees-app/target/*"
  settings:
    - discard-old:
        days-to-keep-build: 5
        builds-to-keep: 10
        days-to-keep-artifacts: 5
        artifacts-to-keep: 10
```

Learn More About Using YAML Files in DevCS

All YAML files must reside in the `.ci-build` directory in the root directory of any hosted Git repository's `master` branch. YAML files in other branches are ignored. Within the `.ci-build` directory on the `master` branch, DevCS assumes any text file ending in the extension `.yml` to be a YAML configuration file. Each YAML file can contain configuration data for exactly one job or one pipeline. You can have YAML files in multiple Git repositories, or use a separate Git repository to host all YAML configuration files. You cannot use an external Git repository to host YAML files. Because the configuration files are stored in Git, you can track changes made to the job's or pipeline's configuration and, if a job or a pipeline is deleted, you can use the configuration file to recreate it.

The build system constantly monitors the Git repositories of the project. When it detects an update to a file with the `.yml` extension in the `.ci-build` directory of a Git repository's `master` branch, it scans the file to determine if it is a job or a pipeline, and creates or updates the corresponding job or pipeline. First, it verifies whether the job or the pipeline of the same name (as in the configuration file) exists on the **Builds** page. If the job or the pipeline exists, it's updated. If the name of the job or pipeline has changed in the configuration file, it's renamed. If the job or the pipeline doesn't exist, it's created.

Note that jobs and pipelines created with YAML can't be edited on the **Builds** page. They must be edited using YAML. Similarly, jobs and pipelines created on the **Builds** page can't be edited using YAML.

YAML stores data as a key-value pair in the `field: value` format. A hyphen (-) before a field identifies it as an array or a list. It must be indented to the same level as the parent field. To indent, always use spaces, not tabs. Make sure that number of indent spaces before a field name matches the number of indented spaces in the template. YAML is sensitive to number of spaces used to indent fields. Also, the field names in a YAML file are similar to the field names in the job configuration user interface.

Example:

```
name: MyFirstYAMLJob
vm-template: Basic Build VM Template
git:
- url: "https://mydevcsinstance-mydomain/.../scm/employee.git"
steps:
- shell:
    script: "echo Build Number: $BUILD_NUMBER"
- maven:
    goals: clean install
    pom-file: "employees-app/pom.xml"
```

If you're editing a YAML file on your computer, always use a text editor with the UTF-8 encoding. Don't use a word processor.

Here's some additional information about YAML files to consider before you create or edit them.

- The `name` field in the configuration file defines the job's or pipeline's name. If no name is specified, the build system creates a job or a pipeline with name as `<repo-name>_<name>`, where `repo-name` is the name of the Git repository where the YAML file is hosted and `<name>.yml` is the name of the YAML file.
  For example, if the YAML file's name is **MyYAMLJob** and it's hosted in the **YAMLJobs** Git repository, then the job's or pipeline's name would be **YAMLJobs_MyYAMLJob**.

  If you add the `name` field later, the job or the pipeline is renamed. It's access URL also changes.
- Each job's configuration must define the `vm-template` field.
- To define a string value, you may or may not use quotes. If a string value contains special characters, always enclose the value in quotes.
  Some examples of special characters are `*`, `:`, `{`, `}`, `[`, `]`, `,`, `&`, `#`, `?`, `|`, `-`, `<`, `>`, `=`, `!`, `%`, `@`, `` ` ``.

  You may use single quotes (`' '`) or double quotes (`" "`). To include a single quote in a single quoted string, escape the single quote by prefixing it with another single quote. For example, to set `Don's job` in the `name` field, use `name=Don''s job` in your YAML file. To use a double quote in a double quoted string, escape the double quote with a backslash (\) character. For example, to set `Don"s job` in the `name` field, use `name=Don\"s job` in your YAML file. No need to escape backslashes in a single quoted string.
- All named passwords must be specified in quotes, unless you're using a password parameter.
  Named Password example:

  ```
  params:
  - string:
      name: myUserName
  ```

```
    value: "don.developer"
    description: My Username
steps:
- docker-login:
     username: $myUserName
     password: "#{PSSWD_Docker}"
```

Password Parameter example:

```
params:
- string:
    name: myUserName
    value: "don.developer"
    description: My Username
  - password:
      name: myPwd
      password: #{PSSWD_Docker}
      description: Defining Build Password
steps:
  - docker-login:
       username: $myUserName
       password: $myPwd
```

- If you specify a field name but don't specify a value, YAML assumes the value to be `null`. This can cause errors. If you don't need to define a value for a field, remove the field name.
  For example, if you don't want to define Maven goals and use the default `clean install`, remove the `goals` field. The following YAML code can cause error because `goals` isn't defined.

```
steps:
- shell:
    script: "echo Build Number: $BUILD_NUMBER"
- maven:
    goals:
    pom-file: "employees-app/pom.xml"
```

- You don't need to define all fields of the job in the YAML file, just the ones you want to configure or change the default value. Make sure you add the parent field(s) when you define a child field.
  Example:

```
steps:
- maven:
    pom-file: "employees-app/pom.xml"
```

- To run a build of the job automatically when its Git repository is updated, use the `auto` field or set `build-on-commit` to `true`.
  For the current Git repository, using `auto` is equivalent to setting `build-on-commit` to `true`. So, don't use `auto` and `build-on-commit: true` together.

  `auto` example:

```
name: MyFirstYAMLJob
vm-template: Basic Build VM Template
auto:
  branch: patchset_1
```

If you use `auto`, don't specify the Git repository URL. The job automatically tracks the Git repository where the YAML file is committed to.

`build-on-commit` example:

```
  name: MyFirstYAMLJob
  vm-template: Basic Build VM Template
  git:
  - url: "https://mydevcsinstance-mydomain.developer.ocp.oraclecloud.com/
mydevcsinstance-mydomain/s/mydevcsinstance-mydomain_my-project_902/scm/
employee.git"
    branch: patchset_1
    build-on-commit: true
```

A commit when pushed to the `patchset_1` branch triggers a build of the **MyFirstYAMLJob** job.

- To add comments in the configuration file, use `#`.
  Example:

```
  steps:
  # Shell script
  - shell:
      script: "echo Build Number: $BUILD_NUMBER"
```

- On the **Builds** page, to configure an existing job or a pipeline, click its **Configure** button or icon. If the job or the pipeline was created in YAML, DevCS opens the YAML file in the code editor on the **Git** page where you can view or edit the configuration.

REST API to Access YAML Files

You can use an API testing tool, such as Postman, or curl commands to run REST API methods. To run curl commands, download curl to your computer. You can also use the Git CLI to run curl commands.

To create the REST API URL, you need your DevCS user name and password, the base URL of your instance, the unique organization ID, and the project ID, which you can get from any of the project's Git repository URLs.

In a Git repository URL, the project's ID is located before `/scm/<repo-name>.git`. For example, if `https://alex.admin%40example.com@mydevcsinstance-mydomain.developer.ocp.oraclecloud.com/mydevcsinstance-mydomain/s/`**`mydevcsinstance-mydomain_my-project_123`**`/scm/NodeJSDocker.git` is the URL of a Git repository in a project, then the project's unique ID is **mydevcsinstance-mydomain_my-project_123**.

Validate a Job's or Pipeline's Configuration

To validate a job's or a pipeline's configuration, use the URL in the following syntax along with the local YAML file on your computer as a parameter:

`https://<base-url>/<identity-domain>/s/<identity-domain>_<unique-projectID>/cibuild/yaml/cibuild/yaml/validate`

Here's an example of a curl command to validate a job's configuration on a Windows computer:

```
curl -X POST -H "Content-Type: text/plain" --data-binary @d:/myApps/
myPHPapp/.ci-build/my_yaml_job.yml -u alex.admin@example.com:My123Password
https://mydevcsinstance-mydomain.developer.ocp.oraclecloud.com/myorg/s/
myorg_my-project_1234/cibuild/yaml/validate
```

Here's an example of a curl command to validate a pipeline's configuration on a Windows computer:

```
curl -X POST -H "Content-Type: text/plain" --data-binary @d:/myApps/
myPHPapp/.ci-build/my_yaml_pipeline.yml -u
alex.admin@example.com:My123Password https://mydevcsinstance-
mydomain.developer.ocp.oraclecloud.com/myorg/s/myorg_my-project_1234/
cibuild/yaml/validate
```

Create a Job or a Pipeline Without Committing the YAML file

To create a job or a pipeline without committing its YAML file to a project Git repository, use the URL in the following syntax along with the local YAML file on your computer as a parameter:

```
https://<base-url>/<identity-domain>/s/<identity-domain>_<unique-
projectID>/cibuild/yaml/cibuild/yaml/import
```

DevCS reads the YAML job or pipeline configuration and, if no errors are detected, creates a new job or pipeline. The job or pipeline must be explicitly named in the YAML configuration. After the job is created, you can edit its configuration on the **Builds** page. If errors are detected, error messages are displayed and the job or pipeline is not created.

Here's an example of a curl command to create a job with a YAML file on a Windows computer:

```
curl -X POST -H "Content-Type: text/plain" --data-binary @d:/myApps/
myPHPapp/my_PHP_yaml_job.yml -u alex.admin@example.com:My123Password
https://mydevcsinstance-mydomain.developer.ocp.oraclecloud.com/myorg/s/
myorg_my-project_1234/cibuild/yaml/import
```

Create or Configure a Job Using YAML

1. On your computer, clone the Git repository that has the YAML file, or where you want host it.
2. Create a file with the job's YAML configuration.

   See Job's YAML Configuration Format.
3. Save the file with the `.yml` extension in the `.ci-build` directory at the root of the cloned Git repository. Example: `.ci-build/my_yaml_job.yml`
4. Validate the local YAML file. See Validate a Job's or Pipeline's Configuration.

   Resolve any errors, if reported.
5. Commit and push the file to the project's Git repository.
6. Open the **Project Home**  page and in the Recent Activities Feed, verify the YAML file's notification. You'll see notifications about the YAML file and the job created.

If there are any validation issues with the YAML file, a notification with a **View Error** link is displayed. Click the **View Error** link to review the error messages. Then, update the YAML file, and commit it again.

7. Click the job's name to open it in the **Builds** page.

You can create the job configuration file online using the code editor on the **Git** page too.
Example:

If you create the YAML file online, you can't validate it without committing. To check any errors, commit the file, and check the notification in the Recent Activities Feed on the **Project Home** page.

Job's YAML Configuration Format

Here is the job's YAML configuration format with the default values. Note that fields with value `""` indicates it accepts a string value, which by default is empty. `""` is not a valid value for some fields, such as `name`, `vm-template`, and `url`. If you want to a field to use its default value, don't add the field in the YAML file.

When you configure a job, fields such as `name`, `description`, `vm-template`, and `auto` must precede groups like `git`, `params`, and `steps`.

```
job:
  name: ""
  description: ""
  vm-template: ""          # required
  auto: false              # true implies branch: master; otherwise, set
branch explicitly
  auto:
    branch: mybranch
  from-job: ""             # create job as copy of another job; ignored
after creation
  for-merge-request: false
  git:
  - url: ""                # required
    branch: "master"
    repo-name: "origin"
    local-git-dir: ""
    included-regions: ""
    excluded-regions: ""
    excluded-users: ""
    merge-branch: ""
    config-user-name: ""
```

```
    config-user-email: ""
    merge-from-repo: false
    merge-repo-url: ""
    prune-remote-branches: false
    skip-internal-tag: true
    clean-after-checkout: false
    update-submodules: false
    use-commit-author: false
    wipeout-workspace: false
    build-on-commit: false
  params:
  # boolean, choice, and string parameters can be specified as string
values of the form - NAME=VALUE
  #   the VALUE of a boolean parameter must be true or false, e.g., -
BUILD_ALL=true
  #   the VALUE of a choice parameter is a comma-separated list, e.g., -
PRIORITY=NORMAL,HIGH,LOW
  #   the VALUE of a string parameter is anything else, e.g., -
URL=https://github.com
  # Alternatively, parameters can be specified as objects:
  - boolean:
      name: ""                # required
      value: true             # required
      description: ""
  - choice:
      name: ""                # required
      description: ""
      choices: []             # array of string value choices; at least
one required
  - merge-request:
      params:
        GIT_REPO_BRANCH=""    # required
        GIT_REPO_URL=""       # required
        MERGE_REQ_ID=""
  - password:
      name: ""                # required
      password: ""            # required - recommended to use named
password reference like "#{NAME}"
      description: ""
  - string:
      name: ""                # required
      value: ""               # required
      description: ""
  before:
  - copy-artifacts:
      from-job: ""
      build-number: 1                # requires which-build:
SPECIFIC_BUILD
      artifacts-to-copy: ""
      target-dir: ""
      which-build: "LAST_SUCCESSFUL"  # other choices: LAST_KEEP_FOREVER,
UPSTREAM_BUILD, SPECIFIC_BUILD, PERMALINK, PARAMETER
      last-successful-fallback: false
      permalink: "LAST_SUCCESSFUL"    # other choices: LAST, LAST_FAILED,
LAST_UNSTABLE, LAST_UNSUCCESSFUL
```

```
                                    # other choices require which-build:
PERMALINK
      param-name: "BUILD_SELECTOR"    # requires which-build: PARAMETER
      flatten-dirs: false
      optional: false
  - oracle-maven:
      otn-login: ""                   # required
      otn-password: ""                # required
      server-id: ""
      settings.xml: ""
  - xvfb:
      display-number: "0"
      screen-offset: "0"
      screen-dimensions: "1024x768x24"
      timeout-in-seconds: 0
      more-options: "-nolisten inet6 +extension RANDR -fp /usr/share/X11/
fonts/misc"
      log-output: true
      shutdown-xvfb-after: true
  steps:
  - ant:
      build-file: ""
      targets: ""
      properties: ""
      java-options: ""
  - bmccli:
      private-key: ""          # required
      user-ocid: ""            # required
      fingerprint: ""          # required
      tenancy: ""              # required
      region: "US_Phoenix_1"
  - docker-build:              # docker commands require vm-template with
software bundle 'Docker'
      source: "DOCKERFILE"     # other choices: DOCKERTEXT, URL
      path: ""                 # file directory
      options: ""
      image:
        registry-host: ""
        registry-id: ""
        image-name: ""         # required
        version-tag: ""
      context-root-path: ""
      docker-text: ""          # required if source: DOCKERTEXT otherwise
not allowed
      context-root-url: ""     # required if source: URL otherwise not
allowed
  - docker-image:
      options: ""
      image:
        registry-host: ""
        registry-id: ""
        image-name: ""
        version-tag: ""
  - docker-load:
      input-file: ""           # required
```

```
- docker-login:
    registry-host: ""
    username: ""            # required
    password: ""            # required
- docker-push:
    options: ""
    image:
      registry-host: ""     # required
      registry-id: ""
      image-name: ""        # required
      version-tag: ""
- docker-rmi:
    remove: "NEW"           # other options: ONE, ALL
    options: ""
    image:                  # only if remove: ONE
      registry-host: ""     # required
      registry-id: ""
      image-name: ""        # required
      version-tag: ""
- docker-save:
    output-file:            # required
    image:
      registry-host: ""     # required
      registry-id: ""
      image-name: ""        # required
      version-tag: ""
- docker-tag:
    source-image:
      registry-host: ""     # required
      registry-id: ""
      image-name: ""        # required
      version-tag: ""
    target-image:
      registry-host: ""     # required
      registry-id: ""
      image-name: ""        # required
      version-tag: ""
- docker-verson:
    options: ""
- fn-build:
    build-args: ""
    work-dir: ""
    use-docker-cache: true
    verbose-output: false
    registry-host: ""
    username: ""
- fn-bump:
    work-dir: ""
    bump: "--patch"         # other choices: "--major", "--minor"
- fn-deploy:
    deploy-to-app: ""       # required
    build-args: ""
    work-dir: ""
    deploy-all: false
    verbose-output: false
```

```
      use-docker-cache: true
      no-version-bump: true
      do-not-push: true
      registry-host: ""
      username: ""
      api-url: ""              # required
  - fn-oci:
      compartment-id: ""       # required
      provider: ""
      passphrase: ""           # required
  - fn-push:
      work-dir: ""
      verbose: false
      registry-host: ""
      username: ""
  - fn-test:
      work-dir: ""
      verbose-output: false
      test-all-functions: false
  - fn-version: {}
  - gradle:
      use-wrapper: false
      make-executable: false                # ignored unless use-wrapper:
true
      from-root-build-script-dir: false     # ignored unless use-wrapper:
true
      tasks: "clean build"
      build-file: "build.gradle"
      switches: ""
      use-workspace-as-home: false
  - maven:
      goals: "clean install"
      pom-file: "pom.xml"
      private-repo: false
      private-temp-dir: false
      offline: false
      show-errors: false
      recursive: true
      profiles: ""
      properties: ""
      verbosity: NORMAL               # other choices: DEBUG, QUIET
      checksum:  NORMAL               # other choices: STRICT, LAX
      snapshot:  NORMAL               # other choices: FORCE, SUPPRESS
      projects: ""
      resume-from: ""
      fail-mode:  NORMAL              # other choices: AT_END, FAST, NEVER
      make-mode:  NONE                # other choices: DEPENDENCIES,
DEPENDENTS, BOTH
      threading: ""
      jvm-options: ""
  - nodejs:
      source: SCRIPT                  # other choice: FILE
      file: ""                        # only if source: FILE
      script: ""                      # only if source: SCRIPT
  - psmcli:
```

```
      username: ""                    # required
      password: ""                    # required
      identity-domain: ""             # required
      region: US                      # other choice: EMEA
      output-format: JSON             # other choice: HTML
  - shell:
      script: ""
      xtrace: true
      verbose: false                  # both verbose and xtrace cannot be
true
  - sqlcl:
      username: ""
      password: ""
      credentials-file: ""
      connect-string: ""
      source: SQLFILE                 # other choice: SQLTEXT
      sql-file: ""                    # only if source: SQLFILE
      sql-text: ""                    # only if source: SQLTEXT
      role: DEFAULT                   # other choices: SYSDBA, SYSBACKUP,
SYSDG, SYSKM, SYSASM
      restriction-level: DEFAULT      # other choices: LEVEL_1, LEVEL_2,
LEVEL_3, LEVEL_4
  - wercker:
      token: ""
      run-id: ""
      pipeline-id: ""
      branch: ""
      message: ""
      script: ""
      status: ""
      display-name: ""
  after:
  - artifacts:
      include: ""                    # required
      exclude: ""
      maven-artifacts: false
      include-pom: false             # ignored unless maven-artifacts:
true
  - git-push:
      push-on-success: false
      merge-results: false
      tag-to-push: ""
      create-new-tag: false
      tag-remote-name: "origin"
      branch-to-push: ""
      branch-remote-name: "origin"
  - javadoc:
      javadoc-dir: "target/site/apidocs"
      retain-for-each-build: false
  - junit:
      include-junit-xml: "**/surefire-reports/*.xml"
      exclude-junit-xml: ""
      keep-long-stdio: false
      organize-by-parent: false
      fail-build-on-test-fail: false
```

```
    archive-media: true
settings:
- discard-old:
    days-to-keep-build: 0
    builds-to-keep: 100
    days-to-keep-artifacts: 0
    artifacts-to-keep: 20
- versions:
    version-map:
      java: "8"
- git-poll:
    cron-pattern: "0/30 * * * * #Every 30 minutes"
- periodic-build:
    cron-pattern: "0/30 * * * * #Every 30 minutes"
- abort-after:
    hours: 0
    minutes: 0
    fail-build: false
- build-retry:
    build-retry-count: 5
    git-retry-count: 5
- log-size:
    max: 50                     # megabytes
- logger-timestamp:
    timestamp: true
- quiet-period:
    seconds: 0
```

YAML Job Configuration Examples

This table shows several examples of YAML job configurations.

Job Configuration	YAML Code

This configuration creates a job to run Maven goals and archive the artifacts.
• Job Name: MyFirstYAMLJob
• Job's Build Template: Basic Build VM Template
• Project Git repository: employee.git
• Maven step
– Goals: clean install
– POM file: employees-app/pom.xml
• After build
– Archived artifacts: employees-app/target/*
	`job:`
	`  name: MyFirstYAMLJob`
	`  vm-template: Basic Build VM Template`
	`  git:`
	`  - url: "https://mydevcsinstance-mydomain/.../scm/employee.git"`
	`  steps:`
	`  - maven:`
	`  goals: clean install`
	`  pom-file: "employees-app/pom.xml"`
	`  after:`
	`  - artifacts:`
	`  include: "employees-app/target/*"`

Job Configuration	YAML Code

This configuration creates a job to run Docker steps that log in, build, and push an image to OCI Registry.
• Job Name: MyDockerJob
• Job's Build Template: Docker and Node.js Template
• Job Description: Job to build and push a Node.js image to OCI Registry
• Project Git Repository: NodeJSMicroDocker.git
• Docker steps
– Docker registry host: iad.ocir.io
– Username: myoci/ociuser
– Password: My123Password
– Image name: myoci/ociuser/mynodejsimage
– Proxy options: --build-arg https_proxy=http://my-proxy-server:80
	`job:`
	`  name: MyDockerJob`
	`  description: Job to build and push a Node.js image to OCI Registry`
	`  vm-template: Docker and Node.js Template`
	`  git:`
	`  - url: "https://mydevcsinstance-mydomain/.../scm/NodeJSMicroDocker.git"`
	`  steps:`
	`  - docker-login:`
	`  registry-host: "https://iad.ocir.io"`
	`  username: "myoci/ociuser"`
	`  password: My123Password`
	`  - docker-build:`
	`  source: "DOCKERFILE"`
	`  options: "--build-arg https_proxy=https://my-proxy-server:80"`
	`  image:`
	`  image-name: "myoci/ociuser/mynodejsimage"`
	`  version-tag: "1.8"`
	`  registry-host: "https://iad.ocir.io"`
	`  path: "mydockerbuild/"`
	`  - docker-push:`
	`  image:`
	`  registry-host: "https://iad.ocir.io"`
	`  image-name: "myoci/ociuser/mynodejsimage"`
	`  version-tag: "1.8"`
	`  - docker-image:`
	`  options: "--all"`

Job Configuration	YAML Code

This configuration creates a job to run SQL commands and script using SQLcl.
• Job Name: RunSQLJob
• Job's Build Template: Basic Build VM Template
• SQL steps
– Username: dbuser
– Password: My123Password
– Connect string: myserver.oracle.com:1521:db1234
– SQL commands:
`CD /home`
`select * from Emp`
– SQL script file: sqlcl/simpleselect.sql
	`job:`
	`  name: RunSQLJob`
	`  vm-template: Basic Build VM Template`
	`  steps:`
	`  - sqlcl:`
	`  username: dbuser`
	`  password: My123Password`
	`  connect-string: "myserver.oracle.com:1521:db1234"`
	`  sql-text: "CD /home\nselect * from Emp"`
	`  source: "SQLTEXT"`
	`  - sqlcl:`
	`  username: dbuser`
	`  password: My123Password`
	`  connect-string: "myserver.oracle.com:1521:db1234"`
	`  sql-file: "sqlcl/simpleselect.sql"`
	`  source: "SQLFILE"`

Job Configuration	YAML Code

This configuration creates a job to run Maven goals and archive the artifacts.
• Job Name: MyADFApp
• Job's Build Template: JDev and ADF Template
• Project Git Repository: ADFApp.git
• Run a build on a push update to the patchset_1 branch: Yes
• Git repository branch: patchset_1
– Files to track for changes: myapp/src/main/web/.*\.java
– Files to ignore for changes: myapp/src/main/web/.*\.gif
– Remove untracked files before running a build: Yes
– Display commit's author in the log: Yes
• Copy artifacts from another job: ADFDependencies
– Artifacts: adf-dependencies.war
• Oracle Maven Repository connection:
– OTN username: alex.admin@example.com
– OTN password: My123Password
• Maven step
– Goals: clean install package
– POM file: WorkBetterFaces/pom.xml
• After build steps
– Artifacts to archive: WorkBetterFaces/target/*.ear
• Other settings:
– Java version: 7
– Discard old builds: Yes
* Number of builds to keep: 50
* Number of builds to keep: 10
– Periodic build trigger
* Hour: 2
* Minutes: 30
– Build retry count: 5
– SCM retry count: 10
– Abort if the build is stuck: 1 hour
	`job:`
	`  name: MyADFApp`
	`  vm-template: Basic Build VM Template`
	`  auto:`
	`  branch: "patchset_1"`
	`  git:`
	`  - url: "https://mydevcsinstance-mydomain/.../scm/ADFApp.git"`
	`  branch: patchset_1`
	`  build-on-commit: true`
	`  included-regions: "myapp/src/main/web/.*\\.java"`
	`  excluded-regions: "myapp/src/main/web/.*\\.gif"`
	`  clean-after-checkout: true`
	`  before:`
	`  - copy-artifacts:`
	`  from-job: ADFDependecies`
	`  artifacts-to-copy: adf-dependencies.war`
	`  - oracle-maven:`
	`  otn-login: "alex.admin@example.com"`
	`  otn-password: My123Password`
	`  steps:`
	`  - maven:`
	`  goals: clean install package`
	`  pom-file: "WorkBetterFaces/pom.xml"`
	`  after:`
	`  - artifacts:`
	`  include: "WorkBetterFaces/target/*.ear"`
	`  settings:`
	`  general:`
	`  - discard-old:`
	`  days-to-keep-build: 50`
	`  builds-to-keep: 10`
	`  software:`
	`  - versions:`
	`  version-map:`
	`  java: 7`
	`  triggers:`
	`  - git-poll:`
	`  cron-pattern: "0/30 5 * 2 *"`
	`  advanced:`
	`  - abort-after:`
	`  hours: 1`
	`  - build-retry:`
	`  build-retry-count: 5`
	`  git-retry-count: 10`

Create or Configure a Pipeline using YAML

1. On your computer, clone the Git repository that has the YAML file, or where you want host it.
2. Create a file with the pipeline's YAML configuration.

3. Save the file with the `.yml` extension in the `.ci-build` directory at the root of the cloned Git repository. Example: `.ci-build/my_yaml_pipeline.yml`
4. Validate the local YAML file. See Validate a Job's or Pipeline's Configuration.

   Resolve issues, if reported.
5. Commit and push the file to the project's Git repository.
6. Open the **Project Home** page and in the Recent Activities Feed, verify the YAML file's notification. You'll see notifications about the YAML file and the pipeline created.

   If there are any issues with the YAML file, a notification with a **View Error** link is displayed. Click the **View Error** link to download a JSON file with the error messages. Review them, update the YAML file, and commit it again.
7. To view the pipeline, open the **Pipelines** tab on the **Builds** page.

   To run a build of the pipeline's jobs, click **Build**. To view its instances, click the pipeline's name. To edit its YAML configuration, click **Configure**.

YAML Pipeline Configuration Examples

This table shows several examples of YAML pipeline configurations.

Pipeline Configuration	YAML Code

In this pipeline configuration, Job 2 depends on Job 1 and runs after Job 1 completes successfully. Job 3 depends on Job 2 and runs after Job 2 completes successfully.
• Pipeline name: My Pipeline
• Description: YAML pipeline configuration
• Start the pipeline if any job in the pipeline runs: Yes
• Allow jobs in the pipeline to run independently while pipeline is running: Yes
Pipeline jobs:
1. Job 1
2. Job 2
3. Job 3
	`pipeline:`
	`  name: My Pipeline`
	`  description: YAML pipeline configuration`
	`  auto-start: true`
	`  allow-external-builds: true`
	`  start:`
	`  - Job 1`
	`  - Job 2`
	`  - Job 3`

Pipeline Configuration	YAML Code

In this pipeline configuration, Job 2, Job 3, and Job 4 depend on Job 1 and run in parallel after Job 1 completes successfully. Job 5 depends on Job 2, Job 3, and Job 4, and runs after all three complete successfully.
• Pipeline name: My Pipeline
• Start pipeline if any job of the pipeline runs: Yes
Pipeline jobs:
1. Job 1
  • Job 2
  • Job 3
  • Job 4
2. Job 5
	`pipeline:`
	`  name: My Pipeline`
	`  auto-start: true`
	`  start:`
	`  - Job 1`
	`  - parallel:`
	`  - Job 2`
	`  - Job 3`
	`  - Job 4`
	`  - Job 5`

In this pipeline configuration, Job 2 runs after Job 1 completes successfully. Job 3, Job 4, and Job 5 run in parallel after Job 2 completes successfully. Job 6 runs after Job 5 completes successfully. Job 7 runs after Job 6, Job 3, and Job 4 complete successfully.
Pipeline name: My Pipeline
Pipeline jobs:
1. Job 1
2. Job 2
  • Job 3
  • Job 4
  a. Job 5
  b. Job 6
3. Job 7
	`pipeline:`
	`  name: My Pipeline`
	`  start:`
	`  - Job 1`
	`  - Job 2`
	`  - parallel:`
	`  - Job 3`
	`  - Job 4`
	`  - sequential:`
	`  - Job 5`
	`  - Job 6`
	`  - Job 7`

In this pipeline configuration, Job 2 runs after Job 1 completes successfully. Job 3 runs if Job 1 fails.
Pipeline name: My Pipeline
Pipeline jobs:
1. Job 1
If Job 1 completes successfully:
  a. Job 2
If Job 1 fails:
  a. Job 3
	`pipeline`
	`  name: My Pipeline`
	`  start:`
	`  - Job 1`
	`  - on succeed:`
	`  - Job 2`
	`  - on fail:`
	`  - Job 3`

Pipeline Configuration	YAML Code

In this pipeline configuration, Job 2 runs after Job 1 completes successfully. Job 3 runs if Job 1 completes successfully but fails tests or any post build action, or if Job 1 fails. Job 3 won't run if Job1 completes successfully.
Pipeline name: My Pipeline
1. Job 1
• (if Job 1 completes successfully) Job 2
• (if Job 1 completes successfully, but tests or other post-processing fails) Job 3
2. (if Job 1 fails) Job 3
	`pipeline:`
	`  start:`
	`  - Job 1`
	`  - on succeed:`
	`  - Job 2`
	`  - on post-fail:`
	`  - Job 3`
	`  - on fail:`
	`  - Job 3`

7

Integrate with External Software

You can integrate Oracle Developer Cloud Service with some common external software, such as Docker registries, Slack, and Jenkins.

This table describes the Oracle Developer Cloud Service pages you'd use to integrate external software.

Use this page ...	To:
Docker	View and link external Docker registries to the project.
Webhooks	Send Oracle Developer Cloud Service event notifications using webhooks to external software.

Send Notifications to External Software Using Webhooks

Using Webhooks, you can send notifications to remote services and applications about Oracle Developer Cloud Service (DevCS) events such as a Git push, an issue update, a merge request update, or a build completion.

When you create a webhook, you specify a webhook provider. When an event occurs and the webhook triggers, the webhook provider processes the event, sets the properties used to generate the HTTP request, and dispatches the HTTP request to the target service.

You must be assigned the project **Owner** role to create and configure a webhook.

Slack

Slack is a cloud based team collaboration software. Using a Slack Webhook, you can configure DevCS to send events and activities notifications to a Slack channel. To find more about Slack, see `https://slack.com/`.

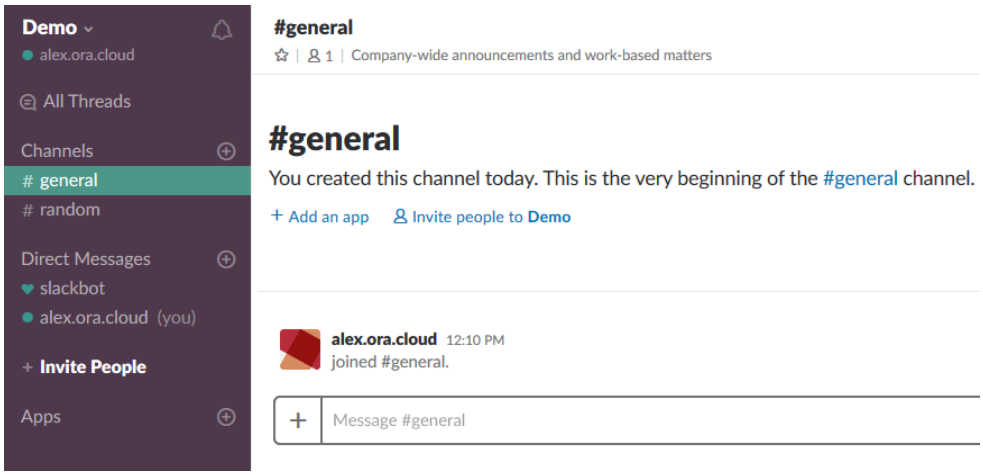
To send notification to a Slack channel, get its incoming webhook URL. Then, create a DevCS webhook and add the incoming webhook URL to the webhook.

Get the Slack Channel's Incoming Webhook URL

You must be the workspace owner to get the incoming webhook URL.

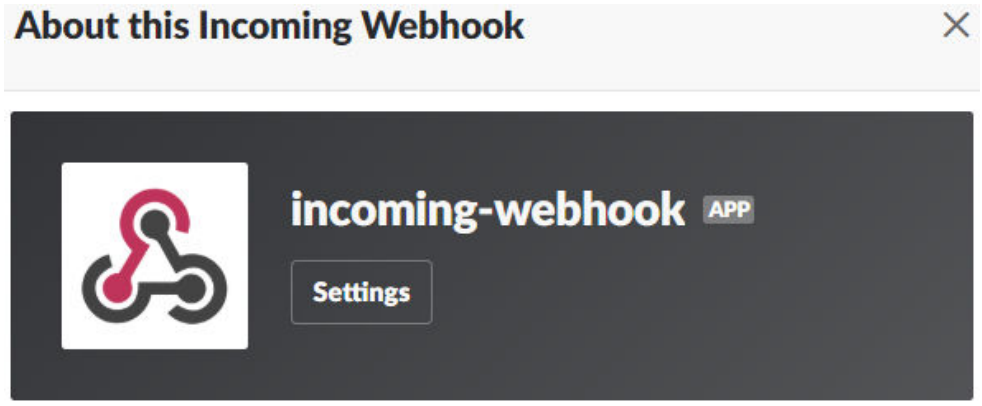
1. Open the Slack workspace in a web browser or the Slack app.

   For example, this image shows a Slack workspace called Demo.



2. In the left navigation bar, click **Apps**.
3. In the search box on the Browse Apps page, enter `incoming webhook`.

4. If **incoming-webhook** is pre-installed, click **View**, and then click **Settings**.

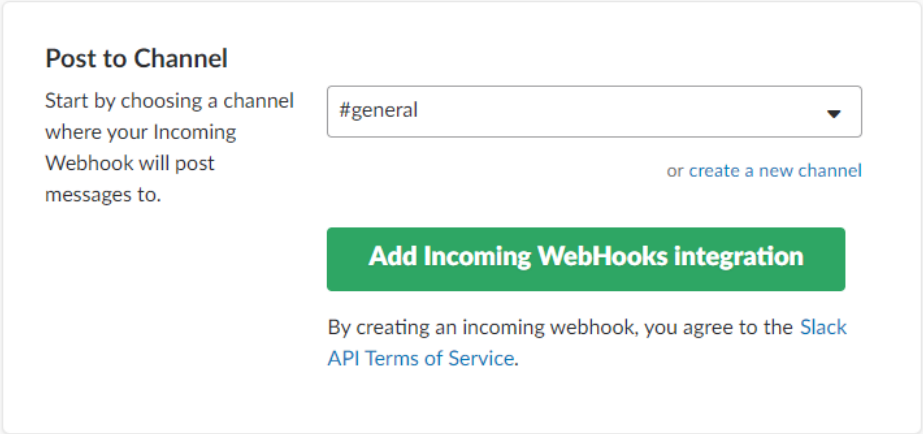


If it isn't installed, then install it and configure it.

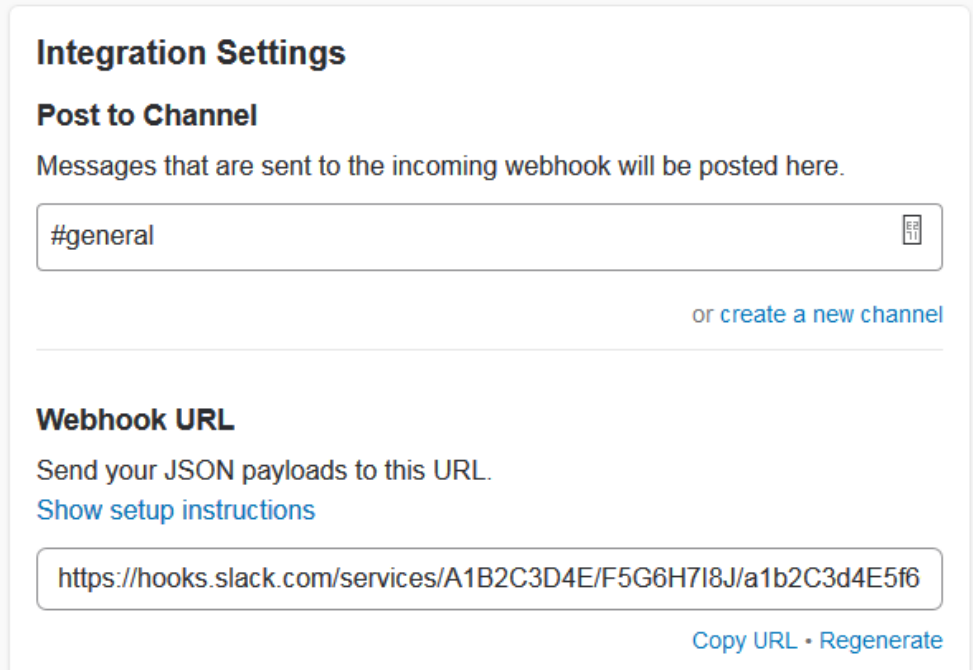
a. Click **Install**.

b. On the Incoming WebHooks page, click **Add Configuration**.

c. From the **Post to Channel** list, select the channel, and click **Add Incoming WebHooks integration**.



5. In **Integration Settings**, from the **Post to Channel** drop-down list, select the channel. In **Webhook URL**, click **Copy URL**.



6. Scroll down to the bottom of the page and click **Save Settings**.

Configure a Slack Webhook in DevCS to Send Event Notifications

The Slack webhook is a outgoing webhook used to send DevCS event notifications to a Slack channel.

You must be assigned the project **Owner** role to create and configure a webhook.

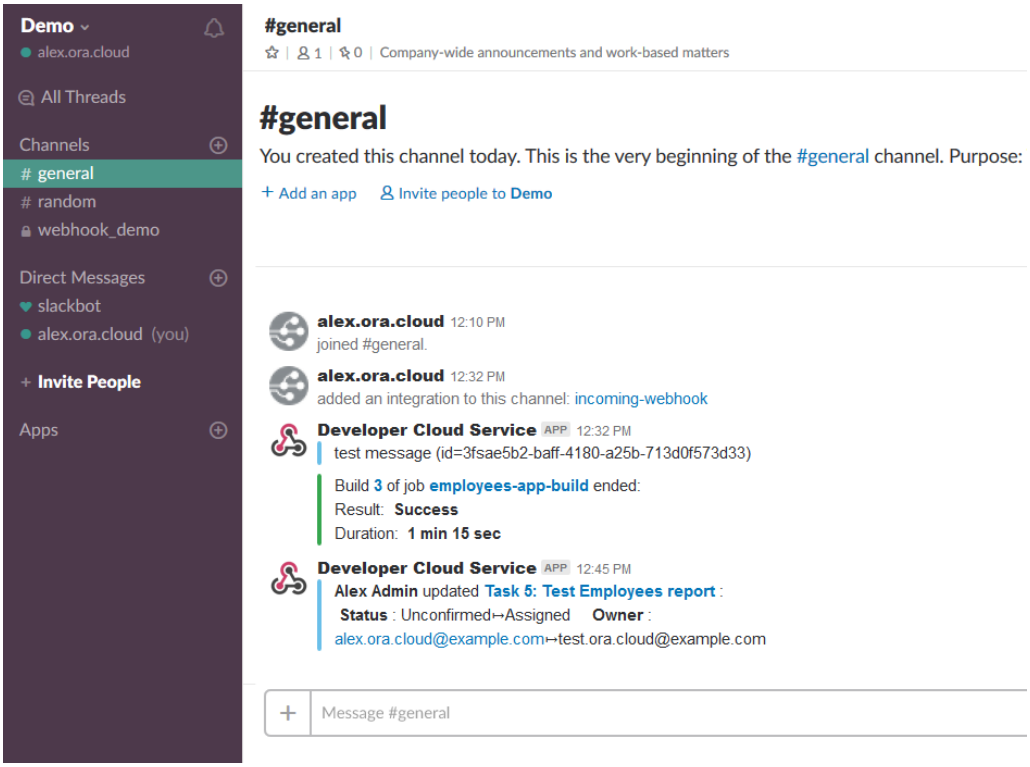
1. In the navigation bar, click **Project Administration** .
2. Click **Webhooks**.
3. Click **+ Create Webhook**.

4. From **Type**, select **Slack**.
5. In **Name**, enter a unique name.
6. In **URL**, enter or paste the Slack channel's incoming Webhook URL.

   Make sure it's in the `https://hooks.slack.com/services/...` format.
7. In **Subscribe**, select the events that trigger the webhook.

   If you select the **Select specific events** option, in **Events**, select the check boxes of events that trigger the webhook.
8. To test the webhook, click **Test**.
9. Click **Done**.

When DevCS events happen, notifications are sent to the Slack channel.



Oracle Social Network

Oracle Social Network (OSN) is a secure enterprise collaboration and social networking solution for business. Using the Oracle Social Network webhook, you can send the DevCS events and activities to OSN conversations.

To send notification to a an OSN conversation, set up an DevCS Incoming Webhook in OSN and associate it with an OSN Conversation. When set up, the Incoming Webhook provides a URL with an authentication token to use in the OSN Webhook of your project. For more information, see `https://cloud.oracle.com/social-cloud`.

You can also set up the OSN functionality for Oracle Public Cloud from Oracle Content and Experience Cloud. The Oracle Content and Experience Cloud Administrator can create an incoming Webhook integration, associate it with an OSN conversation, and

get the URL with an authentication token to use in the OSN Webhook of your project. See Configuring Webhooks in *Administering Oracle Content and Experience Cloud*.

Get OSN Conversation's Incoming Webhook URL

You must be the OSN Administrator to set up or get the incoming webhook URL.

1. Sign in to Oracle Social Network as an administrator.
2. Click **Webhooks**.
3. To the right of **Generic Incoming Webhook**, click **New Instance**.
4. In **Webhook Name**, enter a name.
5. In **Target Conversation or Wall**, specify the OSN conversation.
6. In **Message Template**, specify the wording of the text to be included in the webhook-based message.
7. Fill in the details in other fields of the webhook and click **Save**.
8. In **Webhook URL**, click **Copy to Clipboard**.

Configure an OSN Webhook in DevCS to Send Event Notifications

The Oracle Social Network webhook is a outgoing webhook used to send DevCS event notifications to an OSN conversation.

You must be assigned the project **Owner** role to create and configure a webhook.

1. In the navigation bar, click **Project Administration** .
2. Click **Webhooks**.
3. Click **+ Create Webhook**.
4. From **Type**, select **Oracle Social Network**.
5. In **Name**, enter a unique name.
6. In **OSN URL**, enter the OSN webhook URL (without the authentication token) where the webhook receiver is registered.
7. In **Event Groups**, select the events to trigger the webhook.

   If you selected the **Select specific events** option, in **Events**, select the check boxes of events that trigger the webhook.
8. Click **Done**.

PagerDuty

PagerDuty is an incident management platform that enables you to send notifications via email, push, SMS, and phone. Using the PagerDuty webhook, you can send notifications to your PagerDuty service about events in DevCS. When the PagerDuty service receives notifications from DevCS, it can redirect those notifications via email, push, SMS, and phone. To find more about PagerDuty, see `https://www.pagerduty.com/`.

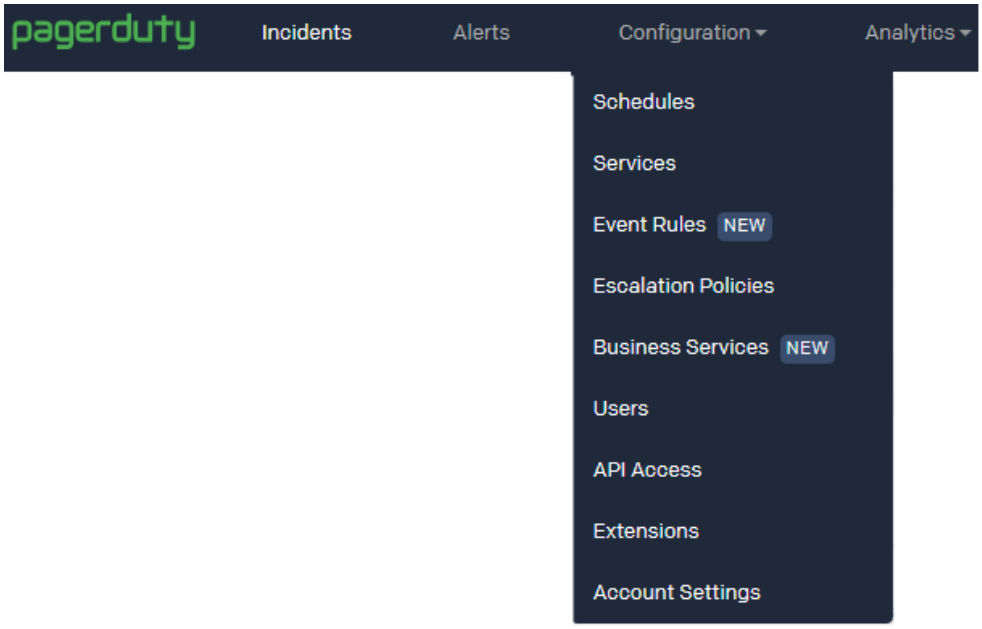
To send notifications to PagerDuty, set up your PagerDuty account to receive notifications and create a DevCS webhook.

Set Up the PagerDuty Account

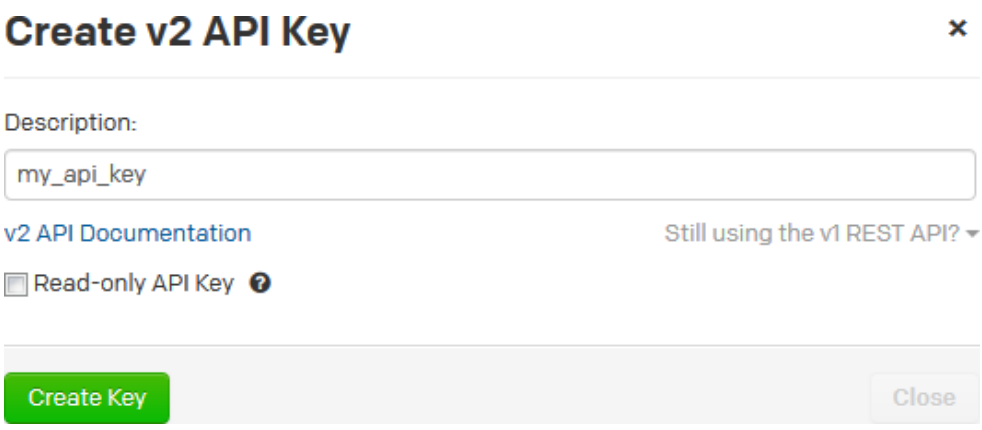
To set up PagerDuty, create an API key, add services, and add users who would receive PagerDuty notifications .

You must be the account owner or assigned the PagerDuty Admin role to set up the PagerDuty account.

1. Log in to PagerDuty as the account owner or administrator.
2. To set up the API key, from the **Configuration** menu, select **API Access**.

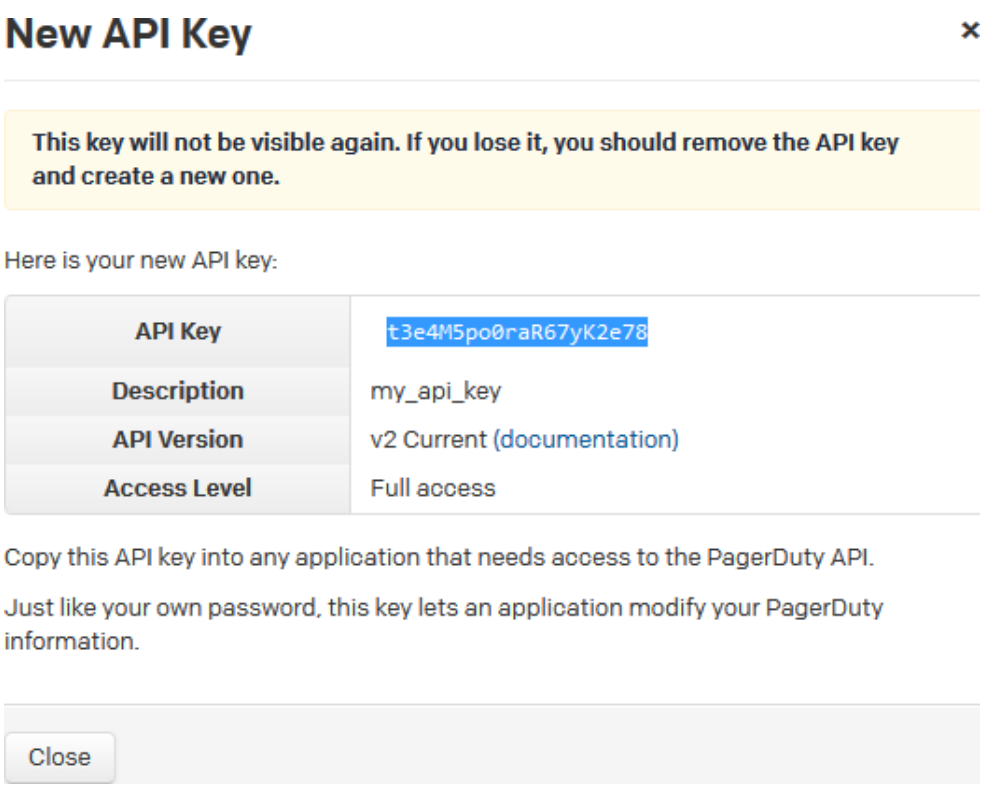


3. Click **Create New API Key**.
4. In the Create API Key dialog box, enter a name for the key and click **Create Key**.

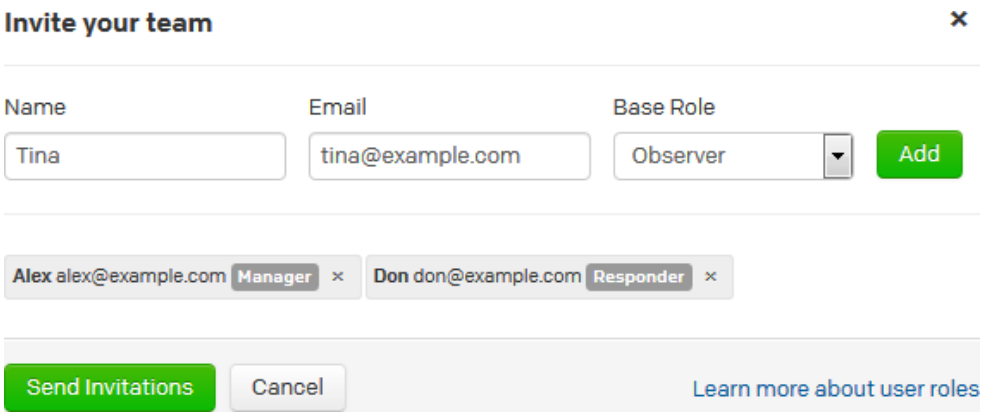


5. From the New API Key dialog box, copy the **API Key** value and keep it safe.

   You can't view or copy the key after closing the dialog box.



6. Click **Close**.
7. If not configured, set up services (such as applications or components) you wish to open incidents against. From the **Configuration** menu, select **Services**.
8. Click **New Service**.
9. Fill in the details and click **Add Service**.
10. If not configured, add users who'd receive notifications. From the **Configuration** menu, select **Users**.
11. Click **Add Users**.
12. In the Invite your team dialog box, add the details of users you want to invite, and click **Add**.



13. When you're finished adding users, click **Send Invitations**.

14. Return to the dashboard page of PagerDuty.

Configure a PagerDuty Webhook in DevCS to Send Event Notifications

The PagerDuty webhook is a outgoing webhook used to send DevCS event notifications to a PagerDuty account.

You must be assigned the project **Owner** role to create and configure a webhook.

1. In the navigation bar, click **Project Administration** .
2. Click **Webhooks**.
3. Click **+ Create Webhook**.
4. From **Type**, select **PagerDuty**.
5. In **Name**, enter a unique name.
6. In **API Key**, enter the API key of the PagerDuty service.
7. In **Service**, select the desired PagerDuty service from the list. The webhook sends event notifications to the selected service.
8. In **Sender**, select the PagerDuty registered user whose name will be attached to the events sent by the webhook.
9. In **Event Groups**, select the events that trigger the webhook.

   If you selected the **Select specific events** option, in **Events**, select the check boxes of events that trigger the webhook.
10. Click **Done**.

Jenkins

Jenkins is an open-source continuous integration software used to build and test your software applications. Using the various Jenkins webhooks, you can integrate your Jenkins with DevCS to run builds. Jenkins must be available on the public Internet to accept webhook notifications.

You can use these webhooks to integrate Jenkins with DevCS:

To do this ...	Use this webhook
Trigger a Jenkins job on SCM polling of the job's Git repository	Hudson/Jenkins Git Plugin
Trigger a Jenkins job on a project's Git repository update	Hudson/Jenkins Build Trigger
Link a Jenkins job with a merge request	Jenkins Merge Requests
Receive notifications in DevCS project's activity feed from Jenkins when a job's build runs or completes	Jenkins Notification Plugin

Trigger a Jenkins Job on SCM Polling

Using the Hudson/Jenkins - Git Plugin Webhook, you can trigger a Jenkins job that uses a DevCS Git repository as source on SCM polling.

To trigger the Jenkins job:

1. If not installed, install the Git plugin.
2. Create or configure the Jenkins job to use the DevCS project Git repository as source.
3. Enable SCM polling in the Jenkins job.
4. Create or configure a webhook to send a notification to Jenkins when the job's Git repository (or any project Git repository) is updated.

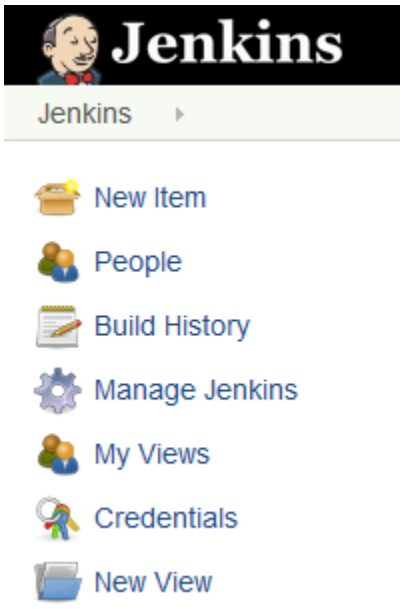
When the Git plugin of Jenkins receives a notification, it goes through all Jenkins jobs that have SCM polling enabled and match the provided notification parameters (such as Git repositories and branches). For all matching jobs, it starts a build. The build won't run if no changes are found by polling.

For more information about the Jenkins Git plugin, see `https://wiki.jenkins-ci.org/display/JENKINS/Git+Plugin#GitPlugin-Pushnotificationfromrepository`.

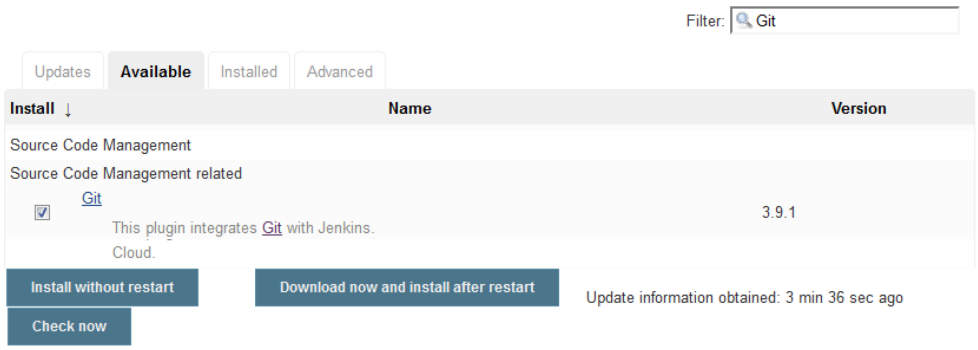
Set Up Git on Jenkins

You must be assigned the Admin role of Jenkins to set up Git on it. Git must also be installed on the computer running Jenkins. If the plugin is already installed and configured, ignore this section.

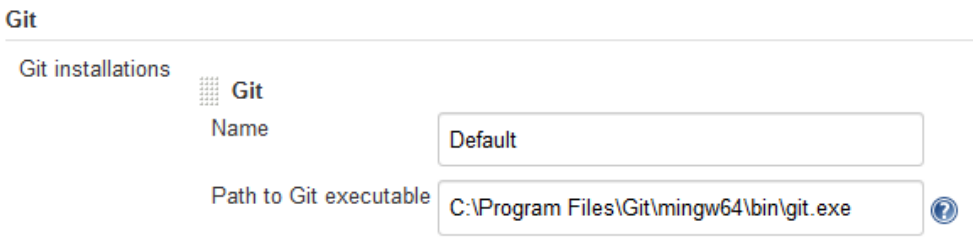
1. Log on to Jenkins using the administrator credentials.
2. From the links on the left side of the page, click **Manage Jenkins**.



3. To install the Git plugin, click **Manage Plugins**.
4. In the **Available** tab, search for `Git`. Under **Source Code Management**, select the plugin's check box and click **Download now and install after restart** or **Install without restart**.



5. Wait for the plugin to install.
6. Restart Jenkins.
7. From the links on the left side of the page, click **Manage Jenkins**.
8. Click **Global Tool Configuration**.
9. In **Git**, enter the local path of the Git executable.



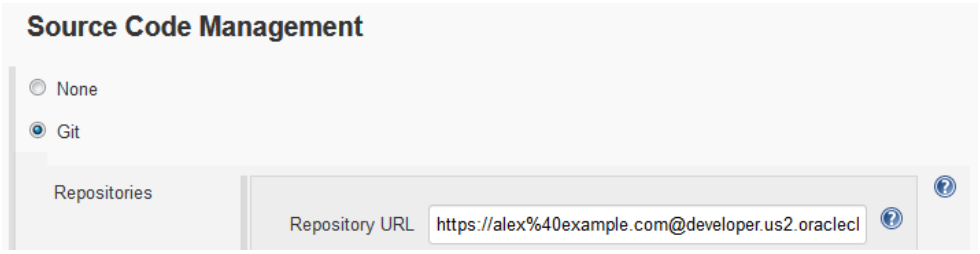
10. Click **Save**.

Configure the Jenkins Job to Use DevCS Git Repository and Enable SCM Polling

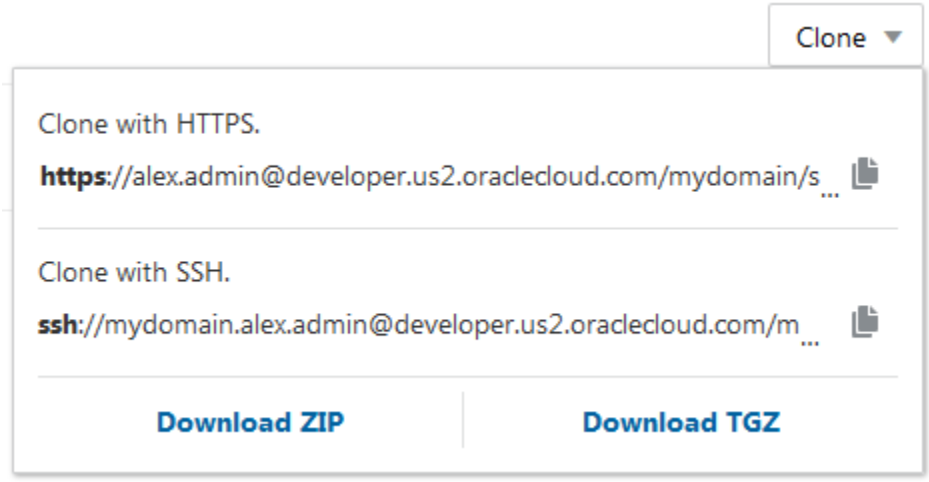
Configure the job to use the DevCS Git repository and enable SCM polling.

1. Log on to Jenkins.
2. Create or open a job.
3. From the links on the left side of the page, click **Configure**.
4. Click the **Source Code Management** tab.
5. Select **Git**.
6. In **Repository URL**, enter the DevCS project's Git repository URL.

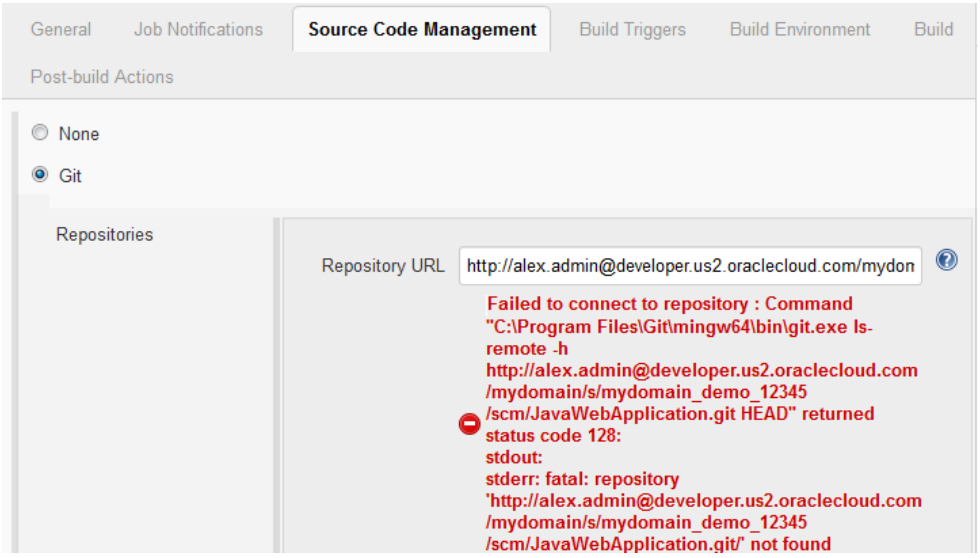
   Remember the URL's protocol as you'd need to specify it when you create the webhook.



You can copy the URL from the **Clone** menu of the DevCS **Git** page.

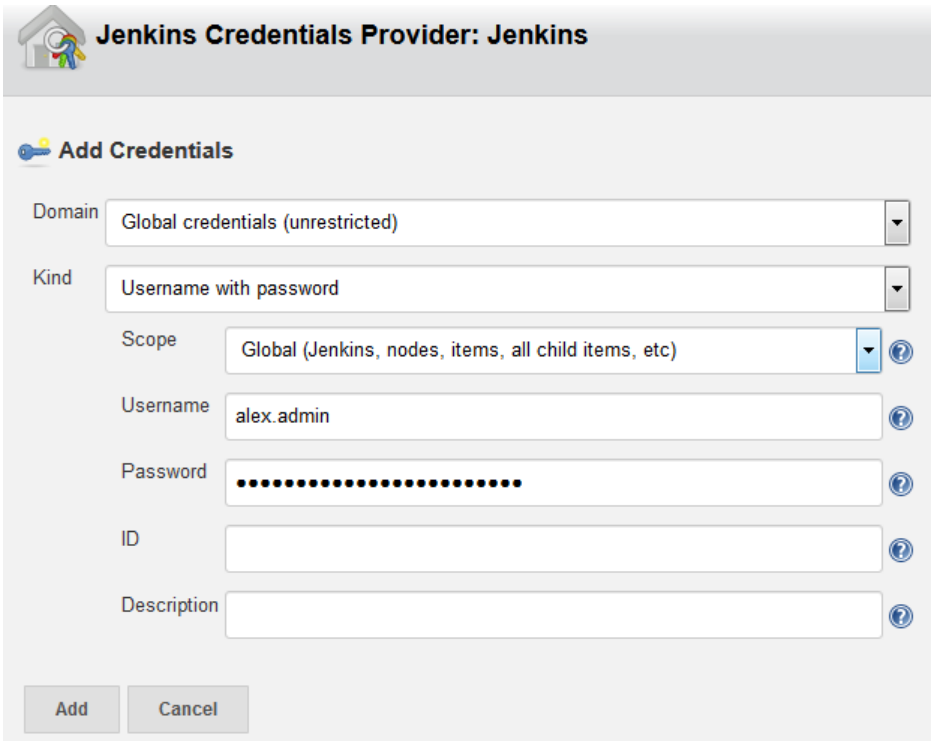


After entering the URL, you might see a `Failed to connect to repository` … error message. It appears because you haven't provided the DevCS access credentials in Jenkins.



a. Next to the **Credentials** list, click **Add** and then select **Jenkins**.

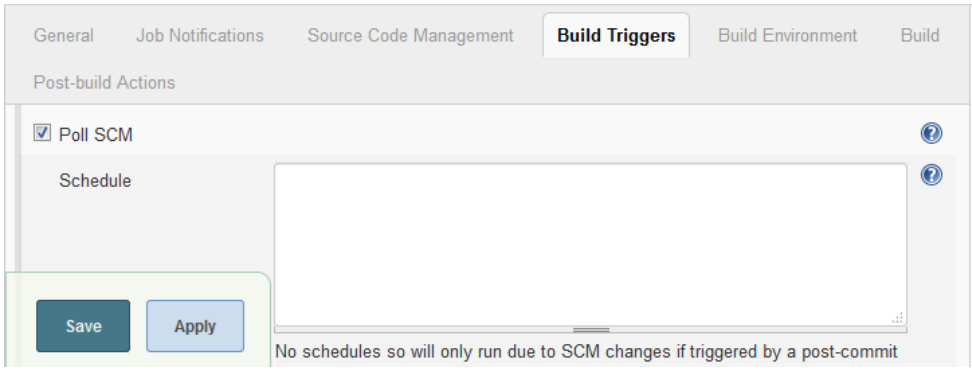
b. In the Jenkins Credentials Provider dialog box, enter the DevCS username and password in **Username** and **Password**. Leave other fields with their default values.



c. Click **Add**.

The error message should disappear. If you still see the error message, configure the proxy settings of Jenkins. See the Jenkins documentation to know more.

7. Click the **Build Triggers** tab.
8. Select the **Poll SCM** check box.



9. Continue to configure the job.
10. When you're finished, click **Save**.

Configure a Webhook in DevCS to Trigger a Jenkins Job on a Git Repository Update

After configuring the Jenkins job, create the DevCS webhook to trigger the job on Git repository update.

1. In the navigation bar, click **Project Administration**  .
2. Click **Webhooks**.
3. Click **+ Create Webhook**.
4. From **Type**, select **Hudson/Jenkins - Git Plugin**.
5. In **Name**, enter a unique name.
6. In **Notification URL**, enter the URL of the target Jenkins server.

   The URL must be in the `http://your_server/.../git/notifyCommit` format. Example: `http://my_jenkins.com:8080/git/notifyCommit`
7. To ignore SSL errors, select the **Ignore SSL Errors** check box.
8. In **Notification Parameters**, specify the URL type.
   - In **Repository URL Type**, select **HTTP Repository Address** to send the HTTP URL of the selected Git repository in the webhook notification. Select **SSH Repository Address** to send the SSH URL of the selected Git repository in the webhook notification.

     You must specify the same protocol that's used in the Jenkins job configuration.
   - In **Append**, to append the SHA-1 Checksum hash of the last commit in the webhook notification, select the **sha1 (Jenkins only)** check box.
   - To append branch information of the last commit in the webhook notification, select the **branches** check box. This enables jobs to poll the specified branches only.
9. In **Repository** and **Branches**, specify the Git repository and branches that trigger the webhook.

   In **Repository**, select **All Repositories** to trigger all Jenkins jobs that uses a Git repository of the project.
10. Click **Done**.

Trigger a Jenkins Job on a Git Repository Update

Using the Hudson/Jenkins - Build Trigger webhook, you can trigger a Jenkins job when a project Git repository updates. It's not necessary that the Jenkins job uses a DevCS project Git repository as source.

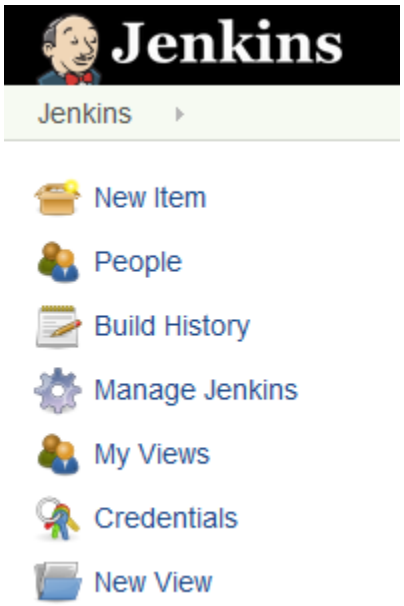
To allow the webhook to connect to Jenkins, you'd need to specify the security settings of Jenkins.

If ...	Do this:
Jenkins allows anonymous user to trigger a build	1. Create an authentication token in the Jenkins job. 2. Configure the webhook to connect to the Jenkins job using the authentication token.
Jenkins allows only authenticated users to trigger a build	1. Get an authenticated user's API Access token. 2. Create an authentication token in the Jenkins job. 3. Configure the webhook to connect to the Jenkins job using the API Access and the authentication token.
Anonymous access is disabled or lacks read permissions on Jenkins and you want to trigger the job without an authenticated user's credentials Or Jenkins uses a build token root to trigger builds	1. If not installed, install the Build Authorization Token Root Plugin on Jenkins. 2. Create an authentication token in the Jenkins job. 3. Configure the webhook to connect to Jenkins job using the authentication token.
Security is completely disabled on Jenkins.	Configure the webhook to connect to Jenkins job. No Jenkins configuration required.

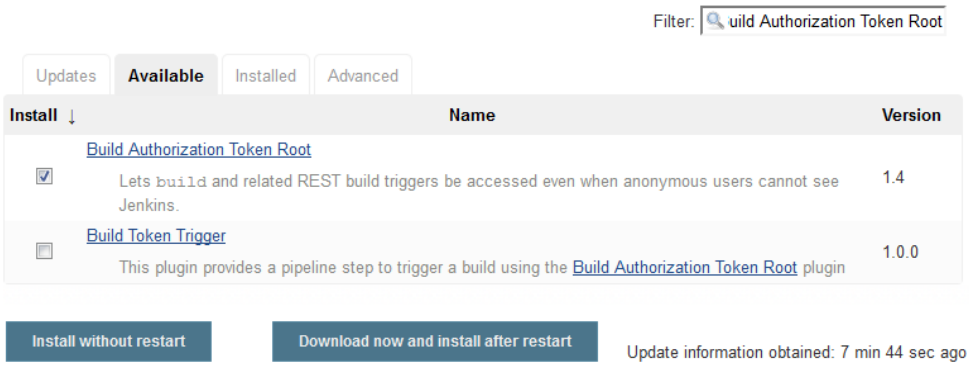
Install the Build Authorization Token Root Plugin on Jenkins

If anonymous access is disabled on Jenkins and you want to trigger Jenkins jobs without an authenticated user's credentials, install the Build Authorization Token Root plugin on Jenkins. You must be assigned the Admin role of Jenkins to install the plugin. The plugin is required To find out more about the plugin, see `https://wiki.jenkins-ci.org/display/JENKINS/Build+Token+Root+Plugin`.

1. Log on to Jenkins using the administrator credentials.
2. From the links on the left side of the page, click **Manage Jenkins**.



3. Click **Manage Plugins**.
4. In the **Available** tab, search for `Build Authorization Token Root`, select its check box, and click **Download now and install after restart** or **Install without restart**.



5. Wait for the plugin to install.
6. Restart Jenkins.

Get the Jenkins API Access Token

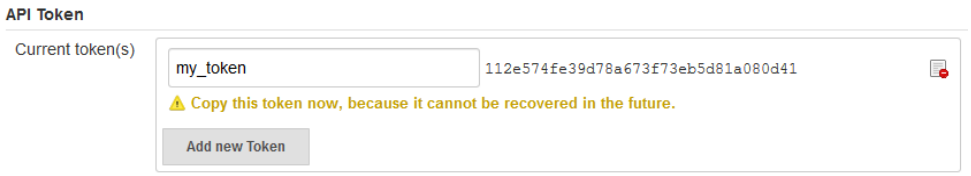
If Jenkins allows only authenticated users to trigger builds, use the API Access token of an authenticated user as the user's credentials in the DevCS webhook.

To use the API Access token in a DevCS webhook, provide the username and the token of an authenticated user. If you don't want to provide a user's details, create a separate username to trigger builds and assign the user the **Overall/Read**, **Job/Read** and **Job/Build** permissions. Then, use this user's details in the webhook.

1. Log on to Jenkins using the user's credentials whose API Access Token you want to use in the webhook.

2. In the upper-right corner, mouse over the user name, click ▾ and select **Configure**.
3. In the **API Token** section, add a new token or use the legacy token.

   To view the legacy token, click **Show Legacy API Token** and then copy the token. Keep the token value some place safe as you'd need to enter in the DevCS webhook.
   To create a token, click **Add new token** and copy the token value immediately. You can't see the token value later and would need to generate another token. Keep the token value some place safe as you'd need to enter in the DevCS webhook.



4. Click **Save**.

Configure the Jenkins Job to Set an Authentication Token

You need to set the authentication token if Jenkins allows anonymous access, access to authenticated users only, or uses the build token root plugin. Use the same token name when you configure the webhook.

1. Log on to Jenkins.
2. Click the job name.
3. From the links on the left side of the page, click **Configure**.
4. Click the **Build Triggers** tab.
5. Select the **Trigger builds remotely (e.g., from scripts)** check box.

6. In **Authentication Token**, enter a unique string as a token. You can enter any string value. Example: `my_auth_token`

   Make sure that the authentication token isn't used in any other job.
7. Continue to configure the job.

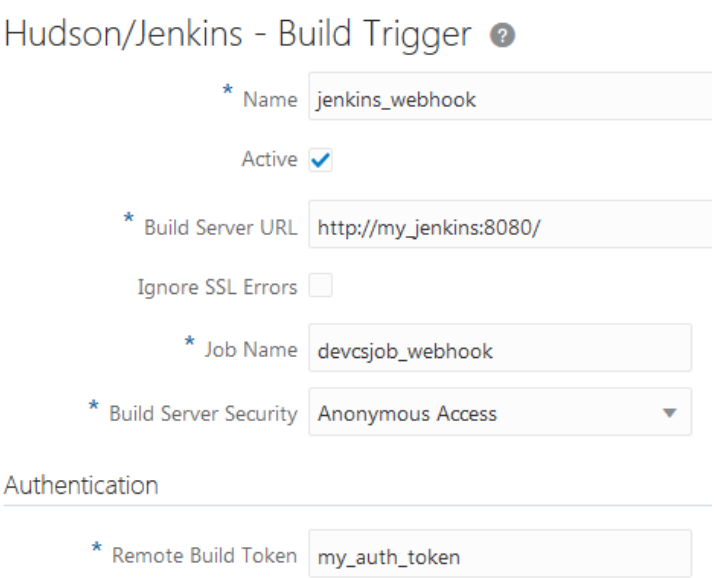
8. When you're finished, click **Save**.

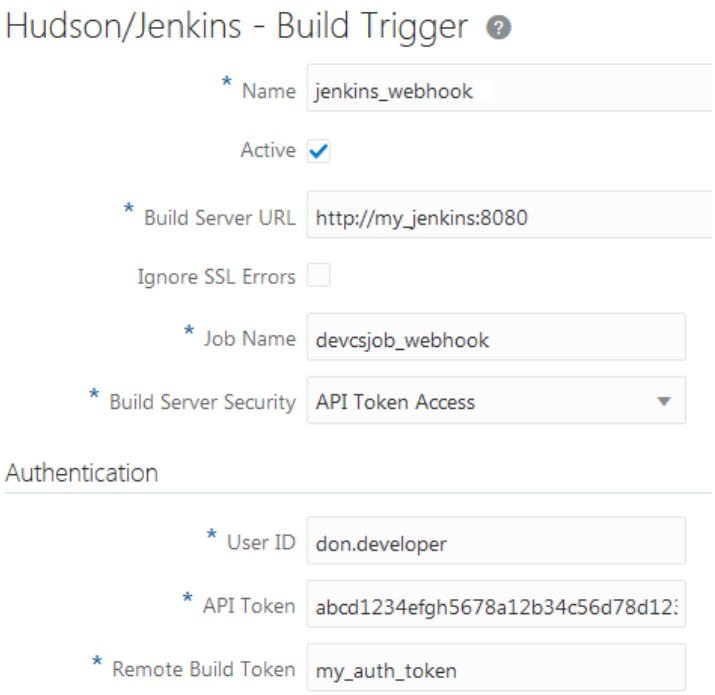
Configure a Webhook in DevCS to Trigger a Jenkins Job on a Git Repository Update

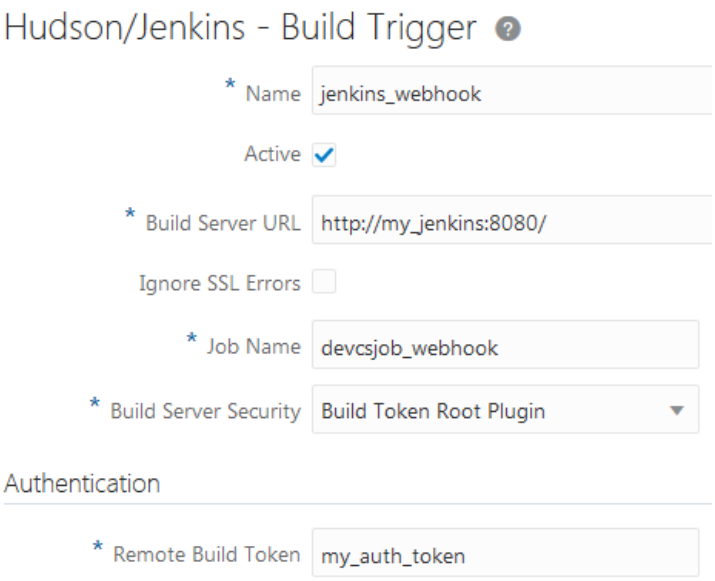
Before you create the webhook, ensure that you have installed the required plugins and have the required token to access Jenkins through the webhook.

1. In the navigation bar, click **Project Administration** .
2. Click **Webhooks**.
3. Click **+ Create Webhook**.
4. From **Type**, select **Hudson/Jenkins - Build Trigger**.
5. In **Name**, enter a unique name.
6. In **Build Server URL**, enter the Jenkins base URL.

   If the Jenkins job URL is `http://my_jenkins/path/job/my_job`, then enter `http://my_jenkins/path/`.
7. To ignore SSL errors if Jenkins uses self-signed (or an invalid) certificate and you've provided an HTTPS URL in **Build Server URL**, select the **Ignore SSL Errors** check box.
8. In **Job Name**, enter the case sensitive name of the job on the target build server.
9. From **Build Server Security**, select the security schema of Jenkins and enter the required details.

Security Option	Fill in these fields
Anonymous Access	Under **Authentication**, in **Remote Build Token**, enter the Jenkins authentication token. Example: 

Security Option	Fill in these fields
API Token Access	Under **Authentication**, enter the authenticated user's details. • In **User ID**, enter the username of the Jenkins user. • In **API Token**, enter the API token of the Jenkins user. • In **Remote Build Token**, enter the Jenkins authentication token. Example: 

Security Option	Fill in these fields
Build Token Root Plugin	Under **Authentication**, in **Remote Build Token**, enter the Jenkins authentication token. Example: 
No Security	NA

10. In **Trigger Event: Git Push**, specify the Git repository and the branch or tag.

    Select the **Parametrized Build** check box if the build job on target server accepts parameters. The target URL differs for parametrized and non-parametrized builds.

    If the **Parametrized Build** is enabled, you can add build parameters using **Add Parameter**. For each parameter, set the name that must match the parameter name defined on build server side.

11. Verify the URL displayed in **Target URL**.

    You can use the URL to check your configuration (for example using `curl -X GET '<Target_URL>'`).

12. Click **Done**.

Trigger a Jenkins Job from a Merge Request

Using the Jenkins - Merge Requests webhook, you can link a Jenkins job to a merge request. When a commit is pushed to the review branch of the merge request, the webhook sends a notification to Jenkins and triggers a build of the linked job. When the build completes, it sends a notification back to DevCS. Based on the build's status, the linked build approves or rejects the merge request.

The Jenkins Merge Request is an outgoing as well as an incoming webhook. The Jenkins job and the webhook must use the merge request's Git repository with parameters to define the branch. The Notification plugin must also be installed on Jenkins.

To allow the webhook to connect to Jenkins, you'd need to specify the security settings of Jenkins.

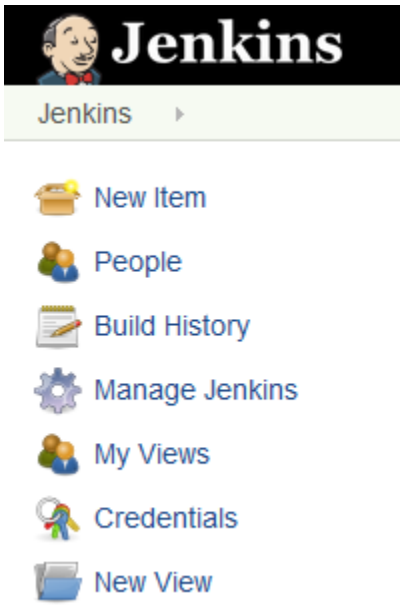
If ...	Do this:
Jenkins allows anonymous user to trigger a build on Jenkins	1. Create an authentication token in the Jenkins job. 2. Configure the webhook to connect to the Jenkins job using the authentication token.
Jenkins allows only authenticated users to trigger a build	1. Get an authenticated user's API Access token. 2. Create an authentication token in the Jenkins job. 3. Configure the webhook to connect to the Jenkins job using the API Access and the authentication token.
Anonymous access is disabled or lacks read permissions on Jenkins and you want to trigger the job without an authenticated user's credentials Or Jenkins uses a build token root to trigger builds	1. If not installed, install the Build Authorization Token Root Plugin on Jenkins. 2. Create an authentication token in the Jenkins job. 3. Configure the webhook to connect to Jenkins job using the authentication token.
Security is completely disabled on Jenkins.	Configure the webhook to connect to Jenkins job. No Jenkins configuration required.

Install the Notification Plugin on Jenkins

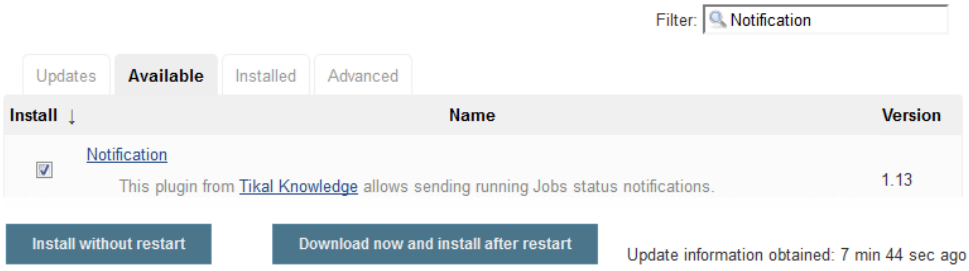
To send notifications from Jenkins, install the Notification plugin.

You must be assigned the Admin role of the Jenkins server to install plugins.

1. Log on to Jenkins using the administrator credentials.
2. From the links on the left side of the page, click **Manage Jenkins**.



3. Click **Manage Plugins**.
4. In the **Available** tab, search for `Notification`, select its check box, and click **Download now and install after restart** or **Install without restart**.

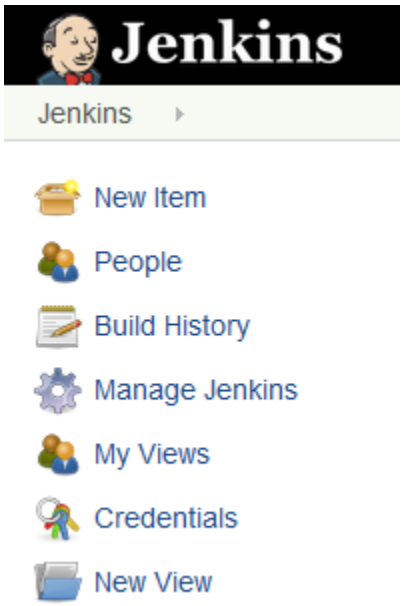


5. Wait for the plugin to install.
6. Restart Jenkins.

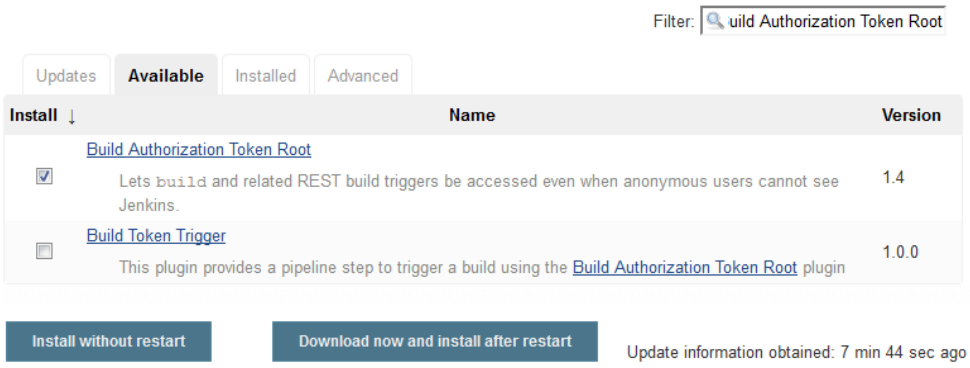
Install the Build Authorization Token Root Plugin on Jenkins

If anonymous access is disabled on Jenkins and you want to trigger Jenkins jobs without an authenticated user's credentials, install the Build Authorization Token Root plugin on Jenkins. You must be assigned the Admin role of Jenkins to install the plugin. The plugin is required To find out more about the plugin, see `https://wiki.jenkins-ci.org/display/JENKINS/Build+Token+Root+Plugin`.

1. Log on to Jenkins using the administrator credentials.
2. From the links on the left side of the page, click **Manage Jenkins**.



3. Click **Manage Plugins**.
4. In the **Available** tab, search for `Build Authorization Token Root`, select its check box, and click **Download now and install after restart** or **Install without restart**.



5. Wait for the plugin to install.
6. Restart Jenkins.

Get the Jenkins API Access Token

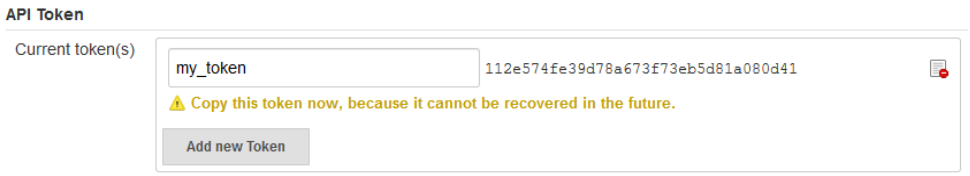
If Jenkins allows only authenticated users to trigger builds, use the API Access token of an authenticated user as the user's credentials in the DevCS webhook.

To use the API Access token in a DevCS webhook, provide the username and the token of an authenticated user. If you don't want to provide a user's details, create a separate username to trigger builds and assign the user the **Overall/Read**, **Job/Read** and **Job/Build** permissions. Then, use this user's details in the webhook.

1. Log on to Jenkins using the user's credentials whose API Access Token you want to use in the webhook.

2. In the upper-right corner, mouse over the user name, click ▾ and select **Configure**.
3. In the **API Token** section, add a new token or use the legacy token.

   To view the legacy token, click **Show Legacy API Token** and then copy the token. Keep the token value some place safe as you'd need to enter in the DevCS webhook.
   To create a token, click **Add new token** and copy the token value immediately. You can't see the token value later and would need to generate another token. Keep the token value some place safe as you'd need to enter in the DevCS webhook.

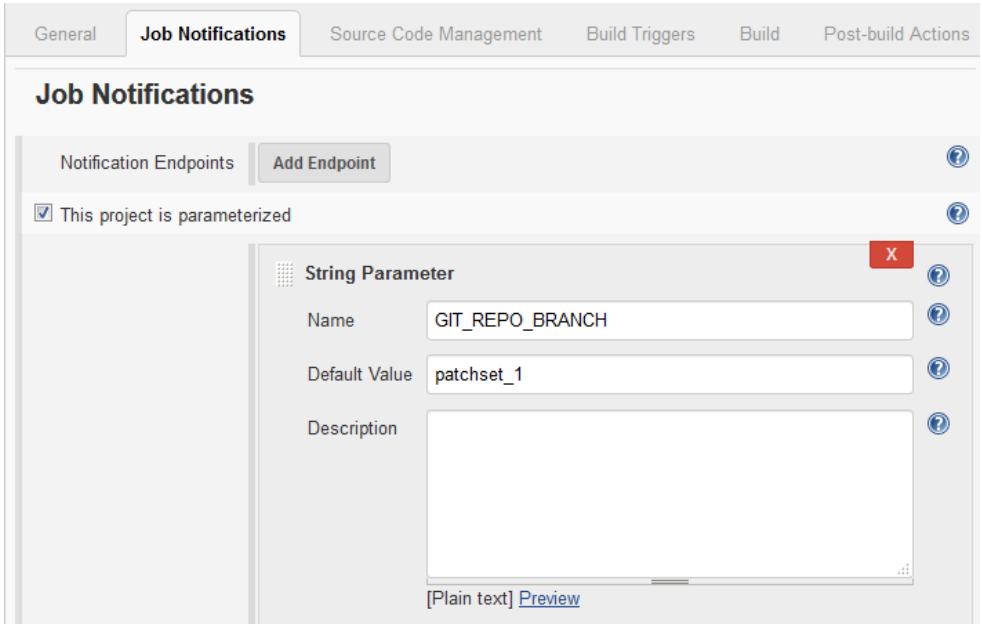


4. Click **Save**.

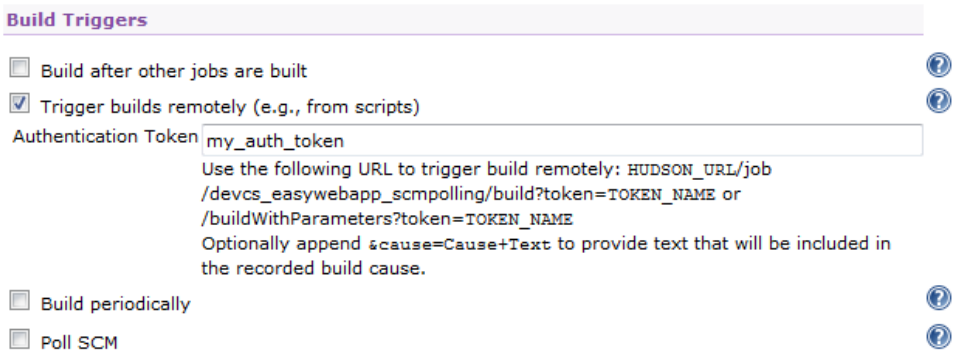
Configure the Jenkins Job to Set an Authentication Token and Accept Build Parameters

To trigger the Jenkins job when it receives a notification from DevCS, configure it to accept the Git repository's branch name as a parameter and set an authentication token.

1. Log on to Jenkins.
2. Create or open the job.
3. On the left side of the page, click **Configure**.
4. Click the **Job Notifications** tab.
5. Select the **This project is parameterized** check box.
6. From **Add Parameter**, select **String Parameter**.
7. In **Name**, enter `GIT_REPO_BRANCH`.
8. In **Default Value**, enter the review branch name. Example: `patchset_1`



9. Click the **Build Triggers** tab.
10. Select the **Trigger builds remotely (e.g., from scripts)** check box.



11. Enter a unique string as a token. You can enter any string value. Example: `my_auth_token`

    Make sure that the authentication token isn't used in any other job.
12. Continue to configure the job.
13. When you're finished, click **Save**.

Configure a Webhook in DevCS to Trigger a Jenkins Job on a Merge Request Update

After installing the required plugins and configuring the Jenkins job, create the webhook.

1. In the navigation bar, click **Project Administration** .
2. Click **Webhooks**.
3. Click **+ Create Webhook**.
4. From **Type**, select **Jenkins - Merge Requests**.

5. In **Name**, enter a unique name.
6. In **Build Server URL**, enter the Jenkins base URL.

   If the Jenkins job URL is `http://my_jenkins/path/job/my_job`, then enter `http://my_jenkins/path/`.
7. To ignore SSL errors if Jenkins uses self-signed (or an invalid) certificate and you've provided an HTTPS URL in **Build Server URL**, select the **Ignore SSL Errors** check box.
8. In **Job Name**, enter the case sensitive name of the job on the target build server.
9. In **Repository**, select the merge request's Git repository.
10. From **Build Server Security**, select the security schema of Jenkins and enter the required details.

Security Option	Fill in these fields
Anonymous Access	Under **Authentication**, in **Remote Build Token**, enter the Jenkins authentication token.
API Token Access	Under **Authentication**, enter the authenticated user's details. • In **User ID**, enter the username of the Jenkins user. • In **API Token**, enter the API token of the Jenkins user. • In **Remote Build Token**, enter the Jenkins authentication token.
Build Token Root Plugin	Under **Authentication**, in **Remote Build Token**, enter the Jenkins authentication token.
No Security	NA

11. Click **Done**.

Link the Jenkins Job with the Merge Request

1. In the navigation bar, click **Merge Request**.
2. Open the merge request.
3. Click the **Linked Builds** tab.

   The tab displays linked jobs, if any.
4. In **Search and Link Build Jobs**, enter the Jenkins job name and select it from the list.
5. Click **Save** .

When a commit is pushed to the review branch of the merge request, the webhook triggers a build of the specified job on the remote Jenkins server and notification is posted on the Recent Activity Feed of the project. If the build succeeds, it's added to the Approve section of the Review Status list in the Merge Request page. If the build fails, it's added to the Reject section of the Review Status list.

Receive Build Notifications from a Jenkins Job

Using the Jenkins - Notification Plugin Webhook, you can configure DevCS to accept build notifications from Jenkins and shows the build notification in the recent activities feed of the **Project Home** page.

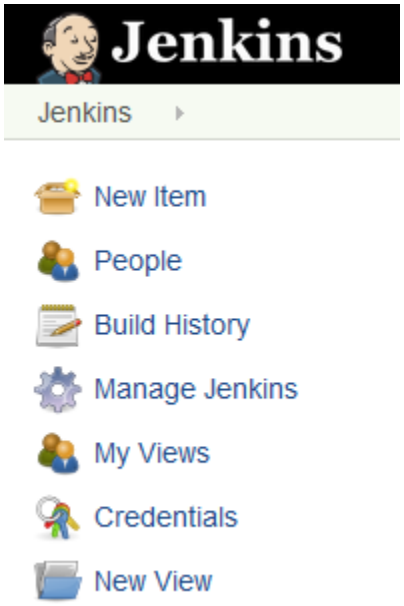
Jenkins - Notification Plugin Webhook is an incoming Webhook and accepts build notifications only. Don't use this webhook to pass information to any external server or accept information of any other type. To use the webhook, install the Notifications plugin on Jenkins, configure the DevCS webhook to connect to Jenkins, and then configure the Jenkins job to send build notifications.

Install the Notification Plugin on Jenkins

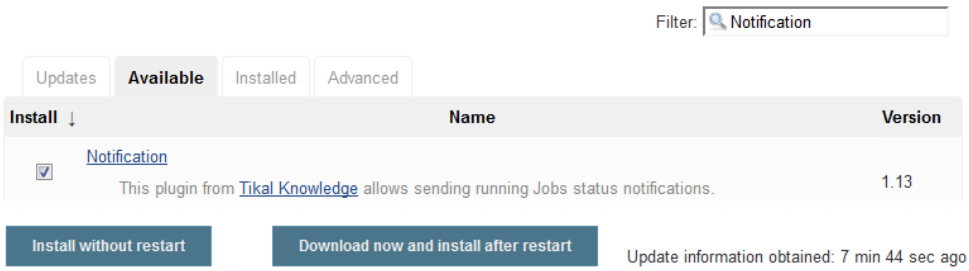
To send notifications from Jenkins, install the Notification plugin.

You must be assigned the Admin role of the Jenkins server to install plugins.

1. Log on to Jenkins using the administrator credentials.
2. From the links on the left side of the page, click **Manage Jenkins**.



3. Click **Manage Plugins**.
4. In the **Available** tab, search for `Notification`, select its check box, and click **Download now and install after restart** or **Install without restart**.

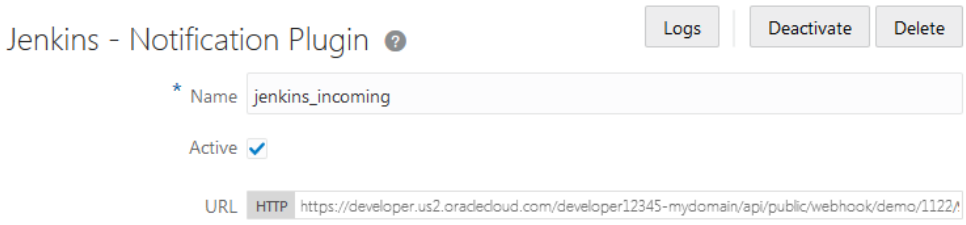


5. Wait for the plugin to install.
6. Restart Jenkins.

Configure a Webhook in DevCS to Accept Notifications from Jenkins

1. In the navigation bar, click **Project Administration** .
2. Click **Webhooks**.
3. Click **+ Create Webhook**.
4. From **Type**, select **Jenkins - Notification Plugin**.
5. In **Name**, enter a unique name.
6. In **Base URL**, enter the base URL of Jenkins.

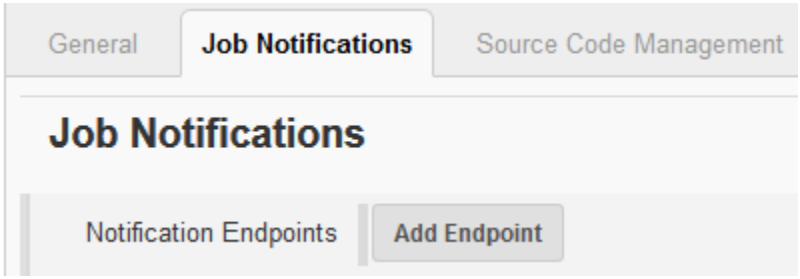
   If the Jenkins job URL is `http://my_jenkins/path/job/my_job`, then enter `http://my_jenkins/path/`.
7. In **Track**, select check boxes of build job actions to be listed in the Recent Activities Feed of the **Project Home** page.
   - To display an activity after the build server job is finished, select the **Build Results** check box.
   - To display an activity of running builds, select the **Ongoing Builds** check box.
8. Click **Done**.
9. On the Webhooks page, from the webhooks list, select the webhook. From the details displayed on the right, copy the value of **URL**.



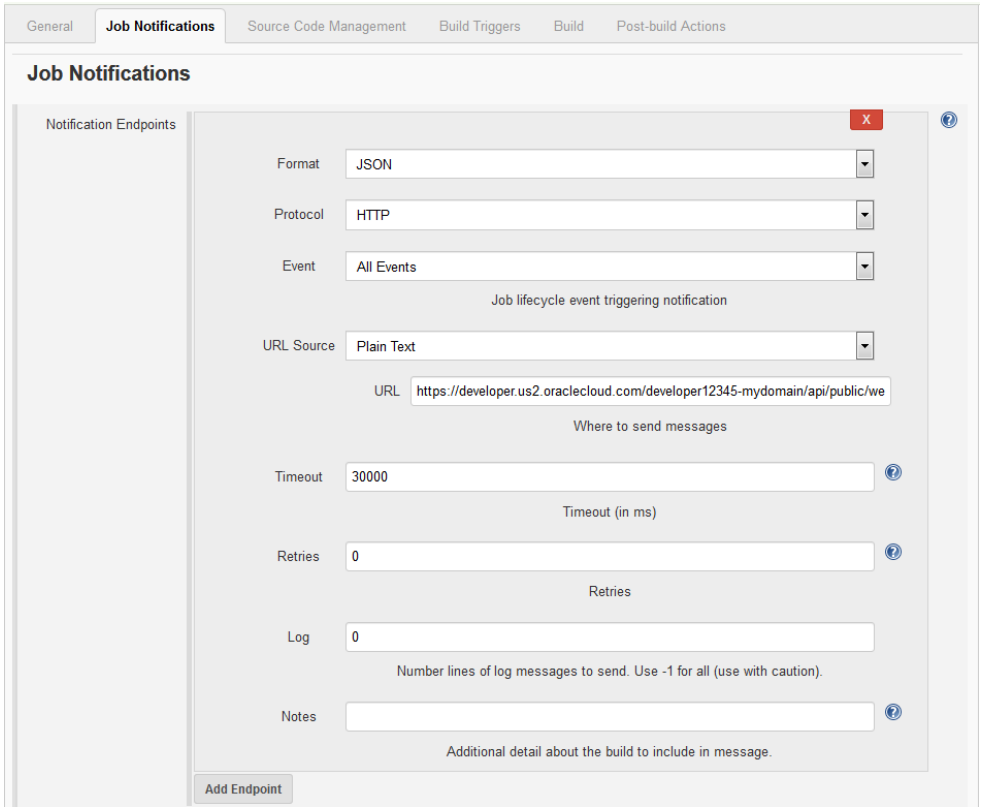
Configure the Jenkins Job to Send Build Notifications

To configure the Jenkins job to send build notifications, add the DevCS webhook's URL as a notification endpoint URL.

1. Log on to Jenkins.
2. Click the job name.
3. From the links on the left side of the page, click **Configure**.
4. Click the **Job Notifications** tab.
5. In **Notification Endpoints**, click **Add Endpoint**.



6. In the **URL** field, paste the URL that you copied from the DevCS webhook. Leave other fields with their default values.



7. Click **Save**.

Hudson

Hudson is an open-source extensible continuous integration software used to build and test your software applications. Using webhooks, you can integrate your Hudson server with DevCS to run builds. Hudson must be available on the public Internet to accept webhook notifications.

You can use these webhooks to integrate Hudson with DevCS:

To do this ...	Use this webhook
Trigger a Hudson job on SCM polling of the job's Git repository	Hudson/Jenkins Git Plugin

To do this ...	Use this webhook
Trigger a Hudson job on a project's Git repository update	Hudson/Jenkins Build Trigger

Trigger a Hudson Job on SCM Polling

Using the Hudson/Jenkins - Git Plugin Webhook, you can trigger a Hudson job that uses a DevCS Git repository as source on SCM polling.

To trigger the job:

- If not installed, install the Git plugin
- Create or configure the Hudson job to use the DevCS project Git repository as source
- Enable SCM polling in the Hudson job
- Create or configure a webhook to send a notification to Hudson when the job's Git repository (or any project Git repository) is updated

When the Git plugin of Hudson receives a notification, it goes through all Hudson jobs that have SCM polling enabled and match the provided notification parameters (such as Git repositories and branches). For all matching jobs, it starts a build. The build won't run if no changes are found by polling.

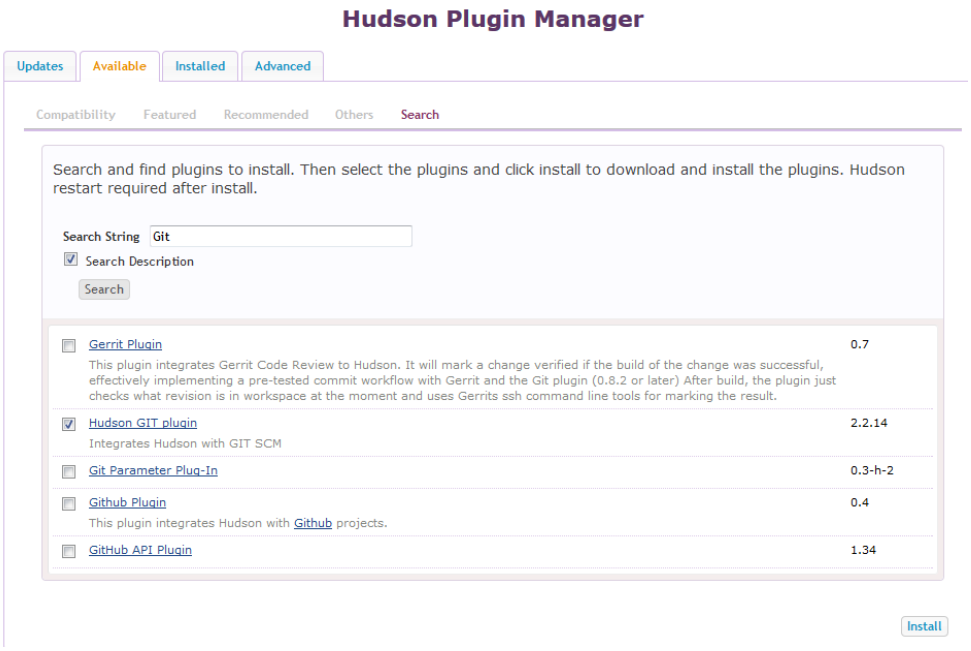
For more information about the Hudson Git plugin, see `http://wiki.hudson-ci.org/display/HUDSON/Git+Plugin#GitPlugin-PostCommitHook`.

Set Up Git on Hudson

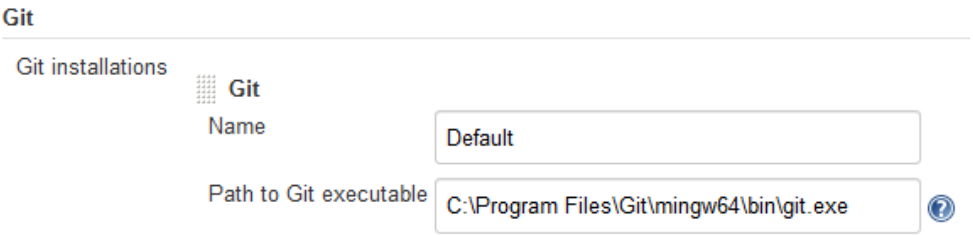
You must be assigned the Admin role of Hudson to set up Git on it. Git must also be installed on the computer running Jenkins. If the plugin is already installed and configured, ignore this section.

1. Log on to Hudson using the administrator credentials.
2. From the links on the left side of the page, click **Manage Hudson**.

3. To install the Git plugin, click **Manage Plugins**.
4. In the **Available** tab, click the **Search** subtab, search for `Git`, select the **Hudson GIT plugin** check box, and click **Install**.



5. Wait for the plugin to install.
6. Restart Hudson.
7. From the links on the left side of the page, click **Manage Hudson**.
8. Click **Configure System**.
9. In **Git**, enter the local path of the Git executable.



10. Click **Save**.

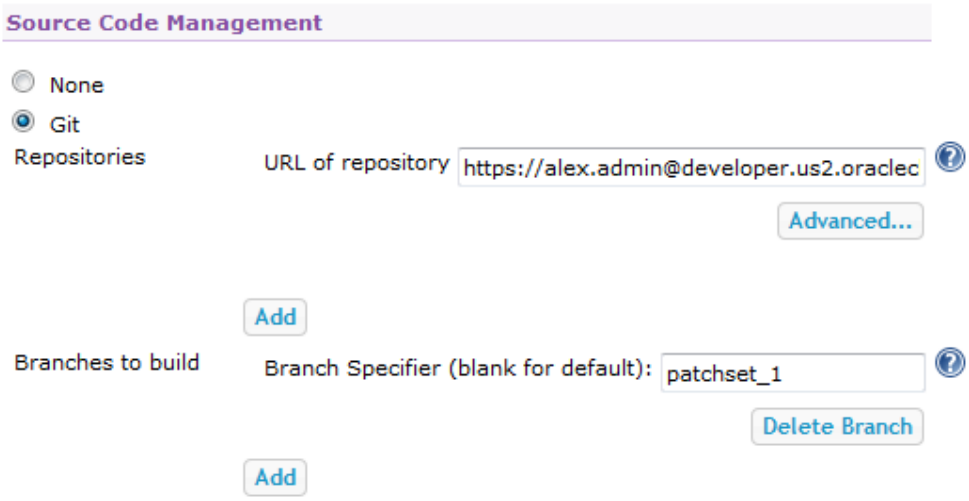
Configure the Hudson Job to Use DevCS Git Repository and Enable SCM Polling

Configure the job to access the DevCS Git repository and enable SCM polling.

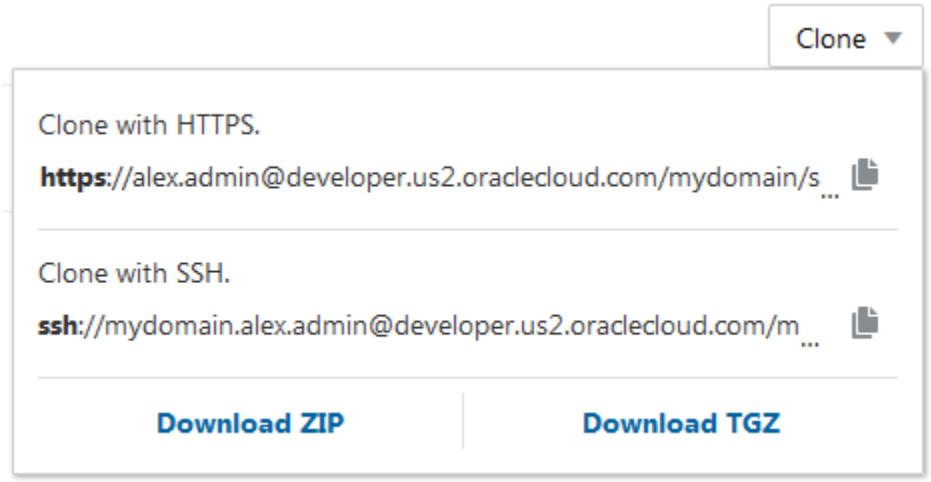
1. Log in to Hudson.
2. Create or open a job.
3. From the links on the left side of the page, click **Configure**.

4. In **Source Code Management**, select **Git**.
5. In **URL of repository**, enter the DevCS project's Git repository URL.

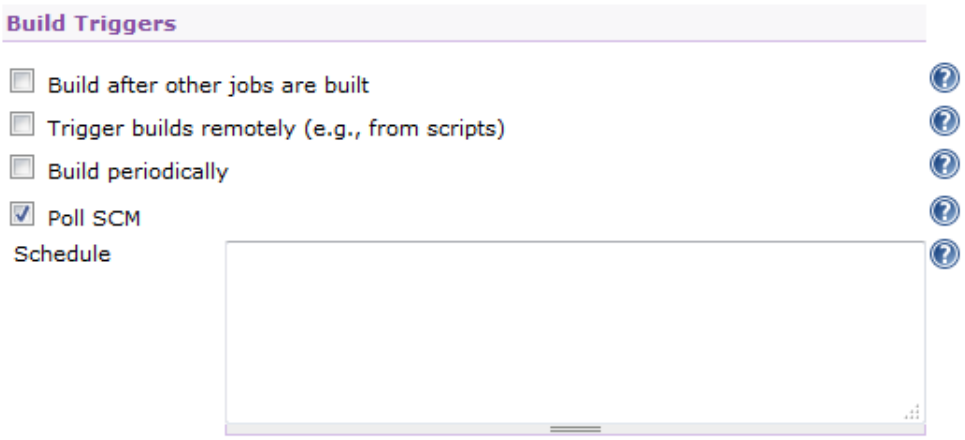
   Remember the URL's protocol as you'd need to specify it when you create the webhook.



   You can copy the URL from the **Clone** menu of the DevCS Code page.



6. In **Branches to build**, specify the branch name.
7. In the **Build Triggers** section, select the **Poll SCM** check box.



8. Continue to configure the job.
9. Click **Save**.

Configure a Webhook in DevCS to Trigger a Hudson Job on the Git Repository Update

After configuring the Hudson job, create the DevCS webhook to trigger the job on Git repository update.

1. In the navigation bar, click **Project Administration** .
2. Click **Webhooks**.
3. Click **+ Create Webhook**.
4. From **Type**, select **Hudson/Jenkins - Git Plugin**.
5. In **Name**, enter a unique name.
6. In **Notification URL**, enter the Hudson URL.

   The URL must be in the `http://your_server/.../git/notifyCommit` format. Example: `http://my_hudson.com:8080/git/notifyCommit`
7. To ignore SSL errors, select the **Ignore SSL Errors** check box.
8. In **Notification Parameters**, specify the URL type.

   In **Repository URL Type**, select **HTTP Repository Address** to send the HTTP URL of the selected Git repository in the webhook notification. Select **SSH Repository Address** to send the SSH URL of the selected Git repository in the webhook notification.

   You must specify the same protocol that's used in your the job configuration to access the Git repository.

   To append branch information of the last commit in the webhook notification, select the **branches** check box. This enables jobs to poll the specified branches only.
9. In **Repository** and **Branches**, specify the Git repository and branches that trigger the webhook.

   In **Repository**, select **All Repositories** to trigger all Hudson jobs that uses a Git repository of the project.
10. Click **Done**.

Trigger a Hudson Job on a Git Repository Update

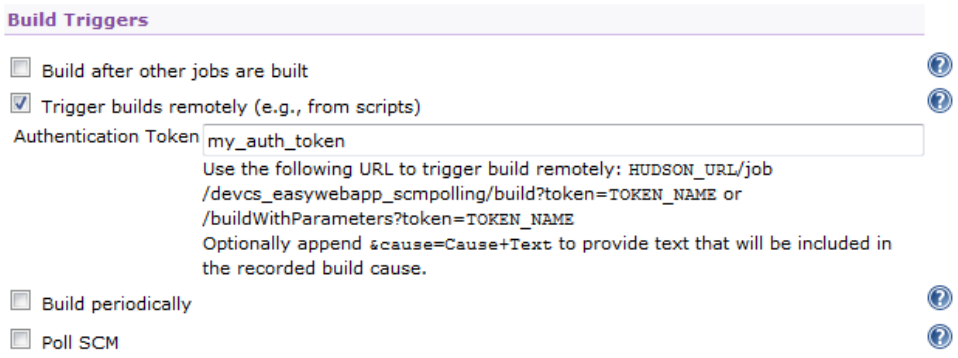
Using the Hudson/Jenkins - Build Trigger webhook, you can trigger a Hudson job when a project Git repository updates. It's not necessary that the Hudson job uses a DevCS project Git repository as source.

To allow the webhook to connect to Hudson, you'd need to specify the security settings of Hudson.

If ...	Do this:
Hudson allows anonymous user to trigger a build	1. Create an authentication token in the Hudson job. 2. Configure the webhook to connect to the Hudson job using the authentication token.
Hudson allows only authenticated users to trigger a build	1. Get an authenticated user's credentials. 2. Create an authentication token in the Hudson job. 3. Configure the webhook to connect to the Hudson job using the credentials and the authentication token.
Security is completely disabled on Hudson	Configure the webhook to connect to Hudson job. No Hudson configuration required.

Configure the Hudson Job

1. Log in to the Hudson server.
2. Click the job name.
3. From the links on the left side of the page, click **Configure**.
4. In the **Build Triggers** section, select the **Trigger builds remotely (e.g., from scripts)** check box.



5. In **Authentication Token**, enter a unique string as a token. You can enter any string value. Example: `my_auth_token`

   Make sure that the authentication token name is not used in any other job.

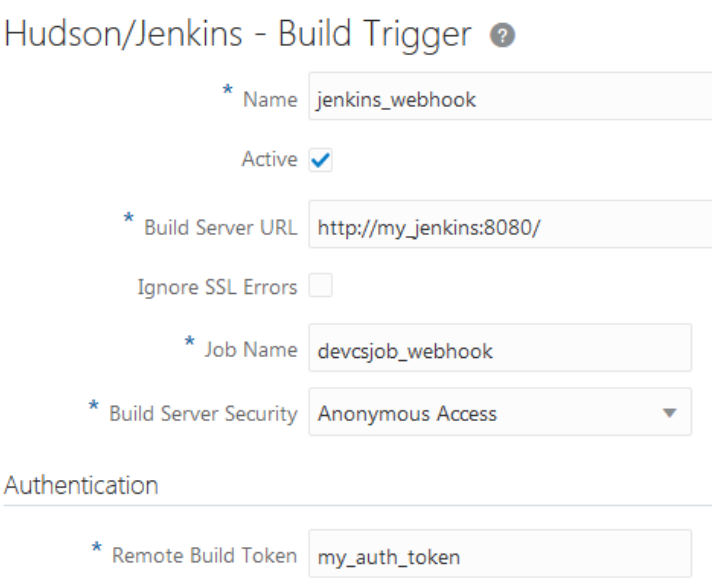
6. Click **Save**.

Configure a Webhook in DevCS to Trigger a Hudson Job on a Git Repository Update

Before you create the webhook, ensure that you have installed the required plugins and have the required token to access Hudson through the webhook.

1. In the navigation bar, click **Project Administration** .
2. Click **Webhooks**.
3. Click **+ Create Webhook**.
4. From **Type**, select **Hudson/Jenkins - Build Trigger**.
5. In **Name**, enter a unique name.
6. In **Build Server URL**, enter the Hudson URL.

   If the target build job has address `http://my_server/path/job/my_job`, then enter `http://my_hudson/path/`.
7. To ignore SSL errors if the target build server uses self-signed (or an invalid) certificate and you've provided an HTTPS URL in **Build Server URL**, select the **Ignore SSL Errors** check box.
8. In **Job Name**, enter the case sensitive name of the Hudson job.
9. From **Build Server Security**, select the job's security schema configured on the target server.

Security Option	Fill in these fields
Anonymous Access	Under **Authentication**, in **Remote Build Token**, enter the Jenkins authentication token. Example: 

Security Option	Fill in these fields
API Token Access	Under **Authentication**, enter the authenticated user's details. • In **User ID**, enter the username of the Jenkins user. • In **API Token**, enter the password of the user. • In **Remote Build Token**, enter the Hudson authentication token.
No Security	NA

10. In **Trigger Event: Git Push**, specify the Git repository and the branch or tag.

    Select the **Parametrized Build** check box if the build job on target server accepts parameters. The target URL differs for parametrized and non-parametrized builds.

    If the **Parametrized Build** is enabled, you can add build parameters using **Add Parameter**. For each parameter, set the name that must match the parameter name defined on build server side.

11. Verify the URL displayed in **Target URL**.

    You can use the URL to check your configuration (for example using `curl -X GET '<Target_URL>'`).

12. Click **Done**.

GitHub Apps

If you're using apps that accept incoming webhook connections from GitHub, use the GitHub Compatible webhook to send DevCS event notifications to those apps. The payload is sent in the similar format as the GitHub, so you don't need to make changes to your GitHub apps.

To find more about GitHub webhooks, see `https://developer.github.com/webhooks/`.

1. In the navigation bar, click **Project Administration**  .
2. Click **Webhooks**.
3. Click **+ Create Webhook**.
4. From the **Type** drop-down list, select **GitHub Compatible**.
5. In **Name**, enter a unique name.
6. In **URL**, enter the URL of the GitHub app.
7. In **Secret**, enter a secret phrase that's passed as a string with the HTTP request as a signature header.
8. From the **Payload Type** drop-down list, select the media type for the payload. You can select either `form-urlencoded` (default) or `json`.
9. To ignore the host's SSL certificate verification when delivering the HTTP request, select the **Ignore SSL Errors** check box.
10. In **Event Groups**, select the events that trigger the webhook.

    If you selected the **Select specific events** option, in **Events**, select the check boxes of events that trigger the webhook.

11. Click **Done**.

When you're finished, use the project navigation bar to switch to another page.

Send Event Notifications to Any Application

Using the DevCS Generic Webhook, you can sends event notifications to any application that accepts webhook requests and can parse payload specific content. The webhook payload format depends on the type of the event.

The generic webhook supports all possible events of DevCS, including Git pushes, issue updates, merge request updates, and project builds. It sends a POST request to the remote service in the JSON format with details of the subscribed events.

1. In the navigation bar, click **Project Administration** .
2. Click **Webhooks**.
3. Click **+ Create Webhook**.
4. From the **Type** drop-down list, select **Generic**.
5. In **Name**, enter a unique name.
6. In **URL**, enter the URL of the remote service where you want to deliver the HTTP request.
7. In **Secret**, enter a secret phrase that's passed as a string with the HTTP request as a signature header.
8. To ignore the host's SSL certificate verification when delivering the HTTP request, select the **Ignore SSL Errors** check box.
9. In **Event Groups**, select the events that triggers the webhook.

   If you selected the **Select specific events** option, in **Events**, select the check boxes of the events to trigger the webhook.
10. Click **Done**.

The newly created Webhook appears in the Webhooks table.
To find more about the data structure of a generic Webhook, see Data Structure of a Generic Webhook.
When you're finished, use the project navigation bar to switch to another page.

Data Structure of a Generic Webhook

The information sent by a generic webhook is delivered using a POST request with the `application/json` content-type, with the UTF-8 character set, in a `Message` object.

This table describes the fields of the `Message` object.

Field	Description
`apiVersion`	Version of the API. It changes when the payload format of the request changes.
`messageId`	Unique identifier of the message
`timestamp`	Timestamp of the message when it was generated
`testEvent`	Set to `true` if this event is generated by the Test button
`projectId`	Unique identifier of the project
`events`	List of events delivered by the message

Each `event` delivered by the message follows a common structure. There are three types of events (`ISSUE/PUSH/BUILD/REVIEW/ACTIVITY`).

Field	Description
`eventId`	Type of the event (`ISSUE/PUSH/BUILD/REVIEW/ACTIVITY`)
`projectId`	Unique identifier of the project
`timestamp`	Timestamp of the event
`data`	Data specific to the type of the event

The structure of `data` of each event type is described in the following sections.

ISSUE Event

The `ISSUE` event contains the fields described in this table.

Field	Description
`type`	Type of the activity (`CREATED` - issue is created, `COMMENTED` - comment added, `UPDATED` - fields changed)
`date`	Timestamp of the activity
`description`	Description of the change
`task`	Description of the issue after the change
`id`	Issue ID
`version`	Change version
`url`	URL of the issue
`title`	Title of the issue
`type`	Type of the issue (`Defect`, `Feature`, or `Task`)
`resolution`	Resolution of the issue. The value is `null` if the issue isn't resolved, otherwise, it's set to one of the issue resolution values such as `FIXED`, `DUPLICATE`, and `WORKSFORME`.
`reporter`	User who reported the issue
`asignee`	User to whom the issue is assigned
`comment`	Content of the added comment, available if the activity type is `COMMENTED`
`fieldUpdates`	List of changed fields, available if the activity type is `UPDATED`
`name`	Field name
`oldValue`	Value before the change
`newValue`	Value after the change

Here is a JSON payload example of an issue create event.

```
{
 "apiVersion": "1.0",
 "messageId": "04abc282-a44e-4c23-ba53-15b519d30066",
 "projectId": "qa-dev_example-project",
```

```
    "testEvent": false,
    "timestamp": 1417810876408,
    "events": [
        {
            "eventId": "ISSUE",
            "projectId": "example-project",
            "timestamp": 1417810876,
            "data": {
                "activities": [
                    {
                        "type": "CREATED",
                        "date": 1417810875820,
                        "description": "",
                        "author": {
                            "gravatarHash":
"8940829abebbc5d8d84e37af7161fd31",
                            "loginName": "alex.admin",
                            "realName": "Alex Admin"
                        },
                        "issue": {
                            "id": 2,
                            "resolution": null,
                            "title": "Test Issue",
                            "type": "Feature",
                            "url": "http://test-server/#projects/example-
project/task/2",
                            "version": "1417810875834",
                            "reporter": {
                                "gravatarHash":
"8940829abebbc5d8d84e37af7161fd31",
                                "loginName": "alex.admin",
                                "realName": "Alex Admin"
                            }
                        }
                    }
                ]
            }
        }
    ]
   }
```


Here is a JSON payload example of an issue update event.


```
   {
    "apiVersion": "1.0",
    "messageId": "ccce183e-097d-4668-a07b-cf762108716e",
    "projectId": "qa-dev_example-project",
    "testEvent": false,
    "timestamp": 1417811058243,
    "events": [
        {
            "eventId": "ISSUE",
            "projectId": "example-project",
            "timestamp": 1417811058
```

```
            "data": {
                "activities": [
                    {
                        "type": "UPDATED"
                        "date": 1417811057698,
                        "description": "Assign to alex.admin\nset
Resolution to FIXED\nset Status to RESOLVED\n",
                        "author": {
                            "gravatarHash":
"8940829abebbc5d8d84e37af7161fd31",
                            "loginName": "alex.admin",
                            "realName": "Alex Admin"
                        },
                        "issue": {
                            "id": 2,
                            "resolution": "FIXED",
                            "title": "Test Issue",
                            "type": "Feature",
                            "url": "http://test-server/#projects/example-
project/task/2",
                            "version": "1417811057698",
                            "asignee": {
                                "gravatarHash":
"8940829abebbc5d8d84e37af7161fd31",
                                "loginName": "alex.admin",
                                "realName": "Alex Admin"
                            },
                            "reporter": {
                                "gravatarHash":
"8940829abebbc5d8d84e37af7161fd31",
                                "loginName": "alex.admin",
                                "realName": "Alex Admin"
                            }
                        },
                        "fieldUpdates": [
                            {
                                "name": "assigned_to",
                                "newValue": "alex.admin",
                                "oldValue": ""
                            },
                            {
                                "name": "resolution",
                                "newValue": "FIXED",
                                "oldValue": ""
                            },
                            {
                                "name": "bug_status",
                                "newValue": "RESOLVED",
                                "oldValue": "UNCONFIRMED"
                            }
                        ]
                    },
                    {
                        "type": "COMMENTED"
                        "date": 1417811057929,
```

```
                        "description": "Feature is implemented",
                        "author": {
                            "gravatarHash":
"8940829abebbc5d8d84e37af7161fd31",
                            "loginName": "alex.admin",
                            "realName": "Alex Admin"
                        },
                        "comment": {
                            "author": {
                                "gravatarHash":
"8940829abebbc5d8d84e37af7161fd31",
                                "loginName": "alex.admin",
                                "realName": "Alex Admin"
                            },
                            "date": 1417811057929,
                            "text": "Feature is implemented",
                            "type": "UNKNOWN"
                        },
                        "task": {
                            "id": 2,
                            "resolution": "FIXED",
                            "title": "Test Issue",
                            "type": "Feature",
                            "url": "http://test-server/#projects/qa-
dev_example-project/task/2",
                            "version": "1417811057698",
                            "asignee": {
                                "gravatarHash":
"8940829abebbc5d8d84e37af7161fd31",
                                "loginName": "alex.admin",
                                "realName": "Alex Alex Admin"
                            },
                            "reporter": {
                                "gravatarHash":
"8940829abebbc5d8d84e37af7161fd31",
                                "loginName": "alex.admin",
                                "realName": "Alex Admin"
                            }
                        }
                    }
                ]
            }
        }
    ]
  }
```


PUSH Event

The `PUSH` event contains the fields described in this table.

Field	Description
`refName`	Updated references
`commits`	Commits of the Push event

Field	Description
sha	Commit identifier
comment	Comment in the commit
author	Author of the commit
date	Timestamp of the commit
parents	List of commit parent identifiers
repository	Name of the repository to which the commit was pushed

Here is a JSON payload example of a Git Push event.

```
 {
   "apiVersion": "1.0",
   "messageId": "c3378be6-6be5-4191-9b20-1fb5d429bfce",
   "projectId": "example-project",
   "testEvent": false,
   "timestamp": 1417810424512,
   "events": [
       {
           "eventId": "GIT_PUSH",
           "projectId": "example-project",
           "timestamp": 1417810424,
           "data": {
               "refName": "refs/heads/master",
               "commits": [
                   {
                       "sha": "32e03bc46a3a42eeab5dd25144a90c5b4f0b2e11",
                       "repository": "example-project.git",
                       "date": 1417810387000,
                       "comment": "file1.txt deleted, file3.txt created
\n",
                       "author": {
                           "email": "alex.admin@example.com",
                           "firstName": "Alex",
                           "lastName": "Admin",
                           "username": "alex.admin"
                       },
                       "parents": [
                           "1106e8c81cb49e71024e9017235f89dc3983d4ee"
                       ]
                   },
                   {
                       "sha": "1106e8c81cb49e71024e9017235f89dc3983d4ee",
                       "repository": "example-project.git",
                       "date": 1417810290000,
                       "comment": "file2.txt updated\n",
                       "author": {
                           "email": "alex.admin@example.com",
                           "firstName": "Alex",
                           "lastName": "Admin",
                           "username": "alex.admin"
```

```
                    },
                    "parents": [
                        "8dab56fb6ba6dd0fc0d0aa8c7ce4f01d77fa0835"
                    ]
                }
            ]
        }
    }
  ]
}
```

BUILD Event

The `BUILD` event contains the fields described in this table.

Field	Description
`jobName`	Name of the job
`timestamp`	Build timestamp
`number`	Build number
`url`	Build URL
`result`	Build result (SUCCESS/UNSTABLE/FAILURE/NOT_BUILT/ABORTED)
`duration`	Build duration
`fileName`	Name of the artifact
`relativePath`	Path relative to the job workspace
`url`	URL of the artifact

Here is a JSON payload example of a Build event.

```
{
 "apiVersion":"1.0",
 "messageId":"4a253425-4598-4838-a4b5-aac30d0b9710",
 "timestamp":1417795613257,
 "testEvent":true,
 "projectId":"test-project",
 "events":[
    {
       "eventId":"BUILD",
       "projectId":"test-project",
       "timestamp":1417795613256,
       "data":{
          "jobName":"example-job",
          "details":{
             "timestamp":1417795590256,
             "number":16,
             "url":"http://server/test-dev/s2/test-project/hudson/job/
test-project.example-job/16/",
             "result":"SUCCESS",
             "duration":36905,
             "artifacts":[
                {
```

```
                    "fileName":"sample-1.0-SNAPSHOT.jar",
                    "relativePath":"sample-project/target/sample-1.0-
SNAPSHOT.jar",
                    "url":"http://server/test-dev/s2/test-project/
hudson/job/test-project.example-job/16/artifact/sample-project/target/
sample-1.0-SNAPSHOT.jar"
                  }
               ]
            }
         }
      }
   ]
  }
```

REVIEW Event

The `REVIEW` event represents changes in merge requests and contains the fields described in this table.

Field	Description
`review`	Description of the merge request
`id`	Unique ID of the merge request
`title`	Title of the merge request
`created`	Timestamp of the merge request creation
`modified`	Timestamp of the merge request last modification
`reporter`	Profile of the user who created the merge request
`repository`	Name of the Git repository
`reviewBranch`	Name of the review branch
`targetBranch`	Name of the target branch
`user`	Profile of the user who performed the action
`action`	Merge request action Here is the list of merge request actions: • `CREATED`: Merge request is created • `COMMIT`: New commits are pushed to the review branch • `MERGED`: Review branch is merged into the target branch The `MERGED` action is created if the review branch is merged via the Merge button in the web user interface. If the review branch is merged from a Git client (such as the Git command line interface), no action is generated. • `REVIEWED`: Reviewer approves or rejects a merge request • `COMMENTED`: A comment is added to the merge request • `CLOSED`: Merge request is closed

Field	Description
commits	List of commits added to the merge request The commits field is generated for the COMMIT action. These fields are also generated for the commits action: • author: Author of the commit • message: Commit message • sha: SHA-1 checksum hash of the commit
text	Text of the comment The text field is generated for the COMMENTED action.
comment	Comment of the rejected or approved review action The comment field is generated for the REVIEWED action.
result	Result of the merge (FAST_FORWARD, FAST_FORWARD_SQUASHED, ALREADY_UP_TO_DATE, FAILED, MERGED, MERGED_SQUASHED, MERGED_SQUASHED_NOT_COMMITTED, CONFLICTING, ABORTED, MERGED_NOT_COMMITTED, NOT_SUPPORTED, CHECKOUT_CONFLICT) The result field is generated for the MERGED action.
status	Status of the merge request (APPROVED, REJECTED, COMPLETED, CANCELLED) The status field is generated for the REVIEWED and the CLOSED action.

Here is a JSON payload example of a REVIEW event.

```
{
    "apiVersion": "1.0",
    "events": [
        {
            "data": {
                "action": "CREATED",
                "review": {
                    "created": 1431944319181,
                    "id": 6,
                    "modified": 1431944319635,
                    "reporter": {
                        "email": "alex.admin@example.com",
                        "firstName": "Alex",
                        "lastName": "Alex Admin",
                        "username": "alex.admin"
                    },
                    "repository": "example-project.git",
                    "reviewBranch": "bug_branch",
                    "targetBranch": "master",
                    "title": "Bug Fix"
                },
                "user": {
                    "email": "alex.admin@example.com",
                    "firstName": "Alex",
                    "lastName": "Alex Admin",
                    "username": "alex.admin"
```

```
                }
            },
            "eventId": "REVIEW",
            "projectId": "example-project",
            "timestamp": 1431944327
        }
    ],
    "messageId": "08758261-e4e7-4c8f-b9fe-7b74f715803f",
    "projectId": "example-project",
    "testEvent": false,
    "timestamp": 1431944329923
}


{
    "apiVersion": "1.0",
    "events": [
        {
            "data": {
                "action": "COMMIT",
                "commits": [
                    {
                        "author": "alex.admin",
                        "message": "fix version #3\n",
                        "sha": "8fd1d2a53a181aa7015e7535b6f64295c432eca7"
                    },
                    {
                        "author": "alex.admin",
                        "message": "fix version #2\n",
                        "sha": "ff2bdf91d0fb6fb664315879ec38acc0931beeb6"
                    }
                ],
                "review": {
                    "created": 1431944319181,
                    "id": 6,
                    "modified": 1431944340209,
                    "reporter": {
                        "email": "alex.admin@example.com",
                        "firstName": "Alex",
                        "lastName": "Alex Admin",
                        "username": "alex.admin"
                    },
                    "repository": "example-project.git",
                    "reviewBranch": "bug_branch",
                    "targetBranch": "master",
                    "title": "Bug Fix"
                },
                "user": {
                    "email": "alex.admin@example.com",
                    "firstName": "Alex",
                    "lastName": "Alex Admin",
                    "username": "alex.admin"
                }
            },
            "eventId": "REVIEW",
```

```
            "projectId": "example-project",
            "timestamp": 1431944353
        }
    ],
    "messageId": "5de98d08-49cd-4a19-86b5-d89757f75a1d",
    "projectId": "example-project",
    "testEvent": false,
    "timestamp": 1431944355646
}

{
    "apiVersion": "1.0",
    "events": [
        {
            "data": {
                "user": {
                    "email": "clara.coder@example.com",
                    "firstName": "Clara",
                    "lastName": "Coder",
                    "username": "clara"
                },
                "review": {
                    "created": 1436521285722,
                    "id": 23,
                    "modified": 1438246154916,
                    "reporter": {
                        "email": "alex.admin@example.com",
                        "firstName": "Alex",
                        "lastName": "Admin",
                        "username": "alex"
                    },
                    "repository": "example-project.git",
                    "reviewBranch": "bug_branch",
                    "targetBranch": "master",
                    "title": "Some Review"
                },
                "action": "REVIEWED",
                "status": "REJECTED",
                "comment": "rejected the request because ...",
            },
            "eventId": "REVIEW",
            "projectId": "example-project",
            "timestamp": 1438246163
        }
    ],
    "messageId": "f0a75815-3470-4dc4-be82-975935152ed3",
    "projectId": "example-project",
    "testEvent": false,
    "timestamp": 1438246165924
}

{
    "apiVersion": "1.0",
    "events": [
        {
```

```
            "data": {
                "action": "COMMENTED",
                "review": {
                    "created": 1431944319181,
                    "id": 6,
                    "modified": 1431944478701,
                    "reporter": {
                        "email": "alex.admin@example.com",
                        "firstName": "Alex",
                        "lastName": "Alex Admin",
                        "username": "alex.admin"
                    },
                    "repository": "example-project.git",
                    "reviewBranch": "bug_branch",
                    "targetBranch": "master",
                    "title": "Bug Fix"
                },
                "text": "General comment",
                "user": {
                    "email": "alex.admin@example.com",
                    "firstName": "Alex",
                    "lastName": "Alex Admin",
                    "username": "alex.admin"
                }
            },
            "eventId": "REVIEW",
            "projectId": "example-project",
            "timestamp": 1431945965
        }
    ],
    "messageId": "d2a36692-dae6-44d4-a112-7a615b524cc3",
    "projectId": "example-project",
    "testEvent": false,
    "timestamp": 1431945967166
}

{
    "apiVersion": "1.0",
    "events": [
        {
            "data": {
                "action": "MERGED",
                "result": "FAST_FORWARD",
                "review": {
                    "created": 1431944319181,
                    "id": 6,
                    "modified": 1431944478701,
                    "reporter": {
                        "email": "alex.admin@example.com",
                        "firstName": "Alex",
                        "lastName": "Alex Admin",
                        "username": "alex.admin"
                    },
                    "repository": "example-project.git",
                    "reviewBranch": "bug_branch",
```

```
                    "targetBranch": "master",
                    "title": "Bug Fix"
                },
                "user": {
                    "email": "alex.admin@example.com",
                    "firstName": "Alex",
                    "lastName": "Alex Admin",
                    "username": "alex.admin"
                }
            },
            "eventId": "REVIEW",
            "projectId": "example-project",
            "timestamp": 1431945438
        }
    ],
    "messageId": "b06d5581-d38a-4972-9c80-dc1455547776",
    "projectId": "example-project",
    "testEvent": false,
    "timestamp": 1431945440287
}

{
    "apiVersion": "1.0",
    "events": [
        {
            "data": {
                "action": "CLOSED",
                "review": {
                    "created": 1431944319181,
                    "id": 6,
                    "modified": 1431945453967,
                    "reporter": {
                        "email": "alex.admin@example.com",
                        "firstName": "Alex",
                        "lastName": "Alex Admin",
                        "username": "alex.admin"
                    },
                    "repository": "example-project.git",
                    "reviewBranch": "bug_branch",
                    "targetBranch": "master",
                    "title": "Bug Fix"
                },
                "status": "COMPLETED",
                "user": {
                    "email": "alex.admin@example.com",
                    "firstName": "Alex",
                    "lastName": "Alex Admin",
                    "username": "alex.admin"
                }
            },
            "eventId": "REVIEW",
            "projectId": "example-project",
            "timestamp": 1431945459
        }
    ],
```

```
    "messageId": "b434f6d2-b5c7-4c0a-bab2-3e6614025865",
    "projectId": "example-project",
    "testEvent": false,
    "timestamp": 1431945453967
}
```

ACTIVITY Event

The `ACTIVITY` event contains the fields described in this table.

Field	Description
`author`	Profile of the user whose action produced the activity The value is `null` for system activities.
`name`	Name of the activity
`properties`	Description of the activity, or the object whose fields depends on the `name` field. Here is the list of supported activities: • `BUILD`: Triggered when a build in the integrated Hudson server ends. • `DEPLOYMENT`: Triggered when the application is deployed, undeployed, started, or stopped using Deploy page in the web user interface. • `MEMBER`: Triggered when a user is added, removed, or role is updated. • `REVIEW`: Triggered when a merge request is created, closed, or updated. • `RSS`: Triggered when a new article is acquired from a registered feed. • `SCM_COMMIT`: Triggered when a commit is pushed to a project repository. • `SCM_REPO`: Triggered when a project repository is added or removed. • `TASK`: Triggered when an issue is created or updated. • `WIKI`: Triggered when a wiki page is created or updated.

Here is a JSON payload example of an Activity event.

```
{
    "apiVersion": "1.0",
    "events": [
        {
            "data": {
                "author": {
                    "email": "alex.admin@example.com",
                    "firstName": "Alex",
                    "lastName": "Alex Admin",
                    "username": "alex.admin"
                },
                "name": "WIKI",
                "properties": {
                    "page": "New Page Title",
```

```
                    "type": "CREATED"
                }
            },
            "eventId": "ACTIVITY",
            "projectId": "example-project",
            "timestamp": 1432035029
        }
    ],
    "messageId": "45066d85-5a5c-4647-9a6c-43fc8e99481a",
    "projectId": "qa-dev_test-rss",
    "testEvent": false,
    "timestamp": 1432035031418
}
```


Access External Docker Registries

If you use an external Docker registry, such as DockerHub or Oracle Cloud Infrastructure Registry (OCIR), you can link the registry to your project and browse its repositories and images from Oracle Developer Cloud Service (DevCS).

A Docker Registry is a server-side application that stores and enables you to distribute Docker images. To find more about Docker images, see `https://docs.docker.com/registry/`.

Link an External Docker Registry to Your Project

You must be assigned the project **Owner** role to link an external Docker registry to your project.

1. In the navigation bar, click **Project Administration** .
2. Click **Repositories**.
3. In **Docker Registries**, click **+ Link External Registry**.

   To view all linked Docker registries, expand **Linked External Docker Registries**.
4. In **Registry Name** and **Short Description**, enter a unique Docker registry name and a description.
5. In **Registry URL**, enter the URL of the Docker registry. For example: `https://registry-1.docker.io`.

   To link the DockerHub registry, leave the field empty.
   To link to OCIR, enter the registry path in the `https://<ocir_region_code>.ocir.io` format. For example: If your region is Ashburn, enter `iad.ocir.io`. To know the region codes, see Availability by Region Name and Region Code.
6. In **Authentication**, select the authentication type
   - **Basic** (default): Select and enter the basic username and password details in **Username** and **Password**.
     To connect to OCIR, enter the username of the OCI user who can access OCIR in the`<tenancy_name>/<oci_user_name>` format. For example, `mytenancy/myociuser`. In password, enter the OCI user's auth token. See Getting an Auth Token.

- **OAuth2**: Select and enter the long-lived access token as the authentication in **Auth Token**. Don't enter a short-lived access token.
  For more information about OAuth2 in Docker, see `https://docs.docker.com/registry/spec/auth/oauth/`.
- **Anonymous**: Select if the registry can be accessed anonymously and doesn't require an authentication.

7. Click **Create**.

After verifying the credentials and the URL, the registry is added to the **Linked External Docker Registries** section. You can browse its repositories and images from the **Docker** page.
After linking a Docker registry, you can also configure a job to use it while running a build. If you update the linked Docker registry's details, the updated details are passed to the build jobs that use it.

Browse a Linked Docker Registry

While browsing a registry, you can view its repositories and images, download its image manifest file, copy pull and push commands, and delete an image tag.

To browse repositories of a linked Docker registry, select it from the Docker Registry drop-down list.

Action	How To
View images of a repository	Open the Docker registry repository and click the repository name. To view tags of an image, click an image tag.
Download the image manifest of a repository	1. Open the Docker registry repository. 2. In **Image Tags**, click the image tag. 3. On the right side, in the **Info** section of **Tag Details**, click **Download**. 4. Save the file to your computer.
Copy pull and push commands	To get images of a repository, you use the docker pull command. To upload the images, you use the docker push command. You can use these commands while configuring a build job that connects to the Docker registry. 1. Open the Docker registry repository. 2. In **Image Tags**, click the image tag. 3. On the right side, in the **Docker Command** section, pull and push commands are displayed. In the **Pull** tab, click **Copy** to copy the `docker pull` commands to the clipboard. In the **Push** tab, click **Copy** to copy the `docker push` commands to the clipboard.

Action	How To
Delete an image tag	1. Open the Docker registry repository. 2. In **Image Tags**, mouse over the tag you want to delete, and click **Delete**. 3. In the Confirm Delete dialog, select the **I understand that my selected tag will be permanently deleted** check box, and click **Yes**.

8

Use Releases and Export/Import Data

After your project is set, you can set up or use release configurations to download release artifacts. You can also export the project data to an OCI Object Storage bucket or Oracle Cloud Infrastructure Object Storage Classic container for backup and import it later.

This table describes the Oracle Developer Cloud Service pages you'd use to set releases and export/import data.

Use this page ...	To:
Releases	Display the release configurations of the project and enables you to download release artifacts.
Data Export/Import	Export and import project data.

Manage Software Releases

A Release enables you to provide a stable code and artifacts of your applications that project users can download. For a release, you can specify tags or branches of Git repositories with stable code, artifacts of project Maven repository, build artifacts of stable builds, and binary files.

For example, you can you create a release titled *V18-Q1* to mark stable code files, artifacts, and binaries of your application for the first quarter release of 2018 release. Project users then won't have to look around or ask which Git repository or branch has the stable code. They can then download Git repository archives and other artifacts of the *V18-Q1* release from the Release page itself.

You can access and manage releases from the **Releases** page. When a project user opens a release, the user can download source snapshots of a specified branch or tag of the project Git repository, artifacts from the project Maven repository, specified binaries, and archived build artifacts.

Release States

A release can be in Draft, Pre-Release, or Public state.

State	Description
Draft	Indicates that the features of the release are under development. When you create a release, you specify the Maven artifacts and the Git repository tags. While adding a Git repository to a release, you may want to specify a branch that has the stable code at the time of release. Usually, it's the `master` branch, but you can specify any branch name. You may also want to specify the Git repository tag that indicates the stable state of the branch. Usually, the tag is created before the release when the code in the branch is stable. While creating a Release, if you specify a tag name that doesn't exist, it's automatically created when you change the status of the Release to **Public**.
Pre-Release	Indicates that the release is stable, but might need some fixes before it's made **Public**. You usually set the release's status to this state when you and your team have completed all features, staged the software, and are waiting for approvals to release the software. If the Maven artifacts, Git repositories, tags, or branch names have changed since the release was in the **Draft** state, edit the release and update the artifacts.
Public	Indicates that the release is public or is ready to go public. While creating a release, if you specify a tag name that doesn't exist, it's automatically created for the specified branch when the release is set to **Public**. If you've specified an existing tag name, it's used. This might be useful when you create a release, which is already public. You might want to edit the release and update the Maven artifacts, Git repositories, tags, or branch names if they have changed while the release was in the **Pre-Release** or the **Draft** state.

Create a Release

When you create a release, you specify the build artifacts, Git repositories and branches, and Maven artifacts. You can create a release or clone an existing release.

Action	How To
Create a release	1. In the navigation bar, click **Releases**  . 2. Click **+ Create Release**. 3. In **Name** and **Description**, enter a release name and description. 4. In **Status**, specify the status of the release. 5. Add the artifacts. 6. In **Notes**, enter the release notes in the **Page Text** tab. Preview the notes in the **Preview** tab. You can use the project's wiki markup language to format the notes. 7. Scroll to the top of the page and click **Save**.

Action	How To
Clone a release	1. In the navigation bar, click **Releases** . 2. Select the release that you want to edit or clone, click **Actions** and then select **Clone**. 3. In **Name** and **Description**, enter a release name and description. 4. In **Status**, specify the status of the release. 5. Add, update, or remove the artifacts. 6. In **Notes**, enter the release notes in the **Page Text** tab. Preview the notes in the **Preview** tab. You can use the project's wiki markup language to format the notes. 7. Scroll to the top of the page and click **Save**.

Specify Artifacts of a Release

You can specify a release's artifacts when you create it or edit it.

Action	How To
Add Build artifacts	Expand **Builds** and specify the job, build number, and its artifact. Click **Add to Release** to add the artifact. You can specify multiple artifacts. To use the last build of the specified job when the release's status changes to Public, in **Build**, select **Last Build**. When the release's status changes to Public, make sure that the last build is successful and has generated desired artifacts. To store the artifact in the project's storage system, in case the specified job or build is removed, select the **Store** check box.
Add Maven artifacts	Expand **Maven Artifacts** and specify the group ID, artifact ID, and version of artifacts. Click **Add to Release** to add the artifact. You can specify multiple artifacts.
Add Git repositories	Expand **Repositories**, and specify Git repositories and branches (or tags). Click **Add to Release** to add the artifact. You can specify multiple artifacts. If you enter a tag name that doesn't exist, a Git repository tag of the same name is created when the release is marked as **Public**.
Add binary artifacts	You can add binary artifacts only when you edit a release. You can't add binary artifacts when you create a release. Expand **Binaries** and upload the binary files.

Change a Release's Status

You can change the status of a release from the Edit Release page.

1. In the navigation bar, click **Releases** .

2. In the Release list, select the release whose status you want to change.
3. On the right, click **Actions** and select **Edit**.
4. In the Edit Release page, change the selected **Status** option to the desired state.
5. Click **Save**.

A notification about the change in the state of the release is displayed in the Activity Feed of the Project Home page.

Manage Releases

After creating a release, you can edit its artifacts and properties, change its status, or delete it.

Action	How To
Edit a release	On the Releases page, select the release that you want to edit. Click **Actions** and then select **Edit**. On the Edit Release page, update its name, description, artifacts, and click **Save**.
Change a release's state	Edit a release. On the Edit Release page, in **Status**, change the state. The name of the release at the top of the page shows the selected release state.
Delete a release	On the Releases page, select the release that you want to edit. Click **Actions** and then select **Delete**.

Download Artifacts of a Release

To download an artifact, expand its section, and click the file name. Save the file at the desired location on your computer.

You can download these artifacts:

- Binary files from the Binaries section
- Build artifacts of successful builds from the Builds section
- Maven artifacts from the Maven Artifacts section
- An archive of a tag of a Git repository in the Repositories section

Export Project Data to and Import Project Data from Oracle Cloud

You can export your project's data to an OCI Object Storage bucket or an OCI Object Storage Classic container of any data center to perform a backup. You can then import the data into the same or another project of the same or a different data center when you want.

To export or import data, first, you'd need to set up a connection to OCI Object Storage or OCI Object Storage Classic.

Exported Data

Before you export a project's data, note that not all the artifacts of the project are exported. You'll have to manually export the remaining artifacts and data manually.

This table shows you which artifacts are exported and which aren't:

Artifact	Exported?	Notes
Project users	No	When you export a project's data, its users are not exported, but all data associated to usernames (such as issue ownership and reviewers of a merge request) is preserved. After you import the project's data to another project, when you add a user to the project with the same username, the data associated to the username is automatically restored.
User's favorite settings or personal preferences	No	
Hosted Git repositories	Yes	
Mirrored public external Git repositories	Yes	
Mirrored private external Git repositories	No	Password protected external Git repositories aren't exported. After you import the project's data to another project, you must add each external private Git repository.
Branch restrictions	Yes	
Merge Requests	Yes	
Default reviewers of a branch	Yes	After you import the project's data to another project, default reviewers are added automatically after the same users are added to the target project.
Maven artifacts	No	
Linked Docker registries	No	

Artifact	Exported?	Notes
Build jobs	Yes	All builds of jobs are exported, along with their logs and artifacts. If a job retains excessive builds, it affects the export process and consumes disk space on your bucket or container. It's recommended that you configure a job to retain reasonable number of builds before you export the project.
Build job's history and artifacts	Yes	
Build job's VM template name	No	
Pipelines	Yes	
Releases	Yes	
Deployment configurations	No	
Environments	No	
Issues	Yes	
Agile boards	Yes	
Wiki pages	Yes	
Snippets	Yes	
Project template definition	No	
Announcements	No	
Webhooks	No	
RSS/ATOM feeds	No	
Link rules	No	
Project tags	Yes	
Issue products and components	Yes	
Default owners of issue components	Yes	After you import the project's data to another project, owners are activated automatically after the same users are added to the target project.
Issue custom fields	Yes	
Named passwords	Yes	

Export to and Import from an OCI Object Storage Bucket

If you're an OCI user, you can export your project's data to an OCI Object Storage bucket and import from it.

The exported data isn't encrypted and can be downloaded from the bucket. If you're exporting the project's data for the first time, set up an OCI Object Storage bucket for the project and users who can read from or write to it. You can use a common bucket for all projects of the organization, but it's recommended that you use a separate bucket for each project. This allows you to organize archive files better as they aren't mixed with the archive files of other projects. Contact the OCI administrator to create

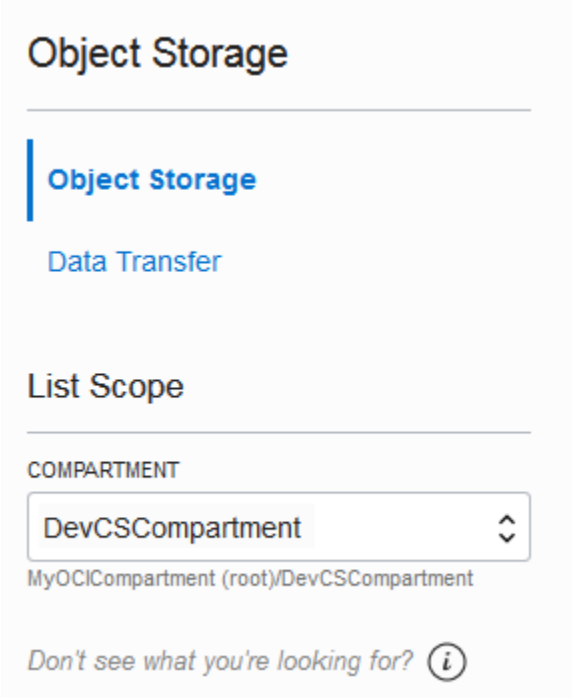
the bucket. You should also ask the OCI administrator to set up users with read-write access to the bucket.

Set Up the OCI Object Storage Bucket

To set up the OCI Object Storage bucket, sign in as the OCI administrator and follow these steps:

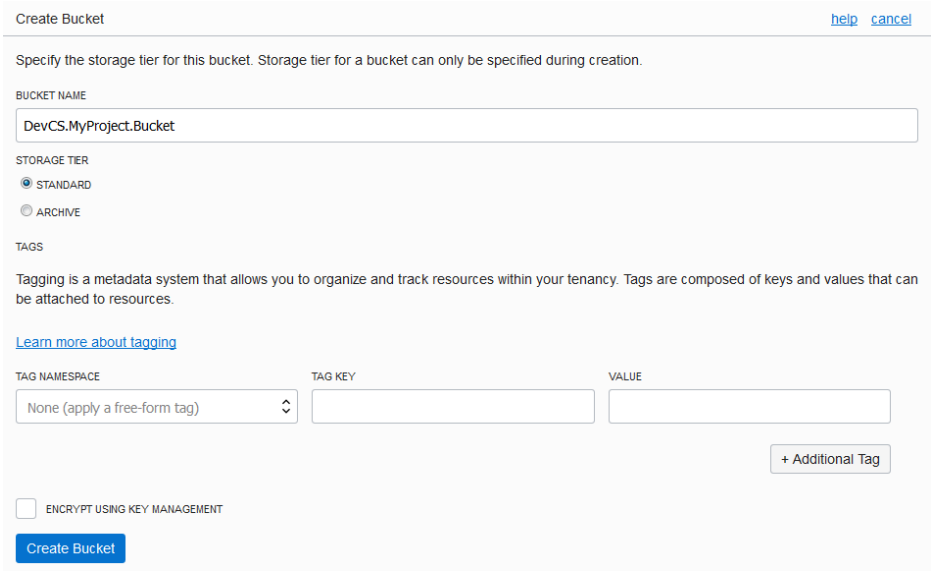
1. In the compartment that hosts DevCS resources, create a bucket for the project.
   a. In the left navigation bar, under **Core Infrastructure**, go to **Object Storage** and click **Object Storage**.
   b. On the left side of the Object Storage page, from the **Compartment** list, select the DevCS compartment.

      Example:



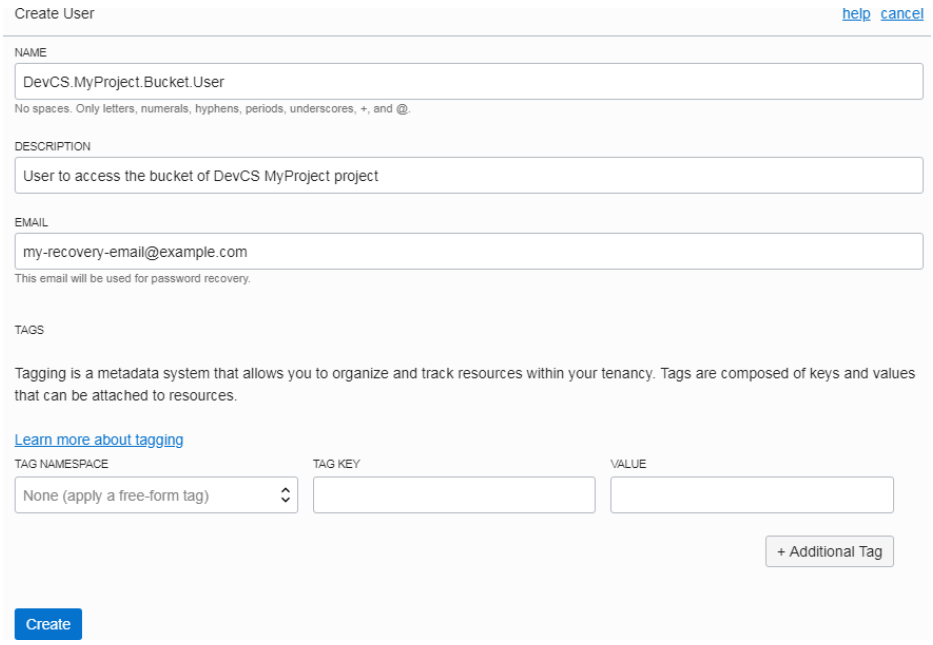
   c. Click **Create Bucket**.
   d. In the Create Bucket dialog box, fill in the details, and click **Create Bucket**.

      Example:



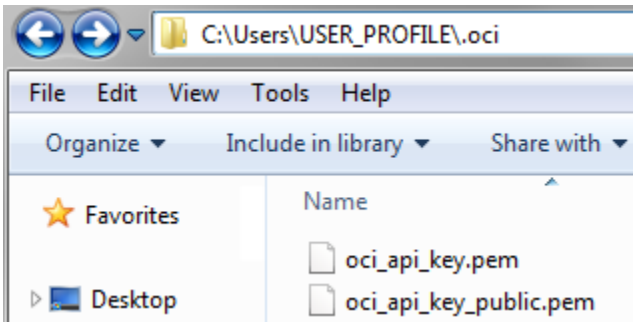
2. Create a user to access the bucket.
   a. In the left navigation bar, under **Governance and Administration**, go to **Identity** and click **Users**.
   b. Click **Create User**.
   c. In the Create User dialog box, fill in the fields, and click **Create**.

      Example:



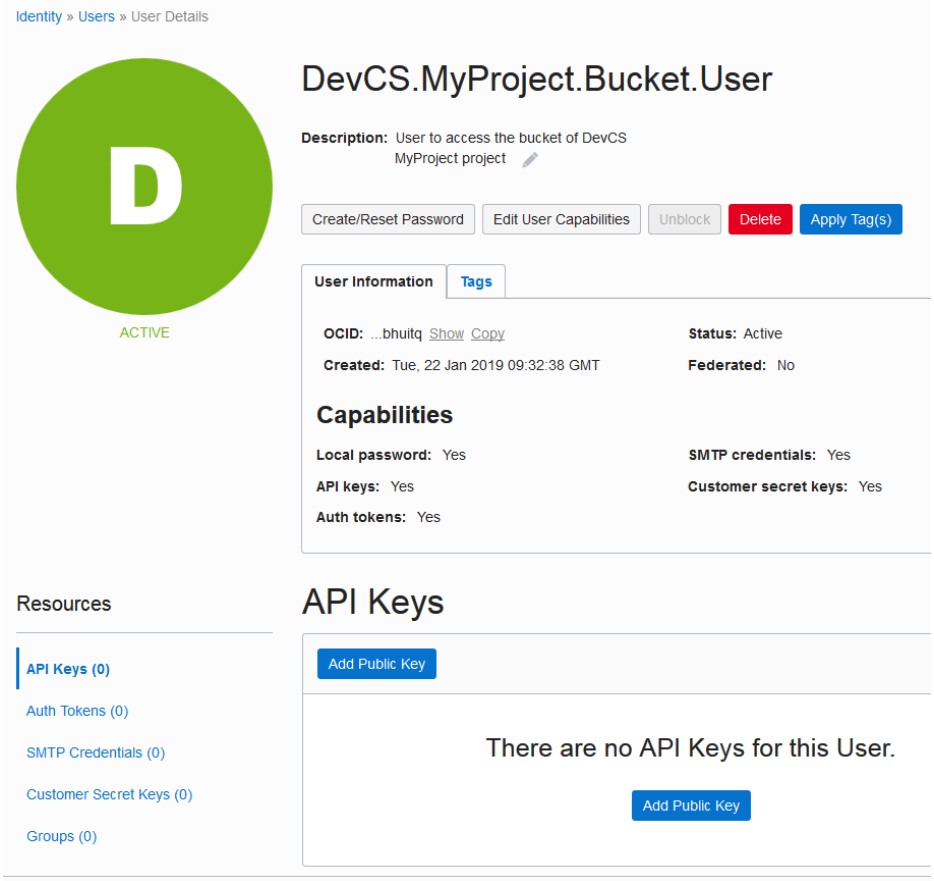
3. On your computer, generate a private-public key pair in the PEM format.

   To learn more, see How to Generate an API Signing Key.
   Example of private-public key files on a Windows computer:



4. Upload the public key to the user's details page.
   a. Open the public key file in a text editor and copy its contents.
   b. In the left navigation bar of the OCI dashboard, under **Governance and Administration**, go to **Identity** and click **Users**.
   c. Click the user's name created in Step 2.
   d. In the User Details page, click **Add Public Key**.

      Example:

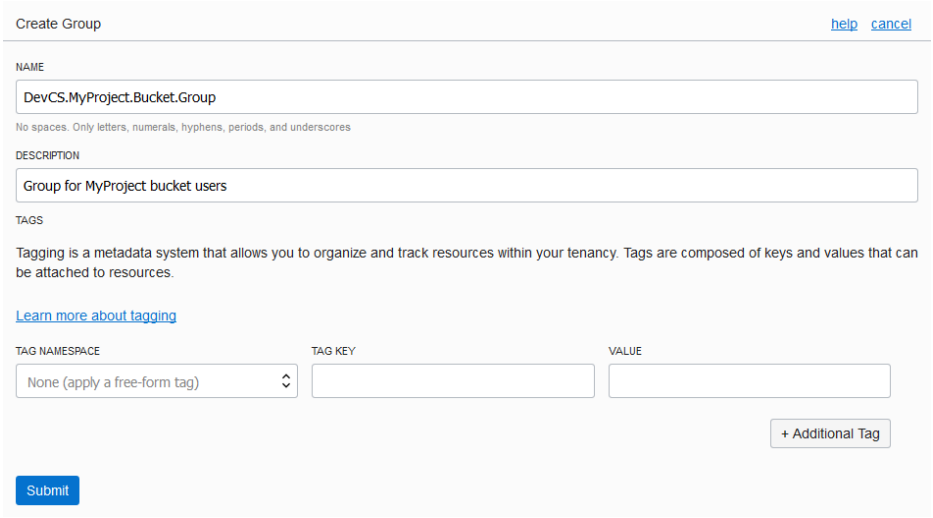


   e. In the Add Public Key dialog box, paste the contents of the public key file, and click **Add**.

To learn more, see How to Upload the Public Key.

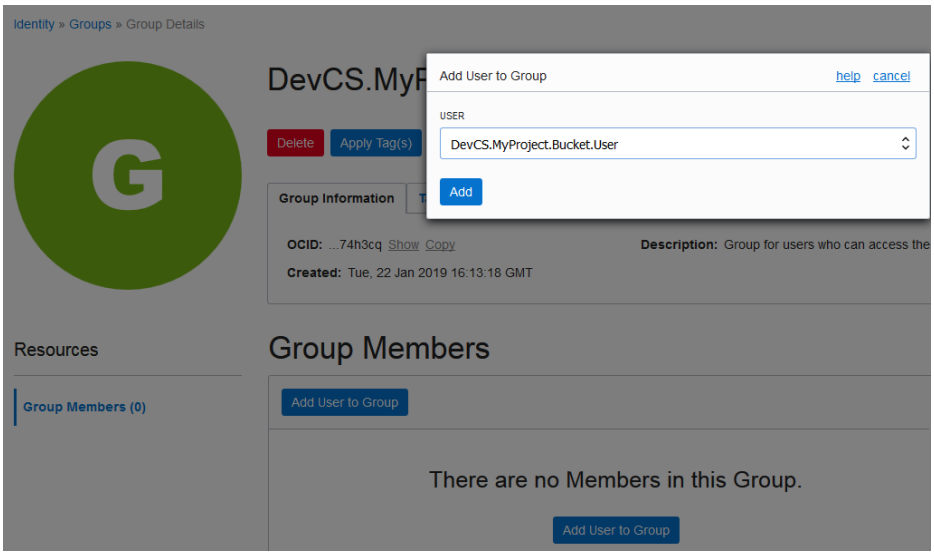
5. On the Groups page, create a group for the user who can access the bucket and add the user to the group.
   a. In the left navigation bar, under **Governance and Administration**, go to **Identity** and click **Groups**.
   b. Click **Create Group**.
   c. In the Create Group dialog, fill in the fields and click **Submit**.

      Example:



   d. On the Groups page, click the group's name.
   e. On the Group Details page, click **Add User to Group**.
   f. In the Add User to Group dialog box, select the user created in Step 2, and click **Add**.

      Example:



To learn more, see Working with Groups.

6. In the DevCS compartment, create a policy with read and write access to the bucket.

   You can give read and write access to the same user, or create different users.

   a. In the left navigation bar, under **Governance and Administration**, go to **Identity** and click **Policies**.

   b. On the left side of the Policies page, from the **Compartment** list, select the DevCS compartment.

   c. Click **Create Policy**.

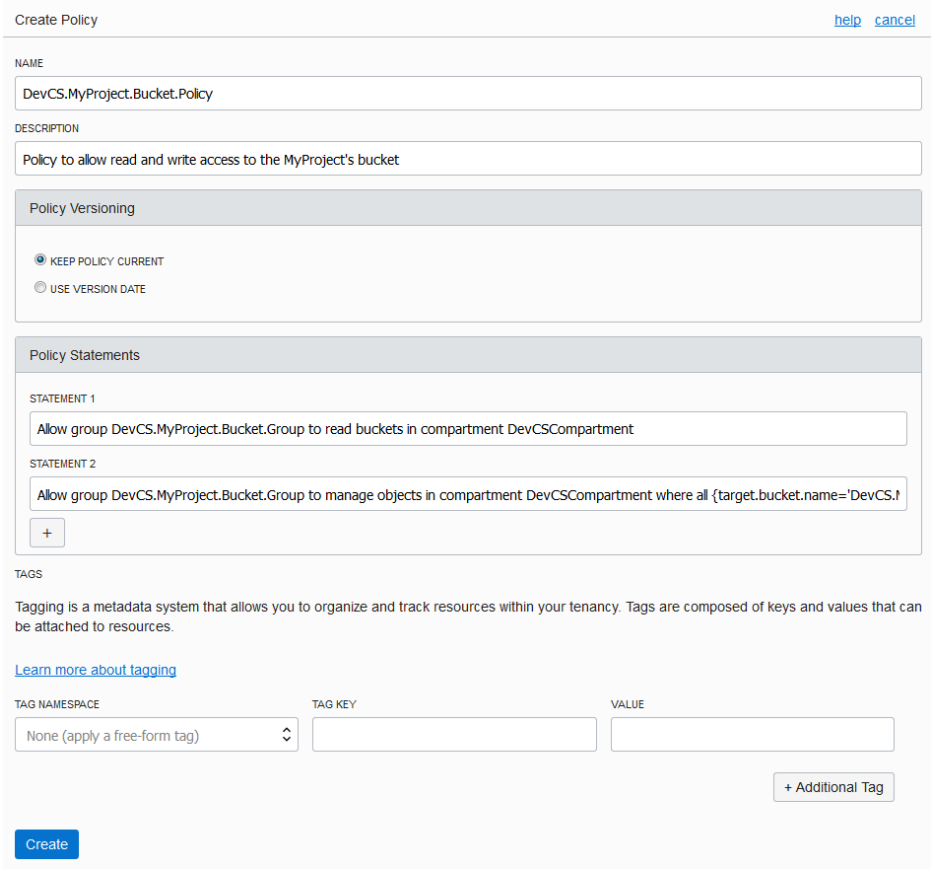
   d. In **Name** and **Description**, enter a unique name and a description.

   e. In **Policy Statements**, add statements to restrict read and write access to the bucket.

      To allow different user groups to read objects from and write objects to the bucket, create separate policies. Here are some statement examples:

To:	Add these statements:
Allow a group to read from and write objects to a bucket (required to import and export a project's data)	`allow group <group-name> to read buckets in compartment <compartment-name>` `allow group <group-name> to manage objects in compartment <compartment-name> where all {target.bucket.name='<bucket-name>', any {request.permission='OBJECT_CREATE', request.permission='OBJECT_INSPECT'}}`
Allow a group to download objects from a bucket (required to import a project's data)	`allow group <group-name> to read buckets in compartment <compartment-name>` `allow group <group-name> to read objects in compartment <compartment-name> where target.bucket.name='<bucket-name>'`

      Example:



f. Click **Create**.

To learn more, see Working with Policies.

Export Project Data

When you export project data, DevCS exports data to an archive file in the specified OCI Object Storage bucket.

To export project data to an OCI Object Storage bucket, you need the following:

- Name of the target bucket
- Private key and fingerprint of a user who can write objects to the bucket
- Details of the compartment that hosts the bucket

Contact the OCI administrator for the details and get the required input values. See Get the Required OCI Input Values.

You must be assigned the project **Owner** role to export data.

1. Open the DevCS project.
2. In the navigation bar, click **Project Administration** .
3. Click **Data Export/Import**.
4. Click the **Job** tab.
5. In **Account Type**, select **OCI**.

6. In **Tenancy OCID**, enter the tenancy's OCID copied from the Tenancy Details page.
7. In **User OCID**, enter the user's OCID value who can access the bucket.
8. In **Home Region**, select the home region of the OCI account.
9. In **Private Key**, enter the private key of the user who can access the bucket.
10. In **Passphrase**, enter the passphrase used to encrypt the private key. If no passphrase was used, leave the field empty.
11. In **Fingerprint**, enter the fingerprint value of the private-public key pair.
12. In **Compartment OCID**, enter the compartment's OCID copied from the Compartments page.
13. In **Storage Namespace**, enter the storage namespace copied from the Tenancy Details page.
14. Click **Connect**.
15. In the Create Job section, in **Type**, select **Export**.
16. In **Name**, enter a name for the export job.
17. In **Description**, enter the job's description.
18. In **Storage Container**, select the bucket to export the project data.
19. In **Storage Object**, if required, update the default `.zip` file name.
20. Click **Export**.
21. In the Confirm Project Export dialog box, select the **Export project data** check box, and click **Yes**.
22. In the Exporting Project page, expand **Steps** to see the status of each module.

After the export is complete, a notification is added to the Recent Activities feed of the **Project Home** page and to the History tab of the **Data Export/Import** page.

Import Project Data

When you import data, it overwrites the data of the project. All artifacts of the project are replaced with the components from the imported project.

To import project data from an OCI Object Storage bucket, you need the following:

- Name of the target bucket
- Name of the archive file with the project data
- Private key and fingerprint of a user who can read objects from the bucket
- Details of the compartment that hosts the bucket

Contact the OCI administrator for the details and get the required input values. See Get the Required OCI Input Values.

 You must be assigned the project **Owner** role to import data.

1. Open the DevCS project.
2. In the navigation bar, click **Project Administration**  .
3. Click **Data Export/Import**.

4. Click the **Job** tab.
5. In **Account Type**, select **OCI**.
6. In **Tenancy OCID**, enter the tenancy's OCID copied from the Tenancy Details page.
7. In **User OCID**, enter the user's OCID value who can access the bucket.
8. In **Home Region**, select the home region of the OCI account.
9. In **Private Key**, enter the private key of the user who can access the bucket.
10. In **Passphrase**, enter the passphrase used to encrypt the private key. If no passphrase was used, leave the field empty.
11. In **Fingerprint**, enter the fingerprint value of the private-public key pair.
12. In **Compartment OCID**, enter the compartment's OCID copied from the Compartments page.
13. In **Storage Namespace**, enter the storage namespace copied from the Tenancy Details page.
14. Click **Connect**.
15. In the Create Job section, in **Type**, select **Import**.
16. In **Name**, enter a name for the import job.
17. In **Description**, enter the job's description.
18. In **Storage Container**, select the bucket to import the project data from.
19. In **Storage Object**, select the `.zip` file name of the exported data.
20. Click **Import**.
21. In the Confirm Project Import dialog box, read and verify the container and object details, select the **Import project data** check box, and click **Yes**.
22. In the Importing Project page, expand **Steps** to see the status of each module.

    If you want to cancel the import process, click **Cancel**.

When an import job is in progress, the project is in the locked state. You can't access other pages of the project until the import job is complete. After the import is complete, you're redirected to the **Project Home** page. A notification of the import action is added to the Recent Activities feed of the **Project Home** page and to the History tab of the **Data Export/Import** page.

Export to and Import from an OCI Object Storage Classic Container

If you're an OCI Classic user, you can export project data to an OCI Object Storage Classic container and import from it.

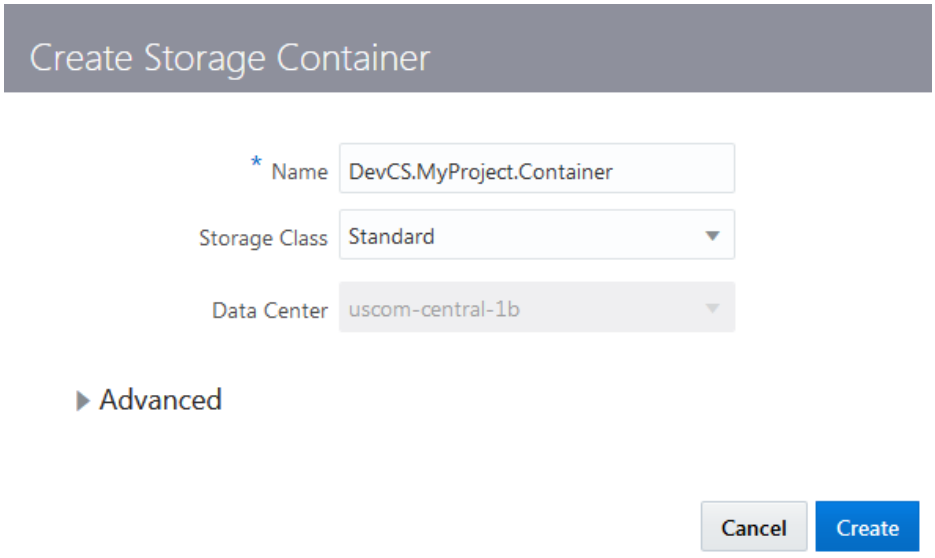
The exported data isn't encrypted and can be downloaded from the container. If you're exporting the project's data for the first time, set up an OCI Object Storage Classic container for the project and users who can read from or write to it. You can use a common container for all projects of the organization, but it's recommended that you use a separate container for each project. This allows you to organize archive files better as they aren't mixed with the archive files of other projects. Contact the identity domain administrator or the OCI Object Storage Classic administrator to create the container. You should also ask the administrator to set up users with read-write access to the bucket.

Set Up the OCI Object Storage Classic Container

An identity domain administrator or the OCI Object Storage Classic administrator can create an OCI Object Storage Classic container. After creating the container, assign a user the **Storage_ReadWriteGroup (Storage_ReadWriteGroup)**. If you want the user to read files from the container, but don't want them to write or delete files, assign the **Storage_ReadOnlyGroup (Storage_ReadOnlyGroup)** role to the user. Then, share the container and user details with the project owner. Project owner will need the details to connect to the container to export or import data.
You must be an identity domain administrator to create a container and set up the user.

1. On the OCI Object Storage Classic console, create a container for the project.
   a. Open the Oracle Cloud My Services dashboard.
   b. On the Oracle Cloud Dashboard, in the **Storage Classic** tile, click **Action** and select **Open Service Console**.
   c. On the Storage Classic page, click **Create Container**.
   d. In the Create Storage Container dialog box, enter the container name, and click **Create**.

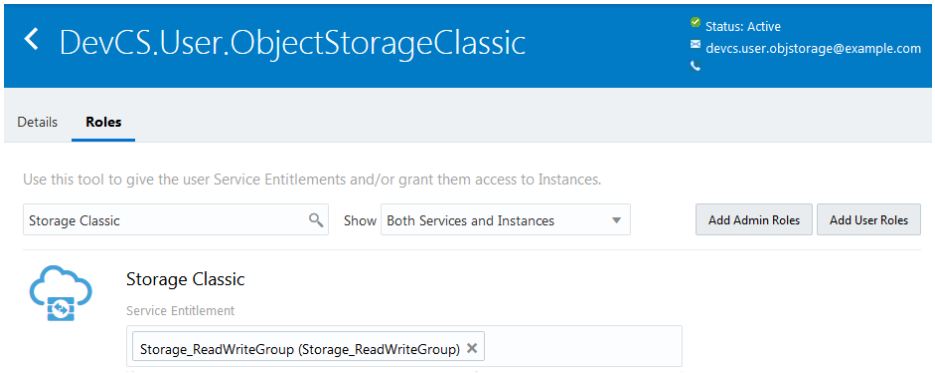
      Example:



2. Set up a user to access the container.
   a. On the Oracle Cloud My Services dashboard, click **Users**.
   b. Click the user's name tile to whom you want to assign the access.

      To create a user, click **Add** and fill in the details.
   c. Click the **Roles** tab.
   d. In **All Services**, search for `Storage Classic`.
   e. To assign read and write access to the user, add the **Storage_ReadWriteGroup (Storage_ReadWriteGroup)** role.

To assign the read access, add the **Storage_ReadOnlyGroup (Storage_ReadOnlyGroup)** role.
Example:



f. Return to the dashboard.

Export Project Data

When you export project data, DevCS exports data to an archive file in the specified OCI Object Storage Classic container.

To export project data to an OCI Object Storage Classic container, you need the following:

- Name of the target container
- Credentials of a user with the **Storage.Storage_Administrator** or the **Storage_ReadWriteGroup** identity domain role.
- Service ID and the authorization URL of OCI Object Storage Classic

Contact the identity domain administrator or the OCI Object Storage Classic administrator for the details and get the required input values. See Get OCI Object Storage Classic Input Values.

You must be assigned the project **Owner** role to export data.

1. Open the DevCS project.
2. In the navigation bar, click **Project Administration** .
3. Click **Data Export/Import**.
4. Click the **Job** tab.
5. In **Account Type**, select **OCI Classic**.
6. In **Service ID**, enter the value copied from the last part of the **REST Endpoint** URL field of the Service Details page.

   For example, if the value of **REST Endpoint** URL is `https://demo12345678.storage.oraclecloud.com/v1/Storage-demo12345678`, then enter `Storage-demo12345678`.
7. In **Username** and **Password**, enter the credentials of the user assigned the **Storage.Storage_Administrator** or **Storage_ReadWriteGroup** identity domain role.

8. In **Authorization URL**, enter the URL copied from the **Auth V1 Endpoint** field of the Service Details page.

   Example: `http://storagetria01234-usoracletria12345.storage.oraclecloud.com/auth/v1.0`.

9. Click **Connect**.
10. In the Create Job section, in **Type**, select **Export**.
11. In **Name**, enter a name for the export job.
12. In **Description**, enter the job's description.
13. In **Storage Container**, select the container to export the project data.
14. In **Storage Object**, if required, update the default .zip file name.
15. Click **Export**.
16. In the Confirm Project Export dialog box, select the **Export project data** check box, and click **Yes**.
17. In the Exporting Project page, expand **Steps** to see the status of each module.

After the export is complete, a notification is added to the Recent Activities feed of the **Project Home** page and to the History tab of the **Data Export/Import** page.

Import Project Data

When you import data, it overwrites the data of the project. All artifacts of the project are replaced with the components from the imported project.

To import project data from an OCI Object Storage Classic container, you need the following:

- Name of the target container
- Name of the archive file with the project data
- Credentials of a user with the **Storage.Storage_Administrator**, **Storage_ReadWriteGroup**, or **Storage_ReadOnlyGroup** identity domain role
- Service ID and the authorization URL of OCI Object Storage Classic

Contact the identity domain administrator or the OCI Object Storage Classic administrator for the details and get the required input values. See Get OCI Object Storage Classic Input Values.

You must be assigned the project **Owner** role to import data.

1. Open the DevCS project.
2. In the navigation bar, click **Project Administration** .
3. Click **Data Export/Import**.
4. Click the **Job** tab.
5. In **Account Type**, select **OCI Classic**..
6. In **Service ID**, enter the value copied from the last part of the **REST Endpoint** URL field of the Service Details page.

For example, if the value of **REST Endpoint** URL is `https://demo12345678.storage.oraclecloud.com/v1/`**`Storage-demo12345678`**, then enter `Storage-demo12345678`.

7. In **Username** and **Password**, enter the credentials of the user assigned the **Storage.Storage_Administrator**, **Storage_ReadWriteGroup**, or **Storage_ReadOnlyGroup** identity domain role.
8. In **Authorization URL**, enter the URL copied from the **Auth V1 Endpoint** field of the Service Details page.

   Example: `http://storagetria01234-usoracletria12345.storage.oraclecloud.com/auth/v1.0`.
9. Click **Connect**.
10. In the Create Job section, in **Type**, select **Import**.
11. In **Name**, enter a name for the import job.
12. In **Description**, enter the job's description.
13. In **Storage Container**, select the container to import the project data from.
14. In **Storage Object**, select the .zip file name of the exported data.
15. Click **Import**.
16. In the Confirm Project Import dialog box, read and verify the container and object details, select the **Import project data** check box, and click **Yes**.
17. In the Importing Project page, expand **Steps** to see the status of each module.

    If you want to cancel the import process, click **Cancel**.

When an import job is in progress, the project is in the locked state. You can't access other pages of the project until the import job is complete. After the import is complete, you're redirected to the **Project Home** page. A notification of the import action is added to the Recent Activities feed of the **Project Home** page and to the History tab of the **Data Export/Import** page.

View Export and Import History of the Project

If you want to know the export and import history of a project, you can do so from the History tab of the **Data Export/Import** page.

1. In the navigation bar, click **Project Administration**  .
2. Click **Data Export/Import**.
3. Click the **History** tab.

The history of all export and import jobs is displayed. Select a job to view its details. In the case of a failure, expand **Steps** to view the modules that passed and failed.

9

Organization and Project Management

With the organization's administrator and the project's owner role, you can configure and manage components of the project, and projects of the organization,

This table describes the Oracle Developer Cloud Service pages you'd use to manage the organization and the project.

Use this page ...	To:
Organization	Manage the organization.
Project Administration	Configure the project.

Manage the Organization

You can update your Oracle Developer Cloud Service (DevCS) organization's name, description, manage projects, manage virtual machines (VM) and VM templates, and view the usage metric of projects.

You must be the **Organization Administrator** to manage the organization.

Update the Organization's Display Name and Description

By default, the organization's display name is `<your_devcs_instance_name>-<your_oracle_account_name>`.

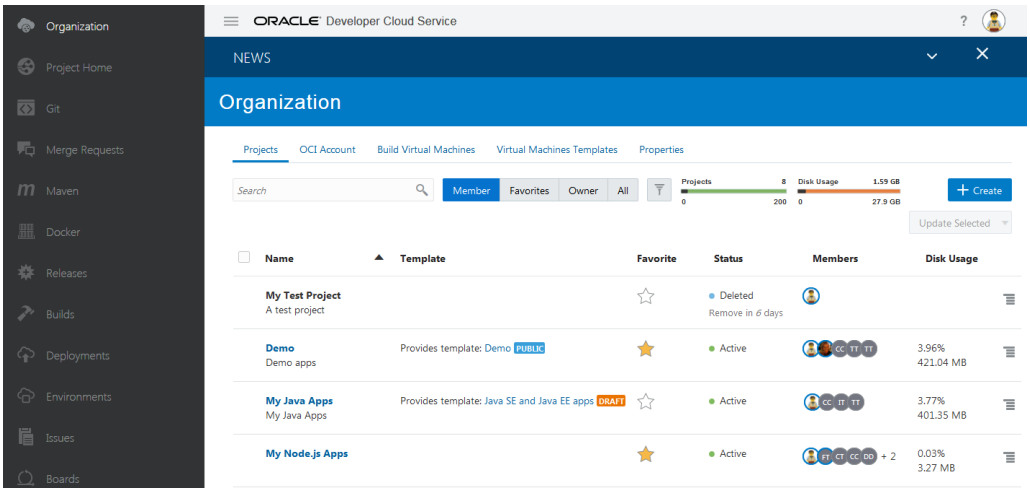
1. In the navigation bar, click **Organization** .
2. Click the **Properties** tab.
3. In **Name**, edit the organization name.
4. In the Confirm Organization Name Change dialog, click **Yes**.
5. In **Description**, if necessary, enter or edit the description of the organization.

Manage Projects of the Organization

As the Organization Administrator, you can access and manage all projects. You can assign a project's Owner role to yourself, delete a project, revoke a deleted project, or postpone a deleted project by a few days.

1. In the navigation bar, click **Organization** .
2. Click the **Projects** tab.

Example:



Action	How To
Create a project	Click **+ Create**. See Create a Project.
Open a project	In the **Name** column, click the project name. See Open a Project. Note that you can open a project only if you're a member or an owner. Projects where you aren't a member or an owner don't appear as links. You can't open a project that's in the deleted state.
Search for a project	Use the search box above the projects list.
Filter projects	In **Member**/**Owner**/**Favorite**/**All**, click a filter option to enable it. Click the **Member** button to view projects where you're a member. Click **Owner** to view projects where you're an Owner. Click **Favorite** to view your favorite projects. Click **All** to view all projects of the organization. To see more filter options, click **Detailed Filter** .
Mark a project as a favorite	In the **Favorite** column, click **Favorite** and add it to your favorites list.
View a project's team members	The **Members** column shows the gravatars of the project's team members. A blue circle around a user's gravatar indicates the user is assigned the project's **Owner** role. Click the gravatars to display names and email addresses of all team members.
Assign a project's Owner role to yourself	You can assign a project's Owner role to yourself. This is usually helpful in a case when all Owners of the project are removed or have left the organization. As the Organization Administrator, you can then take the ownership of such projects and then assign the Owner role to another project user. For the project whose ownership you want to assign to yourself, click **Update Project** and select **Assign Me as Owner**.

Action	How To
Delete a project	As the Organization Administrator, you can delete any project. 1. For the project you want to delete, click **Update Project** and select **Delete**. 2. In the Delete Project dialog box, click **Delete**. The project's **State** is marked as `Deleted - Project will be removed automatically in N days.`, but isn't removed immediately. In the Delete Project dialog box, read the message before you click **Delete**. Also, check the project's delete message to know when the project will be permanently removed. To immediately remove the project permanently, click and select **Remove Forever**.
Postpone removal of a deleted project	At times, you may be unsure if you want to revoke a deleted project that's close to its permanent removal date. For such projects, you can choose to postpone their removal by a few days. For the deleted project you want to postpone, click **Update Project** and select **Postpone Removal**. The project's permanent removal date is rescheduled and the message is displayed at the top. In **Status**, check the project's delete message to know when the project will be permanently removed.
Undelete a project	When you undelete a project, you get an option to update the name and description of the project. When revoked, the project URLs will change if a different name of the deleted project is specified. 1. For the deleted project you want to undelete, click **Update Project** and select **Undelete**. 2. In the Undelete Project dialog, if necessary, update the project name in **Name** and description in **Description**. You would have to change the project name if another project with the same name was created after this project was deleted. 3. Click **Undelete**.
Select multiple projects	You can select multiple projects and perform a common action on them. 1. If necessary, filter and sort the projects as desired. 2. Press the **Ctrl** key or the **Shift** key and select project rows. To select all projects, click the check box to the left of the **Name** column header. 3. Click **Update Selected** and select the desired action from the menu. Some options, such as **Undelete**, are not available in the **Update Selected** menu.
View usage metrics of projects	Next to the **Detailed Filter**, review the projects and disk usage metrics of the organization. For each project, the **Disk Usage** column shows the project's disk usage and its usage percentage with respect to the total disk usage of all projects. Click the number to open a pop-up with details of usage by each service in the project.

Create and Manage OCI Account and Virtual Machines

To configure the OCI account, see Connect to OCI or OCI Classic. To manage virtual machines and virtual machine templates, see Set Up the Build System.

Manage the Project

As an Oracle Developer Cloud Service (DevCS) project owner, you can perform various project-wide actions, such as edit a project's name and description, configure the project as a project template, create announcements, set project tags, manage repositories, and configure link rules.

You must be assigned the project **Owner** role to manage a project.

Edit a Project's Name, Description, or Visibility

After creating a project, you can edit a project's properties from the **Project Administration** page.

1. In the navigation bar, click **Project Administration**.
2. Click **Properties**.
3. In **Name** and **Description**, update the project name and description.
4. In **Security**, update the project's share status.

When you're finished, use the project navigation bar to switch to another page.

Configure Project Templates

You can define an existing project as a template for new projects that users can use as a starting point. When creating a project, if a user selects a project template, the project template's data is copied to the new project, which the user can modify.

Project template data may include its Git repositories, build job configurations, deployment configurations, links, wikis, and announcements. For example, if a project template hosts an application in its Git repositories; information on how to use the application in its wiki pages; build configuration in its jobs; and Oracle Cloud deployment target details in its deployment configurations, then the data of the project template is copied to the new project. Members of the new project can use the application, run builds of pre-configured jobs, and deploy build artifacts to Oracle Cloud using pre-configured deployment configurations without making any changes to the code or any of its configurations.

When you define a project template, you define its visibility (who can use the project template), configure rules (what data can be copied from the project template), and use variables (customize actions based on user input when the data is copied).

After a project is created using a template, any updates made to the template project aren't reflected in the created project.

Define and Manage a Project Template

You can define a project template from the **Project Administration** page.

1. In the navigation bar, click **Project Administration** .
2. Click **Properties**.
3. On the Properties page, in the **Template** section, click **Define Template**.

   The project is now marked as a template with default rules and visibility state.
4. To edit the template, click **Edit**.
5. In **Visibility**, select the template's visibility.

Visibility Type	Description
Draft	The project template is under design and isn't available in the Templates page of the Create Project wizard. No user can copy data from a draft template.
Private	The project template is published and available to organization's users, but isn't visible by name to all. It's visible by name to members of the project template. A non-member user can access the template using the project template's private key and can copy data.
Shared	The project template is published and is available to all users of the organization. Any user can copy data from this project.

6. To change the name and description of the project's template, enter a new name in **Name** and description in **Description**.
7. To specify an icon for the template, in **Icon**, click **Change**, browse and upload an image of size 48x48 pixels.
8. In the **Variables** and **Rules** sections, specify the variables and rules. Click **Save** when you're done.

Define a Private Project Template

You can define a private project template if you don't want all users of the organization to copy data from the project. Private project templates aren't listed by name in the Templates page of the New Project wizard unless you're a member of the project template. Non-members can use a private project template only if they have the private key of the project template.

1. In the navigation bar, click **Project Administration** .
2. Click **Properties**.
3. In the **Template** section, click **Define Template**.

   The project is now defined to be used as a template with the default rules and properties.
4. Click **Edit**.
5. On the Template page, in the **Visibility** section, select **Private**.
6. For the **Private Key** field, click **Show**. Note down the key value.
7. Update settings, add variables, and define rules, as desired.
8. Click **Save**.

Share the private key with users whom you want to use the project template and copy the project data.
To generate a new private key, edit the project template, click **Show**, and then click **Regenerate**. You may want to do this if you don't want users who already have the old key value to copy the project data from the template.

Define Project Template Rules

Rules define the artifacts to copy from the template project to the new project.

When you define a project as a project template, all rules are enabled by default. Some rules can't be edited and some rules can’t be added more than once.

This table describes the available rules.

Use this rule ...	To copy:	Can this rule be edited?	Can this rule be added more than once?
Build Jobs	Build jobs and pipelines of the project template to the new project.	No	No
Wiki Content	Wiki pages of the project template to the new project.	No	No
Links	Link rules of the project template to the new project.	No	No
Git Repository	Specified Git repositories along with its branches to the new project.	Yes	Yes
External Git Repository	Specified external Git repository along with its branches to the new project.	Yes	Yes
Deployments	Oracle Java Cloud Service - SaaS Extension deployment configurations of the project template to the new project. Other deployment configurations are not copied.	No	No
Announcements	Announcements of the project template to the new project.	No	No

To add or remove a rule, follow these steps.

1. Open the Template Settings page.
2. To add a rule, scroll to the **Rules** section. From the **Add Rule** drop-down list, select the **Git Repository** rule.

To remove a rule, click **Remove** .

Add or Edit a Git Repository Rule

By default, all hosted and external Git repositories rules are enabled.

You can edit the default rules and add new Git Repository rules if you added Git repositories to the project after the template was defined. You can also add a rule to make a copy of an existing Git repository on the new project that uses the template. If you don't want a Git repository to be included in the template, remove its rule.

Action	How To
Add or edit a hosted Git repository rule	1. On the **Properties** page, in the **Template** section, click **Edit**. 2. In the **Rules** section, to add the rule, click **Add Rule**. From the menu, select **Git Repository**. To edit an existing rule, to the right side of the Git repository rule, click **Edit** . 3. In **Source Repository**, specify the name of the Git repository to be copied. 4. In **Repository Name**, specify the new name of the Git repository. A `.git` extension is automatically added if you missed it. To use the new project name as the name of the Git repository, select the **Use target project name** check box. 5. If necessary, in **Replacements**, define file name replacements of the Git repository matching the specified criterion. This is useful if you want to make a copy of an existing Git repository. To add a new replacement rule, click **Add new replacement**. a. From the **In** drop-down list, select the files where the replacements apply. You can specify all files, files matching an Ant pattern, or a specific file. b. In **Replace**, specify the search term. c. In **With**, specify the replacement term. You can select a pre-defined variable such as **Project Id**, **Project Name**, **Project URL Name**, or **Repository Name**. To use a variable defined in the **Variables** section, select **Variable** and then select the variable name. d. Click **Save** . When you create a project using this template, the project creation wizard searches through the specified files in the Git repository and replaces the term with the specified value of the selected variables. 6. Click **Save** to save the Git repository rule. 7. Click **Save** to save the project template.

Action	How To
Add or edit an external Git repository rule	1. On the **Properties** page, in the **Template** section, click **Edit**. 2. In the **Rules** section, to add the rule, click **Add Rule**. From the menu, select **External Repository**. To edit an existing rule, to the right side of the Git repository rule, click **Edit** . 3. In **Repository URL**, enter the external Git repository URL. To update a rule, enter a new URL. 4. In **Username** and **Password**, enter credentials to access the external Git repository. For public Git repositories, don't fill these fields. 5. In **Repository Name**, specify the new name of the Git repository. A `.git` extension is automatically added if you missed it. To use the new project name as the name of the Git repository, select the **Use target project name** check box. 6. Click **Save** . 7. Click **Save**.

Add and Manage Variables

Variables define user input. Based on the input, you can configure the template to change the action or properties of data that's copied to the new project. You can add or edit variables from the Variables section on the Template page.

1. Open the Template Settings page.
2. In the Variable section, from the **Add Variable** menu, select the variable type.

Use this variable type:	To accept a:
Boolean	Boolean string value (such as True-False, Yes-No, or any string values).
Choice	Value from a list of values configured by project owners.
String	String value.
URL	URL value.

3. In **Name**, **Display Name**, **Description**, enter the variabe's unique name, the display name, and its description. Fill in the other fields of the variable and click **Save** .

   Later, if you want to edit the variable, click **Edit** . Update the fields of the variable and click **Save** . To delete the variable, click **Remove** .

Delete a Project's Template

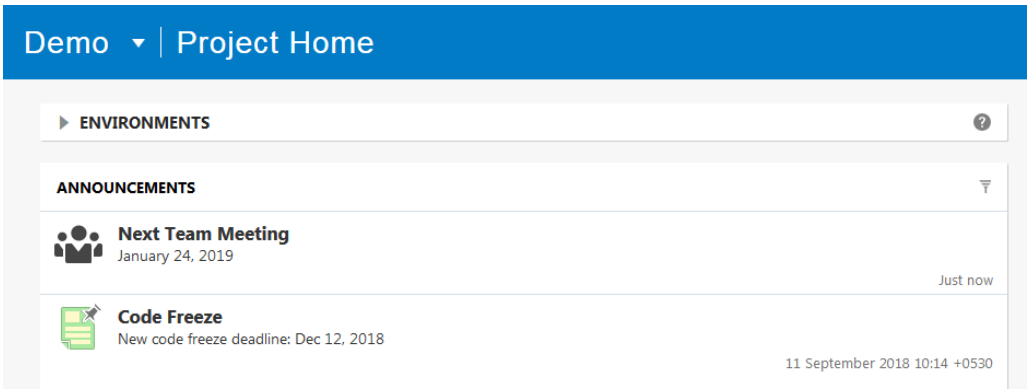
When you delete a project template, you delete the template metadata (rules and variables). The project itself isn't deleted.

1. In the navigation bar, click **Administration**.
2. Click **Properties**.
3. In the **Template** section, click **Delete Template**.
4. In the Delete Template Definition dialog box, click **Delete Template**.

Manage Project Announcements

Project announcements are messages available on the **Project Home** page above the Recent Activity feed.

Example:



To create an announcement, follow these steps:

1. In the navigation bar, click **Project Administration** .
2. Click **Announcements**.

Action	How To
Create an announcement	1. On the Announcements page, click **+ New Announcement**. 2. In **Name** and **Contents**, enter name and announcement's text. You can use the project's wiki markup to format the text. 3. Upload an icon, if necessary. The icon's size must be 48x48 pixels. 4. Click **Done**.
Copy an announcement	Instead of creating an announcement, you can copy the contents and icon of an existing announcement, and edit it. 1. In the Announcements list, select the announcement to copy. 2. Click **Copy Announcement**. 3. Edit the details in the Create Announcement page. 4. Click **Done**.

Action	How To
View or edit an announcement	1. In the Announcements list, select the announcement. 2. In the Announcement section to the right of the list, view or edit the announcement's details. Any changes made to the fields are saved immediately when the focus moves out of the field.
Deactivate or activate an announcement	If you don't want to display an announcement and don't want to delete it either, you can deactivate it. Deactivated announcements aren't visible on the **Project Home** page. Later, if you want, you can activate it and make it visible. 1. In the Announcements list, select the announcement. 2. Click **Deactivate** or **Activate**. A deactivated announcement appears greyed in the Announcements list.
Delete an announcement	1. In the Announcements list, select the announcement. 2. Click **Delete**. 3. In the Delete Announcement dialog box, click **Yes** to confirm.

Manage Project Tags

A project tag is a keyword that you can use to categorize an artifact, such as an issue or a merge request. You can use the tags to search for artifacts. By default, three tags (Plan, Release, and Epic) are available in a project.

1. In the navigation bar, click **Project Administration**  .
2. Click **Tags**.

Action	How To
Create a project tag	1. In the Tags page, click **New Tag**. 2. In **Tag Name**, enter a unique name and press the Enter key. The tag name must contain only letters and numbers.
Rename a tag	1. In the Tags page, select the tag. 2. Type a new name and press the Enter key.
Delete a tag	You can't delete a tag if artifacts refer to it. First, remove all artifacts that refer to the tag or remove the tag from those artifacts, and then remove the tag. In the Tags page, to the right side of the tag name, click **Delete** 

View a Project's Usage Metrics

When you subscribe to DevCS, you're entitled to some storage space on Oracle Cloud. To learn about your storage entitlement, see the Pricing tab on `https://cloud.oracle.com/developer_service`.

You can find out the disk usage metrics of your project's components, such as Git repositories, wikis, and issues.

1. In the navigation bar, click **Project Administration**  .
2. Click **Usage Metrics**.

Manage Repositories

You can manage the Git, Maven, and linked Docker registries from the **Project Administration**: **Repositories** page.

See these links to find out how to manage repositories.

- Create and Manage Git Repositories
- Maven Repository Administration
- Link an External Docker Registry to Your Project

Display RSS/ATOM Feeds

Some websites use ATOM and RSS feeds to publish news feeds. You can subscribe to the RSS/ATOM feeds and configure your project to display them in the Recent Activities feed of the **Project Home** page. All project members can see the feed. You can add any RSS/ATOM feed including Oracle approved RSS Feeds, News, Site monitors, and Jenkins or Hudson servers.

To configure RSS/ATOM feeds, follow these steps:

1. In the navigation bar, click **Project Administration**  .
2. Click **RSS/ATOM Feeds**.

Action	How To
Create an RSS/ATOM Feeds handler	1. On the RSS/ATOM Feeds page, click **+ New Handler**. 2. In **Name**, enter the name of the handler. 3. In **URL**, enter the URL of the feed. 4. From **Display Type**, select the feed's display type. 5. In **Fetch Interval**, enter the feed's fetch interval. By default, the interval is set to 1 day. For the fetch interval period, the feed results are cached. All requests during the interval period retrieve the cached results. When the time expires the cache is cleared. The next request would check for the cached results and not find them and proceed to fetch a new copy to be cached. 6. Click **Done**.
Test a feed handler	1. On the RSS/ATOM Feeds page, select the feed. 2. Click **Test** . 3. Click **Done**. If the test is successful, the status icon changes from **Untested** to **Tested**. If the test fails, the status icon changes to **Failed**.
View logs	1. On the RSS/ATOM Feeds page, select the feed. 2. Click **Logs**. 3. Click **Done**. In the Logs page, all Request and Response logs of each test are available. Select the date-time stamp in the left list of the test to view its logs.
Edit a feed's handler	1. On the RSS/ATOM Feeds page, select the feed. 2. Edit the fields on the left. 3. Click **Done**.
Deactivate or activate a handler	1. On the RSS/ATOM Feeds page, select the feed. 2. Click **Deactivate**. The icon of the feeds handler is greyed out and the **Active** check box is deselected. To activate the feeds handler, click **Activate** or select the **Active** check box. 3. Click **Done**.
Delete a handler	1. On the RSS/ATOM Feeds page, select the feed. 2. Click **Remove** or . 3. In the Remove ATOM/RSS Handler dialog box, click **Yes** to confirm. 4. Click **Done**.

Configure Link Rules

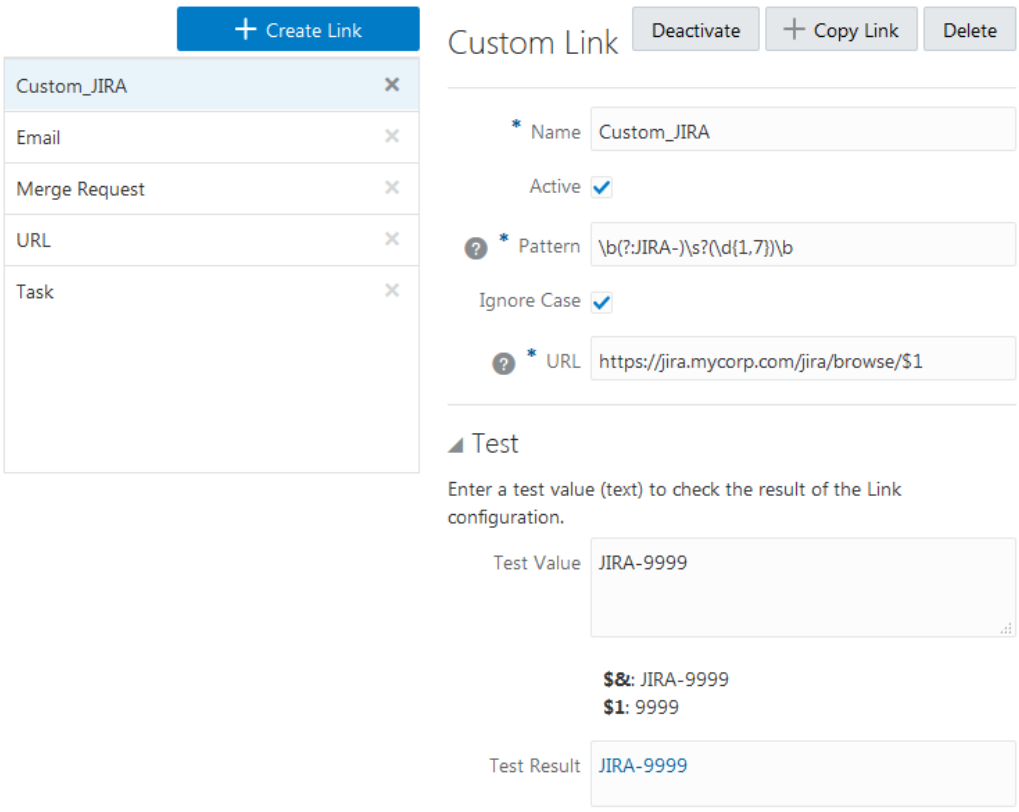
In a project, you can define rules to convert plain text to URL links automatically when the text is entered in commit comments and merge request comments. For example, when you enter an email address or a URL in a merge request comment, it's automatically converted to a link.

To configure link rules, follow these steps:

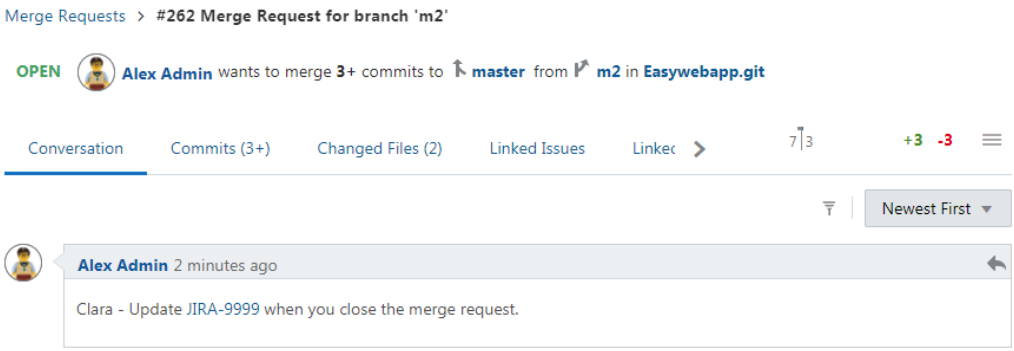
1. In the navigation bar, click **Project Administration** .
2. Click **Links**.

From the Links page, you can create and manage link rules that convert plain text to URL links automatically. You can use Regular Expressions, also called as RegExp, to define the link rules. Some pre-defined built-in link rules are available on the **Links** page. To create a custom rule, you can either copy an existing link rule or create a blank rule. To find more about RegExp, see `http://www.regular-expressions.info`.

This illustration shows an example of a custom link rule.



This illustration shows an example of the link in a merge request comment.



Action	How To
Create a link rule	1. On the Links page, click **+ Create Link**. 2. In **Name**, enter a name. 3. In **Pattern**, enter the RegExp link rule pattern. 4. In **URL**, enter the link URL. You can also use placeholders: • Use `{project}` to insert the project ID. • Use `{organization}` to insert the organization ID. • Use $& to insert the entire matching text, or use $1, $2, $3, and so on to insert text of matched groups. For more information, see `http://www.regular-expressions.info/brackets.html`. 5. To test the rule, expand **Test** and **Test Value**, enter a test value. Verify the result link in **Test Result**. 6. Click **Done**.
Copy a link rule	1. On the Links page, click **+ Copy Link**. 2. On the Create Link page, edit the name, RegExp link rule pattern, and the URL of the link with parameters. You can also use placeholders: • Use `{project}` to insert the project ID. • Use `{organization}` to insert the organization ID. • Use $& to insert the entire matching text, or use $1, $2, $3, and so on to insert text of matched groups. For more information, see `http://www.regular-expressions.info/brackets.html`. 3. To test the rule, expand **Test** and **Test Value**, enter a test value. Verify the result link in **Test Result**. 4. To test the rule, in the Test section, click **Test Value**. Enter a value and verify its result in **Test Result**. 5. Click **Done**.
Edit a link rule	You can't edit a built-in link rule. You can create a copy of built-in rule and edit it, and if required, disable the original pre-configured rule. On the Links page, in the link rule list on the left, select the rule to edit its details on the right.

Action	How To
Activate or deactivate a link rule	If deactivated, the text that matches the rule is not converted to a link. On the Links page, in the link rule list on the left, select the rule. Click **Activate** or **Deactivate**. You can also select or deselect the **Active** check box.
Delete a link rule	You can't delete a built-in rule. On the Links page, in the link rule list on the left, select the rule. On the right side of the page, click **Delete**.

Configure Oracle Maven Repository and SonarQube Connections

Using the Build Administration page, you can configure a project to use build features such as accessing Oracle Maven Repository and using SonarQube Server to analyze build reports.

- Create and Manage Oracle Maven Repository Connections
- Set Up SonarQube

Verify the Organization's Storage Configuration

If you notice any storage issues while archiving artifacts, verify the connection to OCI Object Storage or OCI Object Storage Classic.

1. In the navigation bar, click **Project Administration**  .
2. Click **Storage**.
3. To verify the connection, click **Test Connection**.

   A message is displayed if the connection is successful or fails.

10

Use IDEs

This chapter provides documentation about Oracle Developer Cloud Service integration with Eclipse IDE, Oracle JDeveloper, and NetBeans IDE.

Using the Oracle Developer Cloud Service integration, you can access and update application source code files, manage issues, and monitor builds of your applications from the IDEs.

Eclipse IDE

You can use the Eclipse IDE and the Oracle Enterprise Pack for Eclipse (OEPE) to access Oracle Developer Cloud Service (DevCS) projects, its Git repositories, issues, and builds.

You use the Oracle Cloud view of the Eclipse IDE to access projects. The IDE uses Mylyn (a plugin) to access issues and EGit to access Git repositories.

Sign In to DevCS from the Eclipse IDE

To access DevCS from Eclipse IDE, set up the IDE and create an Oracle Cloud connection.

Set Up the Eclipse IDE

1. In the Eclipse IDE's installation directory, open `eclipse.ini` in a text editor.
2. Add these code lines at the end of the file.

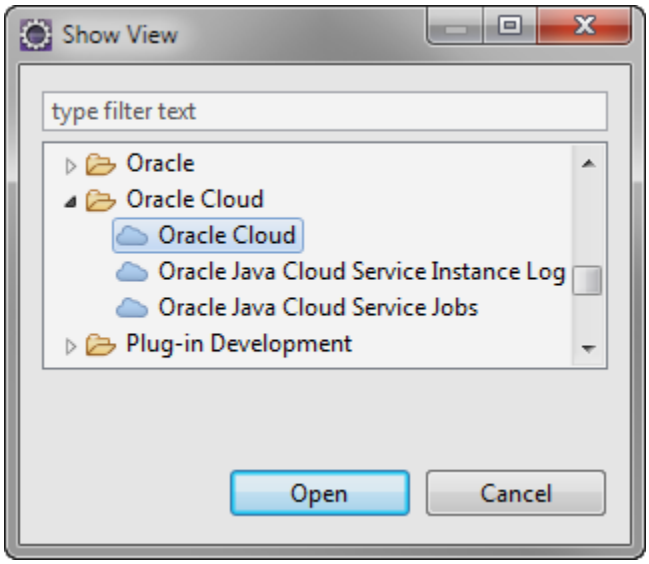
   ```
   -DOracle.Cloud.Debug=true
   -DOracle.Cloud.Dcs.Debug=true
   ```

3. Save the file.

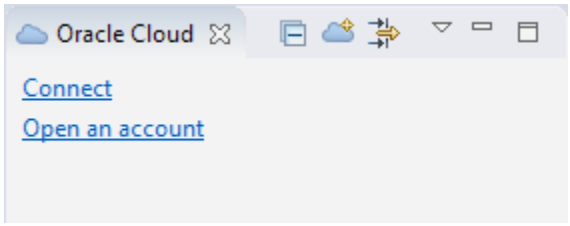
Connect to DevCS from the Eclipse IDE

1. Open the Eclipse IDE.
2. From the **Window** menu, select **Show View** and then select **Oracle Cloud**.

   If the **Oracle Cloud** option doesn't appear in the **Show View** menu, click **Other**. In the Show View dialog box, expand **Oracle Cloud** and select **Oracle Cloud**.



3. If you're connecting to DevCS for the first time, in the Oracle Cloud view, click **Connect** or **New Cloud Connection** .



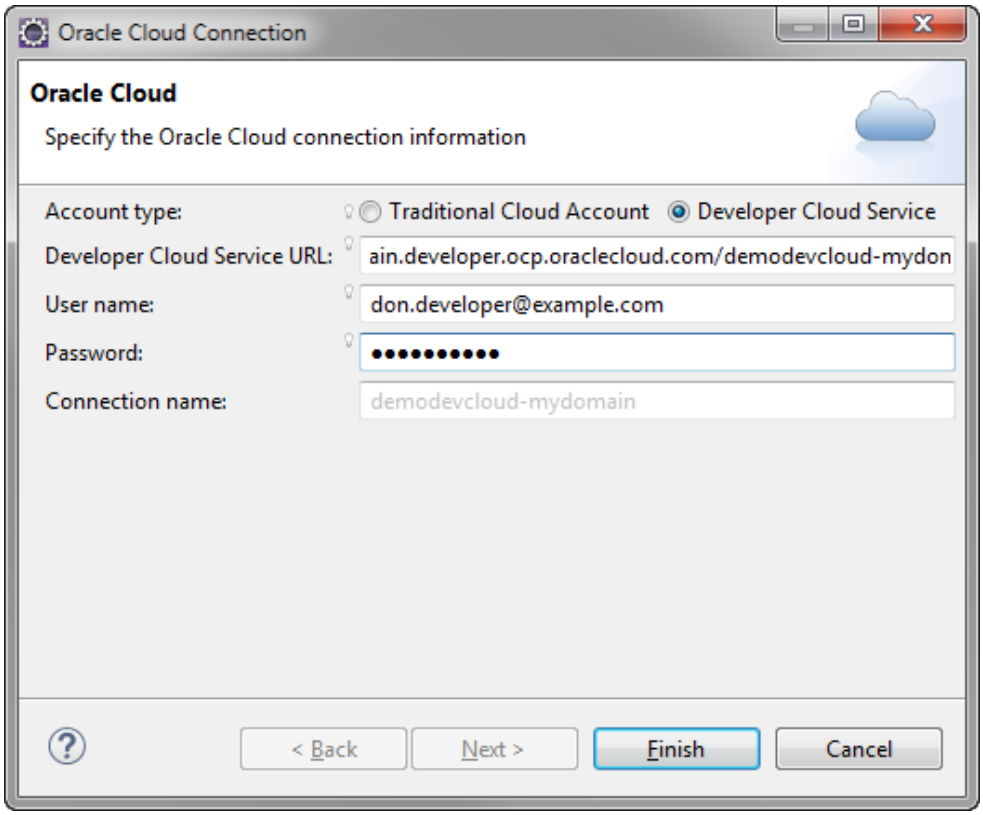
4. In the Oracle Cloud Connection dialog box, select your account type.

   If your account is a traditional account, select **Traditional Cloud Account**. If you're using an IDCS account, select **Developer Cloud Service**.

5. If you selected **Traditional Cloud Account**, enter your identity domain.

   If you selected **Developer Cloud Service**, enter the DevCS URL in the `https://<hostname>.oraclecloud.com/<org-name>/` format.

6. In **Username** and **Password**, enter your Oracle Cloud user name and password.



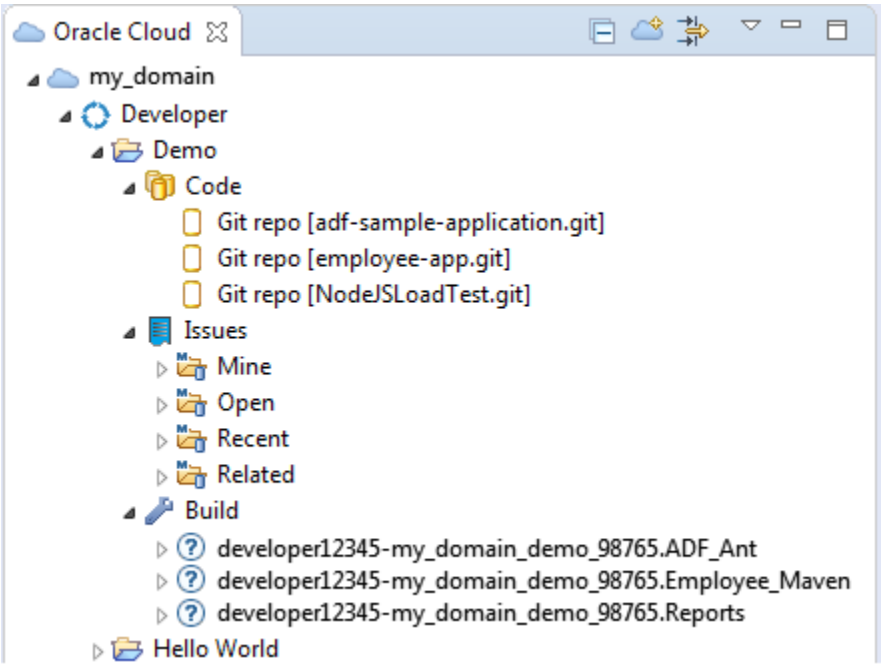
7. Click **Finish**.

After the connection is successful, expand the identity domain node in the Oracle Cloud view, and then expand the **Developer** node to view projects that are assigned to you.

> **Tip:**
>
> The Oracle Cloud view is visible in the Java and Java EE perspective, by default. To open the Java EE perspective, from the **Window** menu, select **Perspective**, then **Open Perspective**, and then **Java** or **Java EE**.

Use the Oracle Cloud View

The Oracle Cloud view shows all projects of which you're a member and provides links to access DevCS features. Some of these links open the DevCS web page, while others open the native IDE interface.



You can perform these actions from the Oracle Cloud view.

Action	How To
Open the Oracle Cloud portal	Select the identity domain node, right-click, and select **Open Cloud Portal**.
Open the Identity Manager console	Select the identity domain node, right-click, and select **Open IDM Console**.
Edit the connection's properties	Select the identity domain node, right-click, and select **Properties**.
Open the DevCS page in a web browser	Select the **Developer** node, right-click, and select **Open Service Console**.
View all project	By default, the Oracle Cloud view shows the projects which you're a member of. To see all projects, select the **Developer** node, right-click, and select **Show All Projects**.

Action	How To
Open a project	Expand the **Developer** node and double-click the project. • To view Git repositories of the project, expand or double-click the **Code** node. • To view issues of the project, expand or double-click the **Issues** node. • To view builds of the project, expand or double-click the **Build** node.

Open a DevCS Project as an Eclipse IDE Project

You can open a DevCS project in the Eclipse IDE and add or edit the files of the project's Git repositories from the IDE. To access files of a DevCS project Git repository, you must clone the repository to the IDE workspace.

1. In the Oracle Cloud view, expand the **Developer** node and double-click the project name.

   You can also right-click the project and select **Activate**.
2. Expand the **Code** node.
3. Double-click the Git repository of the DevCS project that you want to clone. You can also right-click the Git repository and select **Activate**.

   Activating a Git repository clones it to the IDE's workspace. If the project Git repository contains more than one branch, the Clone Git Repository dialog box opens where you can select branches to clone. By default, all branches are selected. Deselect the check box of the branch not to clone. Select the initial branch in **Initial Branch** and click **OK**.

   After the repository is cloned, expand the Git repository in the Oracle Cloud view to view its branches, references, remote repository names, and tags.

   The cloned Git repositories are also visible in the Git Repositories view of the IDE. To open it, from the **Window** menu, select **Show View**, and then select **Git Repositories**.
4. Right-click the Git repository and select **Import Projects**.
5. In the **Import Projects** page of the Import Projects from File System or Archive, verify the details, and click **Finish**.

After cloning the repository, Eclipse IDE displays the project in the Project Explorer view.

Upload an Eclipse IDE Project to DevCS

You can upload local Eclipse project application source code files to an empty Git repository of a DevCS project.

Before you proceed, create an empty Git repository in the project where you want to upload files. You must be a project owner to create the Git repository.

Action	How To
Upload the Eclipse IDE project using drag-and-drop action	1. In the Eclipse IDE, create or open the IDE project. 2. In the Oracle Cloud view, open the DevCS project and expand the **Code** node. 3. From the Project Explorer view, drag the IDE project and drop it to the empty Git repository node. 4. In the Synchronize dialog box, verify the details, select any additional options, and click **Finish**. Wait for some time to upload the files. Eclipse IDE adds all files of the IDE project to the Git repository index, commits all files to branches, and pushes the commits to the DevCS project's Git repository. Open the **Git** page of the DevCS project in the web browser and verify the files.
Use Git actions to upload the Eclipse IDE project	1. In the Eclipse IDE, open the project in the Oracle Cloud view. 2. Expand the **Code** node. 3. Select the empty Git repository to clone, right-click, and select **Activate**. 4. Create or open the IDE project that you want to export. 5. Select the project and then from the **Project** menu, select **Share Project**. You can also right-click the project, select **Team**, and then select **Share Project**. 6. In Project Explorer view, select the project, right-click, select **Team**, and then select **Commit**. 7. In the Git Staging view, click **Add All**  to add all files to the stage index, enter a commit description, and click **Commit and Push**. 8. In the Push Results dialog box, click **Close**. Open the **Git** page of the DevCS project in the web browser and verify the files.
Upload a project that's saved in a local Git repository to the DevCS project	1. In the Eclipse IDE, open the project in the Oracle Cloud view. 2. Select the IDE project, right-click, select **Team**, then select **Remote**, and then **Push**. 3. In the Push Branch dialog box, enter the DevCS project's empty Git repository's URL. 4. In **Authentication**, enter your credentials, if necessary. 5. Click **Next**. 6. Follow the steps of the wizard, verify the details, and click **Finish**. Open the **Git** page of the DevCS project in the web browser and verify the files.

Use Git in the Eclipse IDE

Using Git tools in the Eclipse IDE, you can create and merge branches, commit files, and push commits to the project Git repository.

Eclipse IDE uses EGit to manage Git repositories. To know more about EGit, see `http://eclipse.org/egit/` and the EGit documentation at `http://wiki.eclipse.org/EGit/User_Guide`.

You can view and manage Git repositories from the Oracle Cloud view and the Git Repositories view (**Window** > **Show View** > **Other**). The Git Repositories view shows activated repositories only.

This table describes some common actions you can perform to manage Git repositories.

Action	How To
View Git repositories of a project	In the Oracle Cloud view, expand the **Code** node.
Access branches of a project Git repository	In the Oracle Cloud view, double-click the Git repository and expand the **Branches** node to see local and remote branches.
Checkout a branch	Right-click the **Branches** node, select **Switch To**, and then select the branch name.
Create a branch	Right-click the **Branches** node, select **Switch To**, and then select **New Branch**. In the Create Branch dialog box, enter a branch name, configure other options, and click **Finish**.
Add a file to Git index	In the File Explorer, right-click the file, select **Team**, and the **Add to Index**.
Commit files	Right-click the **Branches** node and select **Commit**. In the Git to Stage view, verify files and stage them, enter a commit message, and click **Commit**.
Push commits to the project's Git repository	Right-click the repository and select **Push** or **Push branch**. In the Push dialog box, verify the details, specify any additional configuration, and submit.

Manage DevCS Issues in the Eclipse IDE

In the Eclipse IDE, you can manage DevCS issues, create issues, update issues, and create a search query. Eclipse uses Mylyn, a built-in plugin, to manage issues.

To know more about Mylyn, see `https://wiki.eclipse.org/Mylyn/User_Guide`.

To view and update DevCS issues of a project, you must activate the **Tasks** node. You can double-click the **Tasks** node, or right-click and select **Activate**.

This table describes common actions you can perform to manage issues from the Eclipse IDE.

Action	How To
View issue filters of the project	Expand the **Tasks** node. By default, the Tasks node displays issues of these filters: • **Mine**: Lists all issues assigned to you • **Open**: Lists all open issues • **Recent**: Lists all recently changed issues • **Related**: Lists all issues related to you
See issues of a query	Expand the **Tasks** node and then expand the issue query.

Action	How To
Open and update an issue	Expand the **Tasks** node, expand the query, and double-click the issue. The issue's details display in a new tab of the IDE. Update the issue's properties and click **Submit**.
Create an issue	Right-click the **Issues** node and select **New Issue**. Update the details and click **Submit**. The issue is automatically synchronized with the DevCS project is available to other project members.
Create a query	Right-click the **Issues** node and select **New Query**. Enter the search criteria and click **Finish**. Issue matching the search criteria display under the new query node in the Oracle Cloud view.
Synchronize a query	To manually synchronize a query, get new issues or updates from DevCS, or send issue updates from the IDE, expand the **Tasks** node, select the query, right-click, and select **Synchronize**.

Associate an Issue with a Commit

When you commit files, you can associate an issue with a commit. To associate the issue, you must activate it first.

Eclipse IDE provides task integration with Mylyn. Activating an issue enables Mylyn to track files related to the current issue. Mylyn automatically hides files that aren't related to the active issue. When you commit changes to the Git repository, the commit message automatically references the active issue's ID. This effectively creates a link between the code commit and the issue, allowing for easy traceability. Links between source commits and issues are also reflected in DevCS.

1. Expand the **Tasks** node.
2. Double-click the issue you want to associate with the commit and update it.
3. In the top-left corner of the issue tab, click **Activate** .
4. Commit your files to the Git repository.

   The issue's name and its URL are automatically included in **Commit Message**.

After a successful commit, open the issue in DevCS. The SHA-1 checksum hash of the commit is available in **Commits** under **Associations**.

Monitor a Project's Builds in Eclipse IDE

You can view your project build jobs from the Build node in the Oracle Cloud view. You can't create, edit, or delete a job from the Oracle Cloud view.

To view build jobs of a project, you must activate the **Build** node. You can double-click the **Build** node, or right-click and select **Activate**. When activated, the Build node shows all jobs and builds of your project. There are no filters available.

This table describes common actions you can perform to manage builds from the Eclipse IDE.

Action	How To
View jobs of the project	Expand the **Build** node.

Action	How To
View builds of a job	Expand the **Build** node and then expand the job. The job node displays its builds along with their status.
See a build's details	Expand the **Build** node, expand the job, select the build, right-click, and select **Open**. The build's details along with its status, artifacts, test results, console, and SCM changes log display in a new tab of the IDE.
Run a build of a job	Expand the **Build** node, select the job, right-click, and select **Run build**. You may want to refresh the job in the Build node to see the running build. Right-click the job node and select **Refresh**. You can also open a build and click **Run Build**.
Abort a running build	Double-click the build node to open its details in the tab and click **Abort Build**.
Open the Build page in a web browser	Right-click the **Build** node and select **Open in Browser**.

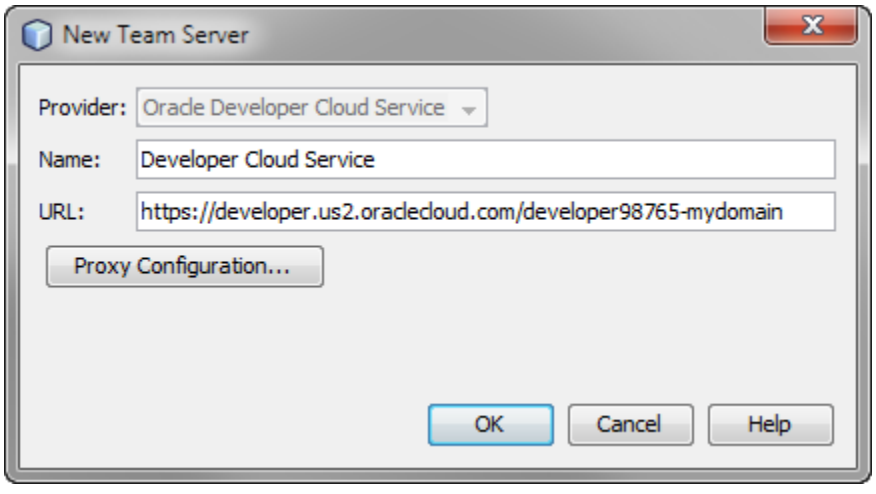
NetBeans IDE

NetBeans IDE is closely integrated with Oracle Developer Cloud Service (DevCS). You can create and open DevCS projects in NetBeans IDE, update files and commit them to the Git repository, create and update issues, and manage project builds.

Sign In to DevCS from the NetBeans IDE

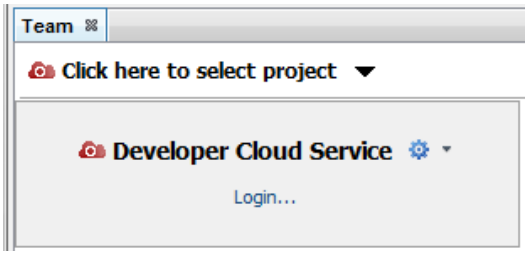
You use the Team window to sign in to DevCS.

1. Open the NetBeans IDE.
2. From the **Team** menu, select **Team Server**, then **Add Team Server**.
3. In the New Team Server dialog box, enter a unique name in **Name** and the DevCS URL in **URL**.



4. Click **OK**.
5. From the **Window** menu, select **Team**.

6. In the Team window, click **Click here to select project** to see the list of team servers. Click **Login** under the DevCS team server.

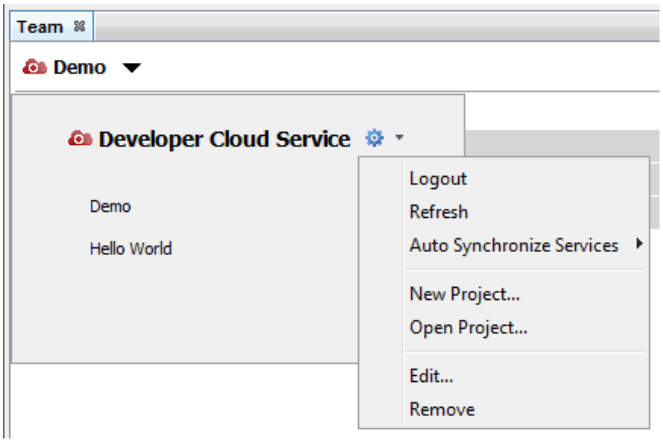


7. In the Login to Team Server dialog box, enter the user name and password, and click **Login**.

Use the Team Server

From the Team tab, you can access DevCS projects, create a project, search a project, access its Git repositories, issues, and builds.

Using the **Configure** menu, you can create or open a project, or logout of DevCS.



Action	How To
Open the project's dashboard	The project's dashboard lists the recent activities and builds of the project. To open the Dashboard, in the Team tab, click **Dashboard**.
Open the project in the web browser	In the Team tab, click **Project Web**.
Create a DevCS project	Click **Configure** and select **New Project**. Follow the steps of the wizard to create the project.
Open a project	Click **Configure** and select **Open Project**. To switch to another project from an open project, you can also click the project name and select another project's name from the list.

By default, NetBeans IDE synchronizes information from all recently selected or opened projects. If you want to synchronize information for the selected project, click

the **Configure**, select **Auto Synchronize Services**, and then **Selected Project Only**.

Create a DevCS Project in NetBeans IDE

You can create a DevCS project in NetBeans IDE and then push it to Oracle Cloud. You can also push an existing NetBeans IDE project to an empty repository of a DevCS project Git repository.

1. Open the NetBeans IDE.
2. From the **Team** menu, select **Team Server**, select *DevCS team server*, and then click **New Project**.

   You may also select **New Project** from the DevCS team server **Configure** menu.
3. In the Name and License page of the New Project wizard, perform these steps:
   a. In **Project Name**, enter a unique project name.
   b. In **Description**, enter a brief description of the project.
   c. In **Security**, select the project's privacy.
   d. In **Wiki Markup**, select the wiki markup language.
4. Click **Next**.
5. In the Source Code and Issues page of the New Project wizard, specify the directory of source code files.

   If you haven't created an IDE project, specify the directory on your computer to use as project's Git repository. In **Local Repository Folder**, click **Browse** and select the local directory.

   If you've an IDE project that you want to upload to the DevCS project, specify the IDE project's directory. In **Folders or Projects**, click **Add Project**, browse, select the project directory, and click **Open Project**. If you don't want to add an existing project or directory, leave the **Folders or Projects** field blank.
6. Click **Next**.
7. In the Summary page, review the information and click **Finish**.
8. After the project is successfully created in DevCS, a dialog box displays a message showing the local repository location. Click **Close** to close the dialog box.

Open a DevCS Project in NetBeans IDE

1. Open the project in the Team window.

   To search for a project, from the **Team** menu, select **Team Server**, select the DevCS server, and then select **Open Project**. You can also click **Configure** and select **Open Project**.
2. In the open project, expand **Sources** and click the **get** link of the Git repository you want to clone.
3. In the Get Sources from Team Server dialog box, verify the Git repository URL and click **Get From Developer Server**.

4. In the Remote Repository page of the Clone Repository wizard, verify the repository URL, and click **Next**.

   The user name and password are automatically picked from the Team Server you're logged into and the wizard automatically moves to the Remote Branches page. To verify the details, click **Back**.

5. If required, in the Remote Branches page of the Clone Repository wizard, select branches to clone and click **Next**.

6. In the Destination Directory page of the Clone Repository wizard, perform these steps:

   a. In **Parent Directory**, specify the directory path.

   b. In the **Clone Name**, enter the name of the cloned repository.

   c. In **Checkout Branch**, select the default branch to checkout.

   d. In **Remote Name**, if necessary, enter or update the remote name of the repository.

   e. If you want NetBeans IDE to scan the cloned project for any NetBeans IDE metadata, select the **Scan for NetBeans Projects after Clone** check box. If metadata is found, the project automatically opens in the Projects window of NetBeans IDE.

7. Click **Finish**.

All cloned Git repositories are available in the Git Repository Browser. To open the Git Repository browser, from the **Team** menu, select **Git**, then **Repository Browser**.

Use Git in NetBeans IDE

Using the Team menu or the Git Repositories Browser, you can commit files, create and merge branches, and push changes to the hosted Git repository.

To know more information about the Git commands and actions in NetBeans IDE, refer to the NetBeans IDE Git documentation at `https://netbeans.org/kb/docs/ide/git.html`.

This table describes some common actions you can perform to manage Git repositories from the NetBeans IDE.

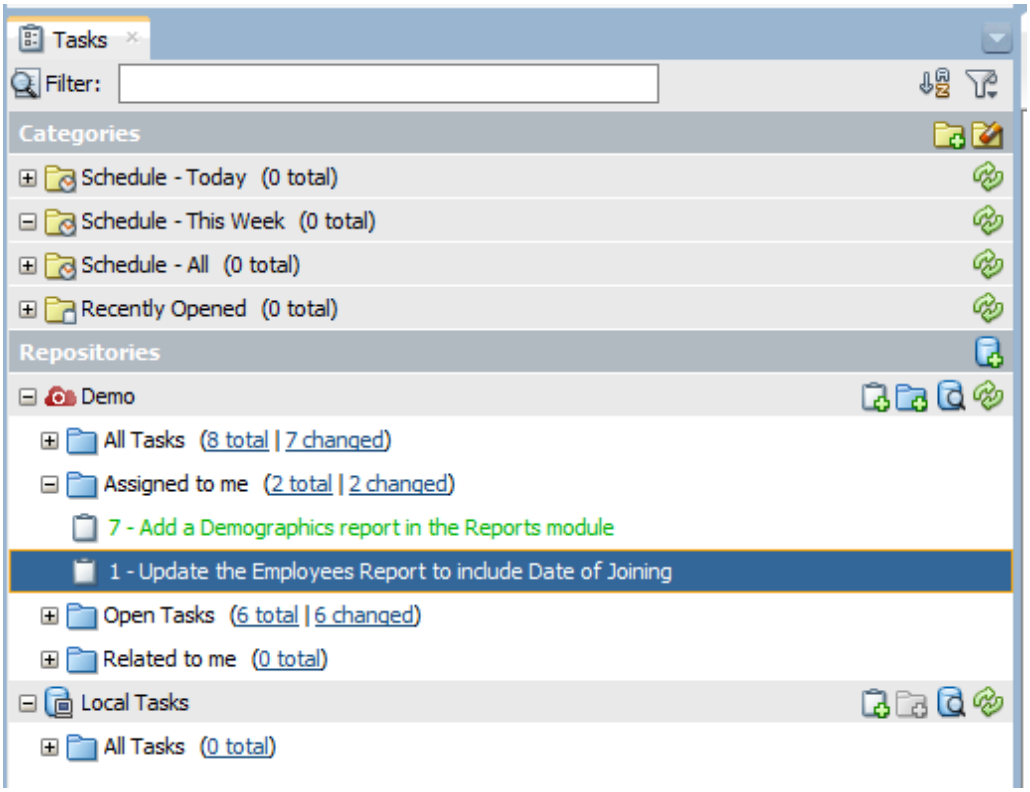
Action	How To
View Git repositories of a project	In the Team tab, expand the **Sources** node.
Access branches of a cloned project Git repository	From the **Team** menu, select **Repository**, and then select **Repository Browser**. Expand the repository node and then the **Branches** node to see local and remote branches.
Checkout a branch	Expand the **Branches** node, right-click the branch, and select **Checkout Revision**.
Create a branch	Expand the **Branches** node, right-click the base branch, and select **Create Branch**. In the Create Branch dialog box, enter a branch name, configure other options, and click **Create**.
Add a file to index	In the **Files** tab, select the file. From the **Team** menu, select **Add**.

Action	How To
Commit files	Select the IDE project in the **Files** tab and then from the **Team** menu, select **Commit**. In the Commit dialog box, enter a message, verify the commit files, author and commiter, and then click **Commit**.
Push commits to the project's Git repository	Select the IDE project in the **Files** tab. From the **Team** menu, select **Remote**, and then select **Push**.

Manage DevCS Issues in the NetBeans IDE

In the NetBeans IDE, you can create and update DevCS issues and search queries.

NetBeans IDE displays tasks in the Tasks tab. To open the tab, from the **Window** menu, select **Tasks**.



This table describes common actions you can perform to manage issues from the NetBeans IDE.

Action	How To
View issues of the project	In the **Tasks** tab, expand **Repositories**. In the project name's node, expand the filter to view its issues. By default, the Tasks tab displays issues of these filters: • All tasks • Assigned to me • Open tasks • Related to me • User-defined custom queries
Open and update an issue	In the Tasks tab, double-click the issue. The issue's details display in a new tab of the IDE. Update the issue's properties and click **Submit**.
Create an issue	In the Tasks tab, select the project node, right-click, and select **Create Task**. Enter the details and click **Submit**. The issue is automatically synchronized with the DevCS project is available to other project members.
Create a query	In the Tasks tab, select the project node, right-click, and select **Create Query**. In the Find Tasks tab, enter the search criteria and click **Save Query**.

Associate an Issue with a Commit

When you commit files, you can associate an issue with a commit.

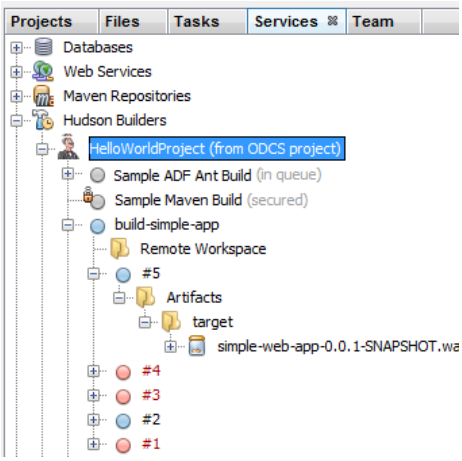
1. In the Tasks tab, open and update the issue that you want to associate with the commit.
2. From the **Team** menu, select **Commit**.
3. In the Commit dialog box, enter the commit's message and verify the files.
4. In the **Update Task** section, select the issue from the **Task** list.
5. Select the desired resolution check boxes and the commit option.
6. Click **Commit**.

Open the issue in DevCS and verify the SHA-1 checksum hash of the commit is now available in **Commits** under **Associations**.

Monitor a Project's Builds in NetBeans IDE

You can view your project build jobs from the Build node in the Team tab. You can't create, edit, or delete a job from the Team tab.

To view a particular build's details, double-click the job name in the Team tab to view its builds in the Services tab. You can also open the Services tab from the Window menu.



This table describes some common actions you can perform to manage builds from the Services tab of the NetBeans IDE.

Action	How To
View jobs of the project	In the Services tab, expand the **Hudson Builders** node, and then expand the project node.
View builds of a job	Expand the job. The job node displays its builds along with their status.
See a build's artifacts and console log	Expand the build node to view its artifacts. To view the console log, right-click the build node, and select **Show Console**. To see the SCM log, select **Show Changes**.
Run a build of a job	Select the job, right-click, and select **Start Job**.

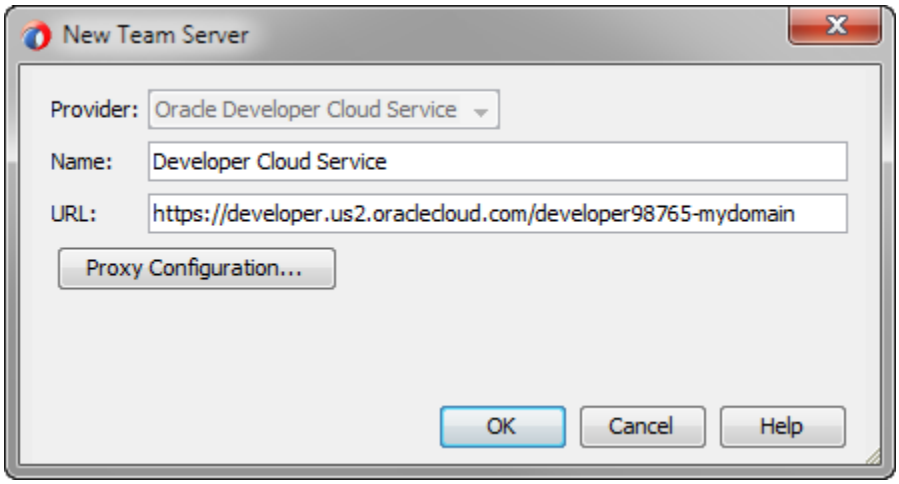
JDeveloper

Oracle JDeveloper enables you to develop applications and commit the files to the Oracle Developer Cloud Service Git repository. JDeveloper also uses the Team Server plugin to access DevCS projects. The plugin is installed by default in JDeveloper.

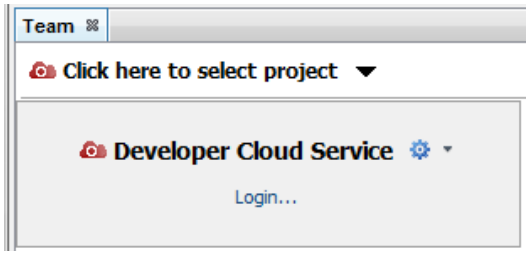
Sign In to DevCS from JDeveloper

You use the Team window to sign in to DevCS.

1. Open JDeveloper.
2. From the **Team** menu, select **Team Server**, then **Add Team Server**.
3. In the New Team Server dialog box, enter a unique name in **Name** and the DevCS URL in **URL**.



4. Click **OK**.
5. From the **Window** menu, select **Team**.
6. In the Team window, click **Click here to select project** to see the list of team servers. Click **Login** under the DevCS team server.

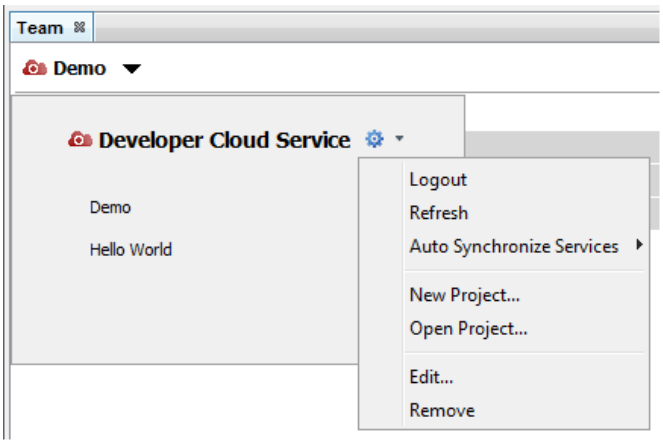


7. In the Login to Team Server dialog box, enter the user name and password, and click **Login**.

Use the Team Server

From the Team tab, you can access DevCS projects, create a project, search a project, access its Git repositories, issues, and builds.

Using the **Configure** menu, you can create or open a project, or logout of DevCS.



Action	How To
Open the project's dashboard	The project's dashboard lists the recent activities and builds of the project. To open the Dashboard, in the Team tab, click **Dashboard**.
Open the project in the web browser	In the Team tab, click **Project Web**.
Create a DevCS project	Click **Configure** and select **New Project**. Follow the steps of the wizard to create the project.
Open a project	Click **Configure** and select **Open Project**. To switch to another project from an open project, you can also click the project name and select another project's name from the list.

Create a DevCS Project in JDeveloper

You can create a DevCS project in JDeveloper and then push it to Oracle Cloud. You can also push an existing JDeveloper project to an empty repository of a DevCS project Git repository.

1. Open JDeveloper.
2. From the **Team** menu, select **Team Server**, select *DevCS team server*, and then select **New Project**.

   You may also select **New Project** from the DevCS team server **Configure** menu.
3. In the Name and License page of the New Project wizard, perform these steps:
   a. In **Project Name**, enter a unique project name.
   b. In **Description**, enter a brief description of the project.
   c. In **Security**, select the project's privacy.
   d. In **Wiki Markup**, select the wiki markup language.
4. Click **Next**.
5. In the Source Code and Issues page of the New Project wizard, specify the directory of source code files.

   If you haven't created an IDE project, specify the directory on your computer to use as project's Git repository. In **Local Repository Folder**, click **Browse** and select the local directory.

   If you've an IDE project that you want to upload to the DevCS project, specify the IDE project's directory. In **Folders or Projects**, click **Add Project**, browse, select the project directory, and click **Open Project**. If you don't want to add an existing project or directory, leave the **Folders or Projects** field empty.
6. Click **Next**.
7. In the Summary page, review the information and click **Finish**.
8. After the project is successfully created in DevCS, a dialog box displays a message showing the local repository location. Click **Close** to close the dialog box.

Open a DevCS Project in JDeveloper

You can open an DevCS project in JDeveloper and update its source code files.

1. Open the project in the Team window.

   To search for a project, from the **Team** menu, select **Team Server**, select the DevCS server, and then select **Open Project**. You can also click **Configure** and select **Open Project**.

2. In the open project, expand **Sources** and click the **get** link of the Git repository you want to clone.
3. In the Get Sources from Team Server dialog box, verify the Git repository URL and click **Get From Developer Server**.
4. In the Remote Repository page of the Clone Repository wizard, verify the repository URL, and click **Next**.

   The user name and password are automatically picked from the Team Server you're logged into and the wizard automatically moves to the Remote Branches page. To verify the details, click **Back**.

5. If required, in the Remote Branches page of the Clone Repository wizard, select branches to clone and click **Next**.
6. In the Destination Directory page of the Clone Repository wizard, perform these steps:
   a. In **Parent Directory** , specify the directory path .
   b. In the **Clone Name** , enter the name of the cloned repository.
   c. In **Checkout Branch**, select the default branch to checkout.
7. Click **Finish**.

Use Git in JDeveloper

Using the Team menu or the Versions tab, you can commit files, create and merge branches, and push changes to the hosted Git repository.

For more information about Git actions in JDeveloper, read the *Versioning Applications with Source Control* chapter of *Oracle Fusion Middleware Developing Applications with Oracle JDeveloper* on `http://docs.oracle.com/middleware`.

Watch a short video to learn more about using Git in Oracle JDeveloper.

Video

This table describes some common actions you can perform to manage Git repositories from the NetBeans IDE.

Action	How To
View Git repositories of a project	In the Team tab, expand the **Sources** node.
Access branches of a cloned project Git repository	From the **Team** menu, select **Versions**. Expand the repository node and then the **Branches** node to see local and remote branches.

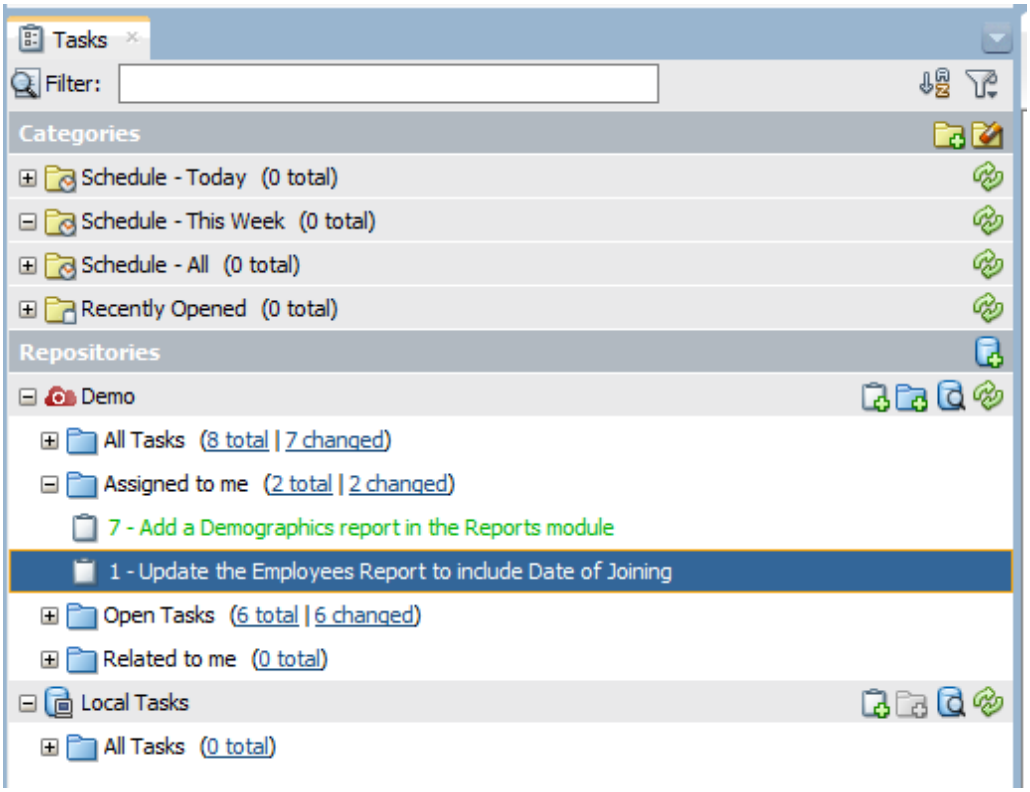
Action	How To
Checkout a branch	Expand the **Branches** node, right-click the branch, and select **Checkout**.
Create a branch	Expand the **Branches** node, right-click the base branch, and select **Create Branch**. In the Create Branch dialog box, enter a branch name, configure other options, and click **OK**.
Add a file to index	In the **Applications** tab, select the file. From the **Team** menu, select **Git**, and then select **Add**.
Commit files	Select the IDE project in the **Applications** tab and then from the **Team** menu, select **Git**, and then select **Commit**. In the Commit dialog box, enter a message, verify the commit files, author and commiter, and then click **Commit**.
Push commits to the project's Git repository	Select the IDE project in the **Applications** tab. From the **Team** menu, select **Git**, and then select **Push**.

You can download the Git extension from the **Help** menu. Open the **Help** menu and select **Check for Updates**. See the JDeveloper documentation for more information.

Manage DevCS Issues in JDeveloper

In JDeveloper, you can manage DevCS issues, create issues, update issues, and create a search query.

JDeveloper displays tasks in the Tasks tab. To open the tab, from the **Window** menu, select **Tasks**.



This table describes some common actions you can perform to manage issues from JDeveloper.

Action	How To
View issues of the project	In the **Tasks** tab, expand **Repositories**. In the project name's node, expand the filter to view its issues. By default, the Tasks tab displays issues of these filters: • All tasks • Assigned to me • Open tasks • Related to me • User defined custom queries
Open and update an issue	In the Tasks tab, double-click the issue. The issue's details display in a new tab of the IDE. Update the issue's properties and click **Submit**.
Create an issue	In the Tasks tab, select the project node, right-click, and select **Create Task**. Enter the details and click **Submit**. The issue is automatically synchronized with the DevCS project is available to other project members.
Create a query	In the Tasks tab, select the project node, right-click, and select **Create Query**. In the Find Tasks tab, enter the search criteria and click **Save Query**.

Associate an Issue with a Commit

When you commit files, you can associate an issue with a commit.

1. In the Tasks tab, open and update the issue that you want to associate with the commit.
2. From the **Team** menu, select **Commit**.
3. In the Commit dialog box, enter the commit's message and verify the files.
4. In the **Update Task** section, select the issue from the **Task** list.
5. Select the desired resolution check boxes and the commit option.
6. Click **Commit**.

Open the issue in DevCS and verify the SHA-1 checksum hash of the commit is now available in **Commits** under **Associations**.

Monitor a Project's Builds in JDeveloper

The Team tab displays all builds and jobs of the project along with their status.

To view a particular build's details, double-click the job name in the Team tab to view its builds in the web browser.

Build Oracle ADF Applications

You can develop Oracle ADF applications in JDeveloper and then build them using OJMake and OJDeploy tools with Ant or Maven in DevCS.

Build ADF Applications with Ant

Tutorial

1. Create an Oracle ADF application in Oracle JDeveloper.
2. Add or configure the `build.properties` and `build.xml` files.
   - In `build.xml`, add `<property environment="env"/>` immediately after the `<project>` element to enable Ant to access the system environment variables and store them in properties, prefixed with `env`.

     Example:

     ```
     <?xml version="1.0" encoding="windows-1252" ?>
     <project name="HelloWorldADFProject" basedir=".">
       <property environment="env" />
       <property file="build.properties"/>
       ...
     </project>
     ```

     Configure the `build.properties` file to use DevCS build executor environment variables to access the Ant library, the OJDeploy tool, and the JDeveloper directory installed on the DevCS build executor.
3. Commit and push the application to the hosted Git repository.
4. In DevCS, create and configure a job with an Ant build step.
5. Run a build of the job to generate the artifacts.

Use OJServer with OJDeploy

When you run a job configured to use multiple OJDeploy builds, the build starts, runs, and then shuts down JDeveloper `ojdeploy` for each invocation. You can increase the OJDeploy performance by using OJServer, which eliminates the requirement to start and stop OJDeploy after each invocation.

To use OJServer with OJDeploy, you must configure the job and update the Ant script to use OJServer.

1. Configure the `build.xml` file of your application to add the `ojserver` argument (`<arg value="-ojserver"/>`) before parameters are defined.

   Example:

   ```
   <property file="build.properties"/>
     <target name="deploy" description="Deploy JDeveloper profiles">
       <taskdef name="ojdeploy"
                classname="oracle.jdeveloper.deploy.ant.OJDeployAntTask"
                uri="oraclelib:OJDeployAntTask"
                classpath="${oracle.jdeveloper.ant.library}"/>
       <ora:ojdeploy xmlns:ora="oraclelib:OJDeployAntTask"
                     executable="${oracle.jdeveloper.ojdeploy.path}"
                     ora:buildscript="${oracle.jdeveloper.deploy.dir}/ojdeploy-
   build.xml"
                     ora:statuslog="${oracle.jdeveloper.deploy.dir}/ojdeploy-
   statuslog.xml">
         <arg value="-ojserver"/>
         <ora:deploy>
           <ora:parameter name="workspace" value="$
   ```

```
{oracle.jdeveloper.workspace.path}"/>
        <ora:parameter name="profile" value="$
{oracle.jdeveloper.deploy.profile.name}"/>
        </ora:deploy>
    </ora:ojdeploy>
```

2. In DevCS, configure the job to run a Shell build step that starts OJServer before running OJDeploy commands.

   Examples of commands:

   ```
   $ORACLE_HOME_SOA_12_1_3/jdev/bin/ojserver -start &
   ```

   ```
   $ORACLE_HOME_SOA_12_2_1/jdev/bin/ojserver -start &
   ```

   - In the command that runs `ojserver`, add a `&` character at the end of the command to keep `ojserver` running in the background. In the command, use the correct environment variables that matches your JDeveloper 12c or above version. OJServer isn't supported in JDeveloper 11g. For more information about environment variables, see Build Executor Environment Variables.
   - In the Shell build step that starts the `ojserver`, use the `sleep` command to add a 30 seconds or more wait time to allow the `ojserver` process to start before any other command runs.

     Example:

     ```
     $ORACLE_HOME_SOA_12_1_3/jdev/bin/ojserver -start &
     ```

     ```
     sleep 30
     ```

   - Configure the job to use JDK 8 (or above) as JDev 12.2.1 and above versions don't support JDK 7 (or below).

   The `ojserver` process automatically stops with any other remaining processes when the build executor is recycled/cleaned up for the next users job. If you are using a Build VM, you must stop `ojserver` manually by adding another Shell build step at the end that runs after all build commands have run.
   Example: `$ORACLE_HOME_SOA_12_1_3/jdev/bin/ojserver -stop`.

Build ADF Applications with Maven

You can configure the Maven POM file to access the Oracle Maven Repository to access JDeveloper and ADF libraries. You'd also need Oracle SSO credentials to access the Oracle Maven repository.

Tutorial

1. Create an Oracle ADF application in Oracle JDeveloper.
2. Add and configure the Maven POM file of the ADF application. Use the DevCS build executor environment variables to access OJMake, OJDeploy, and the Oracle Maven Repository.
3. Commit and push the application to the hosted Git repository.
4. In DevCS, create and configure a job with a Maven build step.
5. Run a build of the job to generate the artifacts.