



Oracle Cloud Infrastructure Architect Professional Workshop

Lab Practices Guide
D1102590GC10

Learn more from Oracle University at education.oracle.com



Copyright © 2024, Oracle and/or its affiliates.

Disclaimer

This document contains proprietary information and is protected by copyright and other intellectual property laws. The document may not be modified or altered in any way. Except where your use constitutes "fair use" under copyright law, you may not use, share, download, upload, copy, print, display, perform, reproduce, publish, license, post, transmit, or distribute this document in whole or in part without the express authorization of Oracle.

The information contained in this document is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

Restricted Rights Notice

If this documentation is delivered to the United States Government or anyone using the documentation on behalf of the United States Government, the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs (including any operating system, integrated software, any programs embedded, installed or activated on delivered hardware, and modifications of such programs) and Oracle computer documentation or other Oracle data delivered to or accessed by U.S. Government end users are "commercial computer software" or "commercial computer software documentation" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, reproduction, duplication, release, display, disclosure, modification, preparation of derivative works, and/or adaptation of i) Oracle programs (including any operating system, integrated software, any programs embedded, installed or activated on delivered hardware, and modifications of such programs), ii) Oracle computer documentation and/or iii) other Oracle data, is subject to the rights and limitations specified in the license contained in the applicable contract. The terms governing the U.S. Government's use of Oracle cloud services are defined by the applicable contract for such services. No other rights are granted to the U.S. Government.

Trademark Notice

Oracle®, Java, MySQL, and NetSuite are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Inside are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Epyc, and the AMD logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

Third-Party Content, Products, and Services Disclaimer

This documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

1004032024

Table of Contents

Network Security Group (NSG) as an Ingress Rule for Another NSG, Instead of a CIDR Block	5
Get Started	6
Launch Virtual Cloud Network and Compute Instances	8
Create a Nested Network Security Group	11
Design and Implement a Real-World Network Architecture: Configuring Private DNS Zones, Views, and Resolvers	13
Get Started	14
Set Up Lab Environment	16
Create zone-a.local Custom Private Zone	18
Create zone-b.local Custom Private Zone	19
Test Instance for Associated Zones	20
Configure the VCN Resolver Adding the Other Private View	21
Test Instance for Associated Zones	22
Design Cloud-Native, Microservices, and Serverless Architecture: Build and Deploy an Oracle Function	23
Get Started	24
Create a VCN and Functions Application	25
Create a Private Repository in OCIR and Set Up Cloud Shell for Access	26
Build & Deploy the Function Container and Test Function	28
Design Cloud-native, Microservices, and Serverless Architecture: Create an API Gateway Deployment	31
Get Started	32
Create a New API Gateway	34
Create a New API Gateway Deployment.....	35
Validate a Policy Statement That Allows API Gateway to Access the Function	37
Add an Ingress Rule for the Public Subnet.....	38
Call the Function via Your API Gateway Deployment	39
Design Cloud-Native, Microservices, and Serverless Architecture: Manage OCIR and Push and Pull Images Using Docker CLI	41
Get Started	42
Access the Dockerfile	44
Build the Docker Image.....	46
Run Your Docker Image as a Container	47
Access the Web Application Running Within the Container	48

Delete the Docker Container.....	49
Create an Auth Token	50
Create a New Container Repository	51
Sign In to OCIR from the Cloud Shell	52
Tag the Docker Image	54
Push the Tagged Docker Image to OCIR Repository.....	55
Verify if the Image has Been Pushed	56
Pull the Image from OCIR Repository	57
Design Cloud-Native, Microservices, and Serverless Architecture: Deploy a Load-Balanced Web application on an OKE cluster using Kubectl.....	59
Get Started	60
Set Up the <code>kubeconfig</code> File.....	62
Run <code>kubectl</code> Commands Against Kubernetes Clusters	63
Create a Kubernetes (OKE) Secret	67
Add the Secret and the Image Path to the Deployment Manifest.....	69
Deploy the Sample Web Application to OKE Cluster	71
Verify if the Sample Web Application Is Accessible	72
Clean Up the Resources Deployed Within OKE Cluster	74
Infrastructure As Code: Create a Reusable VCN Configuration with Terraform	75
Get Started	76
Create a Terraform Folder and File in Code Editor	78
Create and Destroy a VCN Using Terraform	80
Create and Destroy a VCN Using Resource Manager	84

Network Security Group (NSG) as an Ingress Rule for Another NSG, Instead of a CIDR Block

Lab Practices

Estimated Time: 25 minutes

Mahamat Guiagoussou (mahamat.guiagoussou@oracle.com) has a
non-transferable license to this guide.

Get Started

Overview

In this lab, you will create two Network Security Groups (NSG). The first NSG will be a source to the second one. To demonstrate the power of the NSGs, we will create **four** compute instances, the first three of which will be in a public subnet, and the fourth in a private subnet. All three instances in the public subnet will be SSH enabled, which is a default rule in a Security List. We will enable ICMP Echo on the third instance only via an NSG. From your personal computer in your office, you can attempt to SSH to all three compute instances. You will succeed as the Default Security list allows this by default. Then you will ping the public IP address of all three compute instances. Only the third one will reply because the NSG enabled this.

Then we will create the second NSG, where the first one will be a source. In this NSG you will also enable ICMP echo. You will also assign this second NSG to the compute instance in the private subnet.

Thanks to this feature, you will be able to successfully ping the compute instance in the private subnet from the third compute instance, but not from the first or second. You will prove this by using SSH to access the third compute instance, then the first one. You will leave ping running, and then add the first compute instance to the first NSG. You will immediately see that ping will start working in the first compute instance.

A **Network Security Group (NSG)** acts as a virtual firewall for your compute instances and other kinds of resources. An NSG consists of a set of ingress and egress security rules that apply only to a set of vNICs of your choice in a single VCN (for example: all the compute instances that act as web servers in the web tier of a multi-tier application in your VCN).

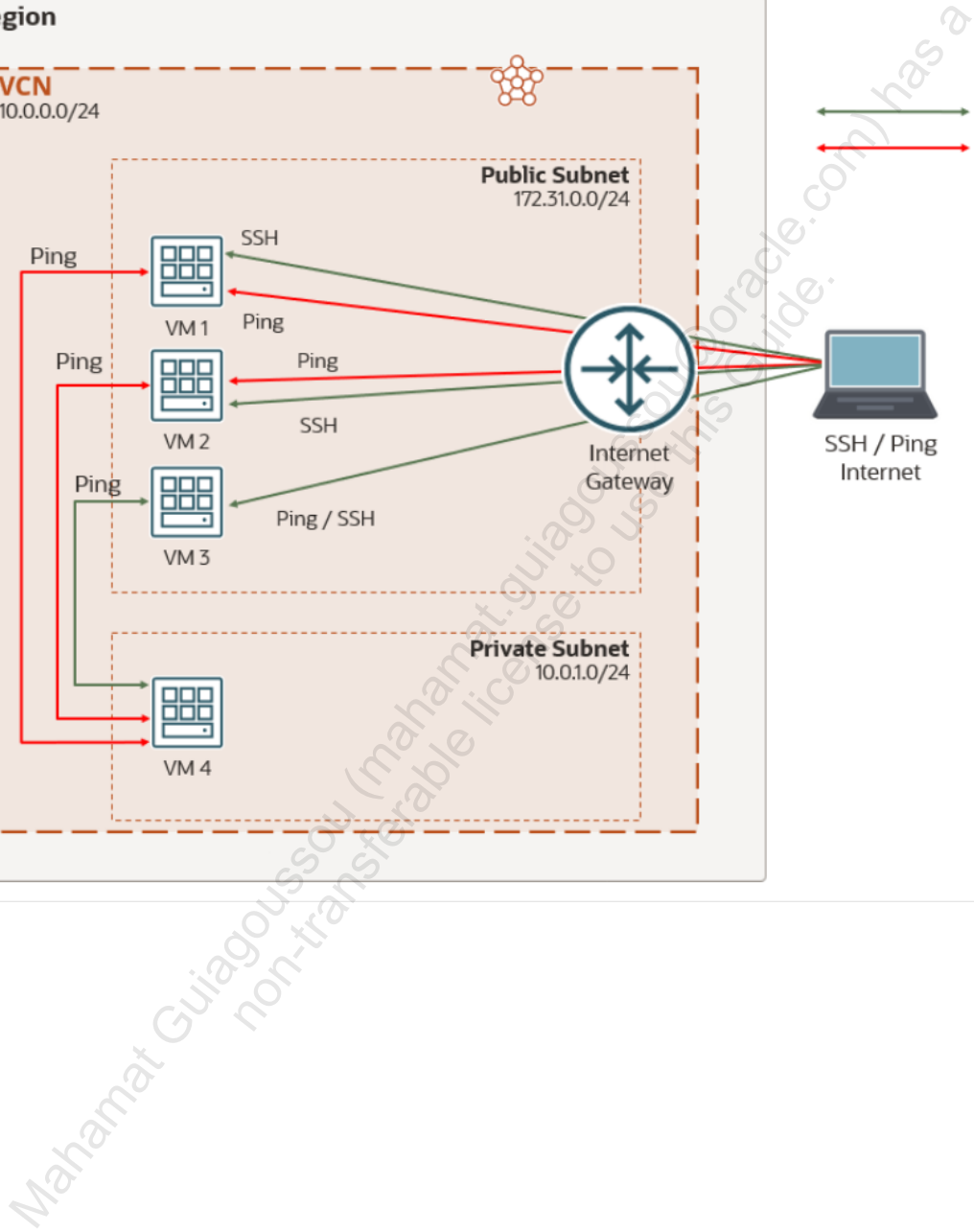
Compared to Security Lists, NSGs let you separate your VCN's subnet architecture from your application security requirements.

You can specify an NSG as the source (for ingress rules) or destination (for egress rules) in a given NSG's security rule. The two NSGs must be in the same VCN.

Currently, the following types of parent resources support the use of NSGs:

- Compute instances
- Load balancers
- DB systems
- Autonomous Databases

- Unauthorized reproduction or distribution prohibited. Copyright© 2024, Oracle University and/or its affiliates.



Launch Virtual Cloud Network and Compute Instances

Create the following resources in the Ashburn region

VCN (Create with the VCN Wizard)

- Name: **IAD-AP-LAB04-1-VCN-01**
- CIDR Block: 10.0.0.0/16
- Public Subnet
- CIDR Block: 10.0.0.0/24
- Private Subnet
- CIDR Block: 10.0.1.0/24

VMs

- **On Public Subnet** (Oracle Linux 8, VM.Standard.A1.Flex)
1. **IAD-AP-LAB04-1-VM-01** (Add SSH keys, with the best option for you)
 2. **IAD-AP-LAB04-1-VM-02** (Add SSH keys, with the best option for you)
 3. **IAD-AP-LAB04-1-VM-03** (Add SSH keys, with the best option for you)
 4. **IAD-AP-LAB04-1-VM-04** (No SSH keys required)
 - **On Private Subnet** (Oracle Linux 8, VM.Standard.A1.Flex)

From your personal computer in your office, ping the public IP addresses of all three compute instances.

They will all fail; the Security List does not have an ICMP Echo rule. Now SSH to all three of them. This will succeed because the Default Security List comes with SSH port 22 enabled by default.

Create the first NSG

1. Log in to your tenancy and compartment on the Cloud Console.
2. Make sure you are in the Ashburn region.
3. In the main menu, in Networking, click **Virtual Cloud Networks**.

4. Select the VCN you just created, **IAD-AP-LAB04-1-VCN-01**.
5. Under **Resources**, click **Network Security Groups**.
6. Click **Create Network Security Group**.
 - a. Name: **IAD-AP-LAB04-1-NSG-01**
 - b. Create in Compartment: <Your Assigned Compartment>
 - c. Click **Next**.
 - d. Direction: **Ingress**
 - e. Source Type: **CIDR**
 - f. Source CIDR: **0.0.0.0/0**
 - g. IP Protocol: **ICMP**
 - h. Type: **8**
 - i. Code: **All**
7. Click **Create**.
8. From the main Menu, under **Compute**, click **Instances**.
9. Click **IAD-AP-LAB04-1-VM-03**.
10. Scroll down to **Resources**, click on **Attached VNICs**.
11. Click **IAD-AP-LAB04-1-VM-03 (Primary VNIC)**.
12. Under **VNIC Information** click on the **Edit** link next to **Network Security Groups**.
13. Select the NSG you just created: **IAD-AP-LAB04-1-NSG-01**.
14. Click **Save Changes**.

You just completed part one of the lab, now let's test it.

1. Go to the Main Menu (Top Right)
2. Under Compute, click Instances
3. Take Note of the three public IP Addresses
4. From your Personal Computer in your office ping all three IP Addresses.
5. **Only** IAD-AP-LAB04-1-VM-03 will reply!

Mahamat Guiagoussou (mahamat.guiagoussou@oracle.com) has a non-transferable license to use this Guide.

Create a Nested Network Security Group

Now SSH again to all three compute instances, and from each one of them ping the Private IP Address of **IAD-AP-LAB04-1-VM-04**. They will all fail. We will now create a nested NSG.

1. Log in to your tenancy and compartment on the Cloud Console.
2. Make sure you are in the Ashburn region.
3. In the main menu, under Networking, click Virtual Cloud Networks.
4. Select VCN IAD-AP-LAB04-1-VCN-01.
5. Under Resources, click Network Security Groups.
6. Click Create Network Security Group.
 - Name: IAD-AP-LAB04-1-NSG-02
 - Create in Compartment: <Your Assigned Compartment>
 - Click on Next
 - Direction: Ingress
 - Source Type: Network Security Group(NSG)
 - Source NSG in <Your Assigned Compartment>: IAD-AP-LAB04-1-NSG-01
 - IP Protocol: ICMP
 - Type: 8
 - Code: All
7. Click **Create**.
8. Go to the Main Menu (Top Right).
9. Under Compute, click **Instances**.
10. Click IAD-AP-LAB04-1-VM-04.
11. Scroll down to **Resources**, and click Attached VNICs.
12. Click IAD-AP-LAB04-1-VM-04 (Primary VNIC).
13. Under VNIC Information, click the Edit link next to Network Security Groups.
14. Select NSG IAD-AP-LAB04-1-NSG-02.
15. Click **Save Changes**.

Now SSH again to all three compute instances, and from each one of them ping the Private IP Address of IAD-AP-LAB04-1-VM-04. Only **IAD-AP-LAB04-1-VM-03** succeeds!

It's because compute instance IAD-AP-LAB04-1-VM-04's vNIC is bound by the rule in NSG IAD-AP-LAB04-1-NSG-02 in addition to the rules of the Default Security List. **Also** because NSG IAD-AP-LAB04-1-NSG-02 has NSG IAD-AP-LAB04-1-NSG-01 as a source, which is the NSG you added to the vNIC of IAD-AP-LAB04-1-VM-03.

You can continue this lab on your own with the two NSG you have. Assign the right one to the vNIC of compute instance IAD-AP-LAB04-1-VM-01.

Mahamat Guiagoussou (mahamat.guiagoussou@oracle.com) has a non-transferable license to use this Guide.

Design and Implement a Real-World Network Architecture: Configuring Private DNS Zones, Views, and Resolvers

Lab Practices

Estimated Time: 30 minutes

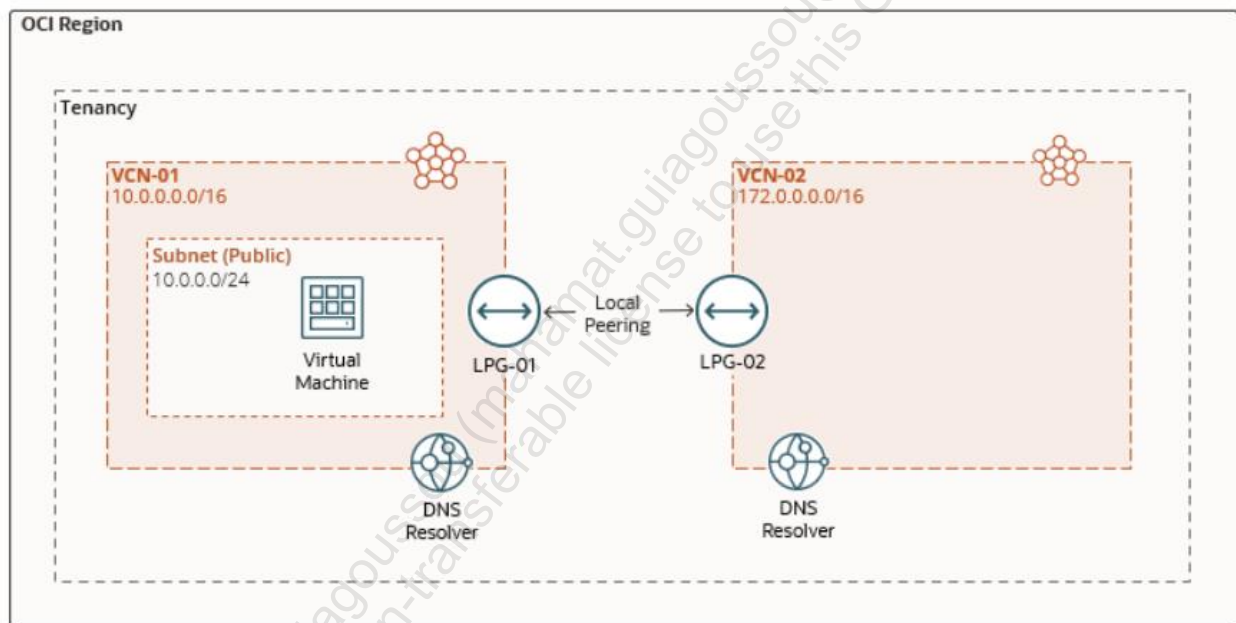
Mahamat Guiagoussou (mahamatguiagoussou@oracle.com) has a
non-transferable license to use this guide.

Get Started

Overview

Customers want to specify their own private DNS domain names to manage their private assets in OCI, as well as support DNS resolution between VCNs and between VCNs and on-premises networks. With private DNS, customers can:

- Create private DNS zones with their desired names and create records for their private resources
- Create a private DNS resolver for DNS resolution to and from other private networks
- Resolve queries for custom private zones and system-generated zones, such as oraclevcn.com
- See DNS views and implement conditional forwarding for split-horizon environments



In this lab, you'll:

- a. Create custom private zones
- b. Configure a VCN resolver
- c. Configure the VCN resolver to add the other private view

Prerequisites

- Required policies have been set up for you.

Mahamat Guiagoussou (mahamat.guiagoussou@oracle.com) has a non-transferable license to use this Guide.

Set Up Lab Environment

In order to do this lab, you'll first need to create two peered VCNs and an instance.

Create Two VCNs and a Subnet

1. Log in to your tenancy and compartment on the Cloud Console.
2. Make sure you are in the Ashburn region.
3. In the navigation menu, navigate to Networking, and click **Virtual Cloud Networks**.
4. Confirm that you are in the proper compartment at the left.
5. Create the First VCN. Click **Start VCN Wizard**
 - a. **VCN Name:** IAD-AP-LAB06-1-VCN-01
 - b. Make sure your compartment is selected.
 - c. **VCN IPv4 CIDR block:** 10.0.0.0/16
 - d. **Public Subnet CIDR Block:** 10.0.0.0/24
 - e. **Private Subnet CIDR Block:** 10.0.1.0/24
 - f. Leave everything else as default.
6. Create the Second VCN. Click **Start VCN Wizard**.
 - a. **VCN Name:** IAD-AP-LAB06-1-VCN-02
 - b. Make sure your compartment is selected.
 - c. **VCN IPv4 CIDR block:** 172.0.0.0/16
 - d. **Public Subnet CIDR Block:** 172.0.0.0/24
 - e. **Private Subnet CIDR Block:** 172.0.1.0/24
 - f. Leave everything else as default.

Establish Local Peering for VCNs

1. Click the first VCN, and then click **Create Local Peering Gateway**.
2. Name it **IAD-AP-LAB06-1-LPG-01**.
3. Click the second VCN and create a second local peering gateway named **IAD-AP-LAB06-1-LPG-02**.
4. Click the More Actions menu (three vertical dots at the far right of the listed LPG) and select **Establish Peering Connection**.
5. Select the other VCN and LPG in your compartment.

Create a VM Instance

1. In the navigation menu, navigate to Compute, and then **Instances**.
2. Click **Create Instance**. Fill in the following fields:
 - **Name:** IAD-AP-LAB06-1-VM-01
 - **Availability Domain:** AD-1
 - **Image and Shape:** Oracle Linux 8.x, VM.Standard.A1.Flex with 1 OCPU and 6 GB Memory.
 - **Virtual cloud network:** IAD-AP-LAB06-1-VCN-01
 - **Subnet:** IAD-AP-LAB06-1-VCN-01
 - Assign a public IP address.
 - Generate or upload SSH keys.
3. Click **Create Instance**.

Create zone-a.local Custom Private Zone

Private DNS zones contain DNS data only accessible from within a virtual cloud network (VCN), such as private IP addresses. A private DNS zone has similar capabilities to an internet DNS zone, but provides responses only for clients that can reach it through a VCN. In this practice, you'll create a private zone and a record. Later you will use that to play with the VCN resolvers and fetch records from zones associates with other VCNs.

Tasks

1. From the main menu, navigate to **Networking > DNS Management > Zones > Private Zones**. You should see the private zones that are created automatically for your subnets.
2. Click **Create Zone** and create a zone called **zone-a.local**. Select **Selecting existing DNS Private View**, and then select **IAD-AP-LAB06-1-VCN-01**. Click **Create**.
3. After the zone is created, you will see the automatically generated NS and SOA records.
4. Click **Manage Records**.
5. Click **Add Record**.
 - **Name:** server01
 - **Type:** A - IPv4 Address
 - **TTL:** 30 seconds. If the lock icon is engaged, click on it to disengage, and enable the field.
 - **Address:** 10.0.0.2
6. Click **Add Record**.
7. Click **Publish Changes** and Click **Confirm Publish Changes**.

Create zone-b.local Custom Private Zone

In this practice, you'll create a second private zone for your second VCN.

Tasks

1. In the breadcrumbs, click **Zones**, and then **Private Zones**. You should see the private zones that are created automatically for your subnets.
2. Click **Create Zone** and create a zone called **zone-b.local**. Select **Selecting existing DNS Private View**, and then select **IAD-AP-LAB06-1-VCN-02**. Click **Create**.
3. After the zone is created, you will see the automatically generated NS and SOA records.
4. Click **Manage Records**.
5. Click **Add Record**.
 - **Name:** server01
 - **Record Type:** A - IPv4 Address
 - **TTL:** 60 seconds.
 - **Address:** 172.16.0.123
6. Click **Add Record**.
7. Click **Publish Changes** and Click **Confirm Publish Changes**.

Test Instance for Associated Zones

Next, you'll SSH into the instance you created to verify the DNS records and how they are displayed, depending on the zones and private views associated to the VCNs. Note that while both zone-a.local and zone-b.local have been created, only the VCN for zone-a.local has a configured resolver and view.

Tasks

1. Click the Cloud Shell icon (next to where your region is listed at the top right) and SSH into your instance with your private SSH key filename and public IP address (Remember not to include the \$ when you paste the commands).

```
$ ssh -i <private_ssh_key> opc@<ip-address>
```

2. Look up server01. zone-a.local:

```
$ host server01.zone-a.local
```

You should see this response:

```
server01.zone-a.local has address 10.0.0.2
```

3. Look up the system-generated zone entry:

```
$ host -t NS zone-a.local
```

You should see this response:

```
zone-a.local name server vcn-dns.oraclevcn.com.
```

4. Look up the authority record for the zone

```
$ host -t SOA zone-a.local
```

You should see this response:

```
zone-a.local has SOA record vcn-dns.oraclevcn.com.  
hostmaster.oracle.com. 2 3600 3600 3600 10
```

5. Lookup server01.zone-b.local:

```
$ host server01.zone-b.local
```

You should see this response:

```
Host server01.zone-b.local not found: 3 (NXDOMAIN)
```

This means that zone-b is not associated with any of the VCN's views. Exit cloud shell for now.

Configure the VCN Resolver Adding the Other Private View

Now, you'll configure a resolver by adding a private view for the other VCN.

Tasks

1. From the main menu, click **Virtual Cloud Networks** under Networking.
2. Click **IAD-AP-LAB06-1-VCN-01**. You will be directed to the VCN details.
3. Within the VCN information, go to the DNS resolver and click **IAD-AP-LAB06-1-VCN-01**. You will be directed to the Private resolver details.
4. Click **Manage Private Views**. In the Manage Private View page, perform the following actions:
 - a. In **Choose a Private View** of the new added line, select **IAD-AP-LAB06-1-VCN-02** from the drop-down list.
 - b. Click **Save Changes**.

Note: Please wait until the status of Private resolver details page changes from Updating to Active.

Test Instance for Associated Zones

Now, you'll SSH into your instance one more time to verify that the server resolves properly.

Tasks

1. In Cloud Shell, SSH to your instance.

```
$ ssh -i <private_ssh_key> opc@<ip-address>
```

2. Lookup `server01.zone-b.local`

```
$ host server01.zone-b.local
```

You should see this response:

```
server01.zone-b.local has address 172.16.0.123
```

If it does not resolve, be aware that the resolver's change may take a few minutes. If you don't want to wait, reboot the instance.

Mahamat Guiagoussou (mahamat.guiagoussou@oracle.com) has a non-transferable license to use this Guide.

Design Cloud-Native, Microservices, and Serverless Architecture: Build and Deploy an Oracle Function

Lab Practices

Estimated Time: 30 minutes

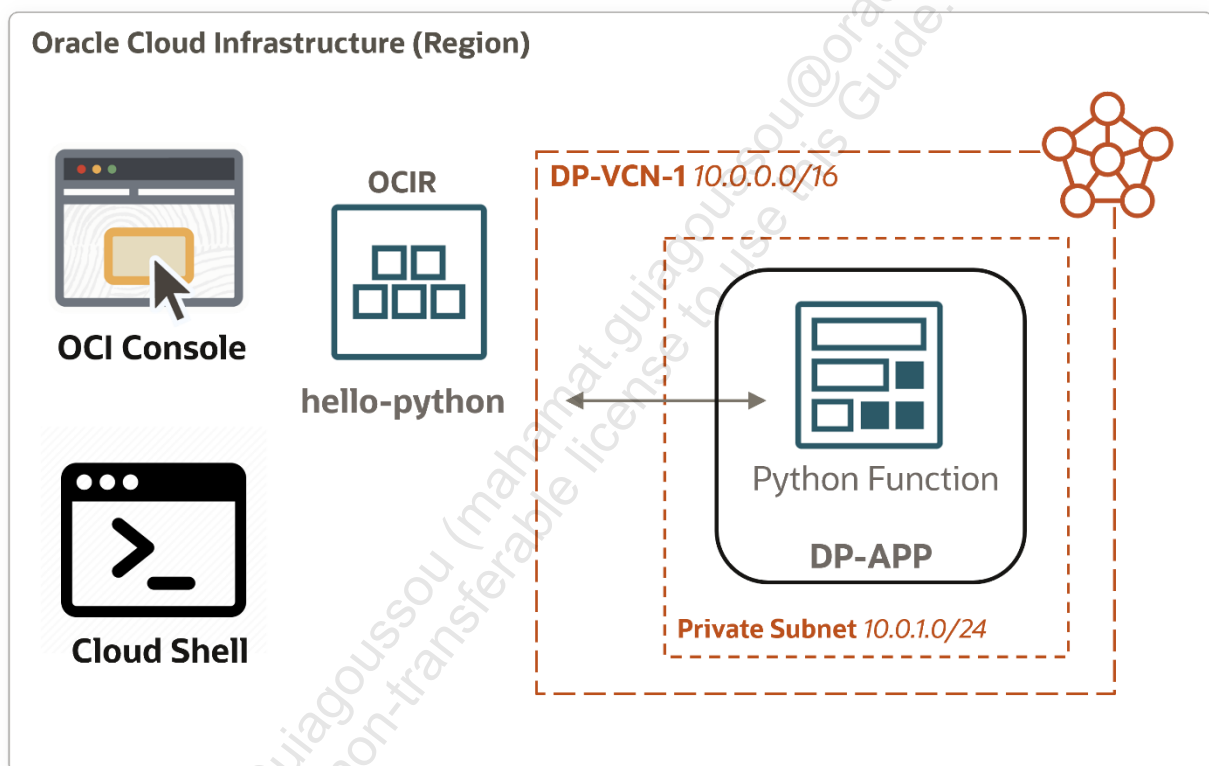
Mahamat Guiagoussou (mahamatguiagoussou@oracle.com) has a
non-transferable license to use this guide.

Get Started

Overview

In this practice, you will build and deploy a sample serverless function written in Python to Oracle Functions. This will be segmented into three sections:

- Create a Virtual Cloud Network (VCN) and Functions application.
- Create a private repository in OCIR and setup Cloud Shell for access.
- Build and deploy the Function container and validate function.



Prerequisites

- An assigned OCI tenancy, compartment, and user credentials.
- The code for the Python function has been staged in GitHub at this location: <https://github.com/ou-developers/oci-functions>

Create a VCN and Functions Application

Serverless functions are deployed to an OCI Functions Application, which must be configured to run functions in one to three subnets belonging to a specific VCN. The tasks for this part will assume that you require a new VCN.

You'll use the VCN with the Internet Connectivity wizard to create the new VCN that includes a regional public subnet, a regional private subnet, an internet gateway, a NAT gateway, and a service gateway. In addition, the wizard will set up basic security list rules for the two subnets.

Tasks

1. Sign in to the Oracle Cloud Infrastructure (OCI) console.
2. Click the navigation menu, click **Networking**, and then click **Virtual Cloud Networks**.
3. Select the compartment that has been assigned to you in the left column under **List Scope**.
4. Click **Start VCN Wizard**.
5. Select **Create VCN with Internet Connectivity**, and then click **Start VCN Wizard**.
6. Enter **AP-LAB07-1-VCN-1** as the VCN name. Leave the default values for the remaining fields. Click **Next** and click **Create** to provision the VCN.
7. In the main menu, click **Developer Services**. Under Functions, click **Applications**.
8. Click **Create application**, and specify:
 - a. **Name:** DP-APP
 - b. **VCN:** AP-LAB07-1-VCN-1
 - c. **Subnets:** Private Subnet-AP-LAB07-1-VCN-1 (Regional)
 - d. **Shape:** Generic_X86
9. Click **Create**.

Create a Private Repository in OCIR and Set Up Cloud Shell for Access

Prior to uploading and deploying a container image containing the function code, you need to specify a private repository in the Container Registry that is within the same OCI region of the Function Application.

Tasks

1. In the Console, open the navigation menu and click **Developer Services**. Under **Containers & Artifacts**, click **Container Registry**.
2. Click **Create repository**.
3. For Repository name, enter `<userID>/hello-python` where `<userID>` is the OCI username assigned to you. For example: **user22/hello-python**
4. Ensure **Access** is set to **Private**. Click **Create repository**.
5. Go to the navigation menu and click **Developer Services**. Go to **Functions** and click **Applications**.
6. Click **DP-APP**, and then click **Getting Started** in the left navigation pane under **Resources**.
7. Scroll down to reveal **Begin your Cloud Shell session**. Click **Launch Cloud Shell**.

Note: The Cloud Shell environment can take up to 60 seconds to start.

8. Click within the **Cloud Shell window** to open the **Actions menu** located in the top-left pane, then select **"Architecture."**
9. Select **X86_64 architecture**, select the appropriate radio button and then click the **Confirm and Restart button**.

Note: After a successful architecture migration, you will see this notification: **Welcome back you have successfully switched your architecture to X86_64**

10. In the Console, scroll down further to quickly familiarize yourself with the series of commands listed in the **Setup fn CLI on Cloud Shell** section. In the following steps, you will execute some but not all those listed commands. Note the **Copy** links found to the right of listed commands. You will click these links and paste the copied commands into Cloud Shell when executing the subsequent steps.

11. Proceed to **(2) Use the context for your region**. Click the **Copy** link and paste to execute the `fn use context <region name>` in Cloud Shell to set your region identifier.

You'll see a message such as: "Fn: Context `<region name>` currently in use" or "Now using context: `<region name>`"

12. Perform **(3) Update the context with the function's compartment ID**. Issue the `fn update context oracle.compartment-id` command to update the fn CLI context for your compartment.
13. Perform **(4) Provide a unique repository name prefix to distinguish your function images from other people's**. Edit, then issue the `fn update context registry` command to update the fn CLI context for the **prefix** of the repository you just created. **Note:** Replace `[repo-name-prefix]` with the `<userID>` prefix you used earlier when creating the repository.

For example: `fn update context registry phx.ocir.io/ocuocictrng21/user22`

14. Perform **(5) Generate an Auth Token**.
 - a. Click **Generate an Auth Token**. This will open the **User Details** page.
 - b. Click **Generate Token** and enter `mytoken` for the description. Click **Generate Token**, then copy the token in a notepad so you can use it later.

WARNING: The token cannot be retrieved in the Console later, so this value must be saved for later use. After you have copy-pasted your token to a saved text document, click **Close**. Close the Auth Token browser tab.

15. Return to the list of **Setup fn CLI on Cloud Shell** commands. Perform **(6) Log into the Registry using the Auth Token as your password**. Log in to the Container Registry using the listed docker login command.

Note: When prompted for the password, paste in the Auth Token you just copied in the previous step. You will not see the password be pasted, but you will get a login success message.

10. Do not execute any other commands listed in **Setup fn CLI on Cloud Shell**.

Build & Deploy the Function Container and Test Function

The function code is available in a GitHub repository. In this section, you will bring the files into your Cloud Shell VM, and then use the `fn` command to build and deploy the function.

Tasks

1. In Cloud Shell, create a new directory called **labs** and then navigate to that new subdirectory.

```
$ mkdir labs
$ cd labs
```
2. Clone the GitHub repository that contains the Python function.

```
$ git clone https://github.com/ou-developers/oci-functions
```
3. Navigate to the directory containing the **hello-python** function code.

```
$ cd oci-functions/hello-python
```
4. Use the `fn deploy` command to build a container image for the function and add it to the repository. *(This may take up to 60 seconds).*

```
$ fn -v deploy --app DP-APP
```
5. Use the `fn invoke` command to execute the function. *(This may take up to 30 seconds).*

```
$ fn invoke DP-APP hello-python
```

If successful, a JSON result will be returned: **{"message": "Hello World"}**

6. To further validate the deployment, in the Console, open the navigation menu and click **Developer Services**. Under Containers & Artifacts, click **Container Registry**.
7. Expand the repository to view the image label and information.
8. Now navigate to **Developer Services**, and then click **Functions**.
9. Click the application link for **DP-APP**.
 - a. Scroll down and notice the Image, Image digest, and Invoke endpoint for the **hello-python** function.
 - b. Click the **hello-python** link. Note the General Information, as well as the function metrics below.

Congratulations, you created a VCN and function application. You also built and deployed a sample serverless function written in Python to Oracle Functions.

Mahamat Guiagoussou (mahamat.guiagoussou@oracle.com) has a non-transferable license to use this Guide.

Mahamat Guiagoussou (mahamat.guiagoussou@oracle.com) has a non-transferable license to use this Guide.

Design Cloud-native, Microservices, and Serverless Architecture: Create an API Gateway Deployment

Lab Practices

Estimated Time: 30 Minutes

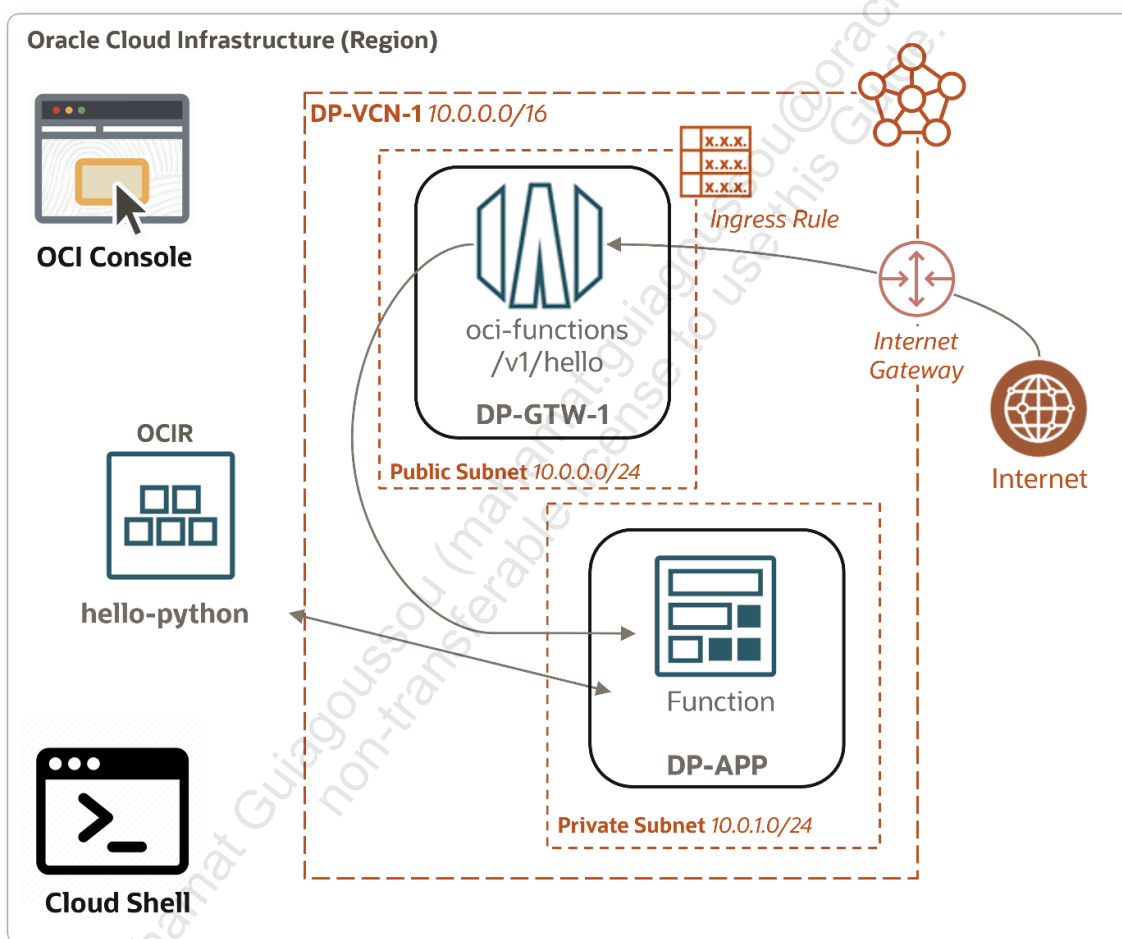
Mahamat Guiagoussou (mahamatguiagoussou@oracle.com) has a
non-transferable license to use this guide.

Get Started

Overview

Oracle Cloud Infrastructure (OCI) API Gateway makes it possible to expose OCI Functions on public endpoints that do not require complex signed HTTP requests. Any function that should be publicly accessible can be given such easy access by creating an API deployment on an API gateway and associating a route in that API deployment with an Oracle Function Backend.

In this lab exercise, you will expose a function previously deployed to a private subnet to be accessed via the API Gateway.



The instructions will be organized into these five practice sections:

- Create a new API gateway.
- Create a new API gateway deployment.

- c. Validate a policy statement that allows API gateway to access the function.
- d. Add an ingress rule for the public subnet.
- e. Call the function via your API gateway deployment.

Prerequisites

- An assigned OCI tenancy, compartment, and user credentials
- You must have completed the preceding lab (Build and Deploy an Oracle Function) to use the same virtual cloud network and application to perform tasks for this practice.

Mahamat Guiagoussou (mahamat.guiagoussou@oracle.com) has a non-transferable license to use this Guide.

Create a New API Gateway

You can use a single API gateway to link multiple backend services (such as load balancers, compute instances, and Oracle Functions) into a single consolidated API endpoint.

You will create an API Gateway that will later be used to create an API gateway deployment to call one or more functions.

Tasks

1. Open the navigation menu and click **Developer Services**. Under **API Management**, click **Gateways**.
1. Select your assigned compartment from the **Compartment** drop-down.
2. Click **Create Gateway**.
3. Fill in the following information to define your API gateway.
 - a. **Name:** AP-LAB08-1-GTW-1
 - b. **Type:** Public
 - c. **Compartment:** *<your-compartment-name>*
 - d. **Virtual Cloud Network:** AP-LAB07-1-VCN-1

This is the VCN that you created in the *Serverless Functions and API Management: Build and Deploy an Oracle Function* practice.
 - e. **Subnet:** Public Subnet-AP-LAB07-1-VCN-1
4. Click **Create**. Wait a few minutes for API Gateway to be created.

Create a New API Gateway Deployment

Having used the API Gateway service to create an API gateway, you can now create an API Deployment that invokes serverless functions defined in Oracle Function.

You will create a new API deployment for your API gateway named `oci-functions` (using `/v1` as the path prefix) in the `AP-LAB08-1-GTW-1` and create a new Route (using `/hello` as the path; selecting **GET** as the method) that invokes the `hello-python` function.

Tasks

1. On the **Gateways** page, click the name of the API gateway you just created, for example, `AP-LAB08-1-GTW`.
2. On the **Gateway Details** page, select **Deployments** from the **Resources** list and then click **Create Deployment**.
3. Click **From Scratch** and fill in the **Basic Information** section:
 - **Name:** `oci-functions`
 - **Path Prefix:** `/v1`
*Note that the deployment path prefix you specify must be preceded by one or multiple forward slash but must not end with it. It can include alphanumeric uppercase and lowercase characters, special characters like `$ - _ . + ! * ' () , % ; : @ & =`, and must not include parameters and wildcards*
 - **Compartment:** `<your-compartment-name>`
4. Click **Next** to display the **Authentication** page and select **No Authentication** to give unauthenticated access to all routes in the API deployment.
5. Click **Next** to enter details of the routes in the API deployment and edit **Route 1** to specify the first route in the API deployment that maps a path and one or more methods to a back-end service:
 - **Path:** `/hello`
 - **Methods:** `GET`
 - Select **Add a single backend**
 - **Type:** `Oracle Functions`
 - **Application:** `DP-APP`
 - **Function Name:** `hello-python`

where,

- **Path** refers to the path for the API calls using the listed methods to the back-end service.
- **Methods** refers to one or more methods accepted by the back-end service.
- **Type** refers to the type of the back-end service.
- **Application** refers to the name of the application in Oracle Functions that contains the function.
- **Function Name** refers to the name of the function in Oracle Functions.

6. Click **Next** to review the details you entered for the new API deployment.

7. Click **Create** to create the new API deployment.

Note that it can take a few minutes to create the new API deployment.

Mahamat Guiagoussou (mahamat.guiagoussou@oracle.com) has a non-transferable license to use this Guide.

Validate a Policy Statement That Allows API Gateway to Access the Function

Verify if an IAM policy statement is present in your compartment that allows API Gateway to access Oracle Functions in your compartment.

Tasks

1. Open the navigation menu and click **Identity & Security**. Under **Identity**, click **Policies**.
2. Ensure you are in your assigned compartment, then click the existing policy link.
3. Verify that the following policy statement is present:

```
allow any-user to use functions-family in compartment
[compartment-name] where ALL {request.principal.type =
'ApiGateway' }
```

Add an Ingress Rule for the Public Subnet

You will create a new stateful CIDR Ingress Rule that allows TCP HTTPS traffic (port 443) from all IP addresses and ports in the default Security List for the virtual network. It will also allow you access from your Cloud Shell.

Tasks

1. Open the navigation menu, click **Networking**, and then click **Virtual Cloud Networks**.
2. Click the VCN that you have created earlier. For example: **AP-LAB07-1-VCN-1**.
3. Under **Resources**, click **Security Lists**.
4. In the **Security List** page, click the **Default Security List for AP-LAB07-1-VCN-1** link.
5. Click **Add Ingress Rule**. Choose whether it's a stateful or stateless rule. By default, rules are stateful unless you specify otherwise. Enter the other basic information:
 - **Source Type:** CIDR
 - **Source CIDR:** 0.0.0.0/0
 - **IP Protocol:** TCP
 - **Destination Port:** 443
6. Click **Add ingress Rule**. The rule will be added and you can see that in the Ingress Rules table.

Call the Function via Your API Gateway Deployment

With your API Gateway and deployment created, you can now call the Function via your API Gateway deployment.

You will use `curl` to call the function via your API Gateway deployment.

Tasks

1. To determine the deployment endpoint, navigate back to **Developer Services**, under **API Management**, and click **Gateways**.
2. Select your API Gateway **AP-LAB08-1-GTW-1**.
3. Under the **Resources** section, click the **Deployments** link. Copy the Endpoint URL for **oci-functions** deployment.
4. Click the **Cloud Shell** icon at the right of the OCI console header to launch it.

In Cloud Shell, execute `curl endpoint-url/hello` to invoke the function. (Be sure to append `/hello` to the end of the URL you copied earlier).

To create the URL for `curl`, add your deployment path to your endpoint.

On successful execution, it will return: **{"message": "Hello World"}**.

Congratulations, you have successfully exposed OCI Functions on public endpoints for easy access.

Mahamat Guiagoussou (mahamat.guiagoussou@oracle.com) has a non-transferable license to use this Guide.

Design Cloud-Native, Microservices, and Serverless Architecture: Manage OCIR and Push and Pull Images Using Docker CLI

Lab Practices

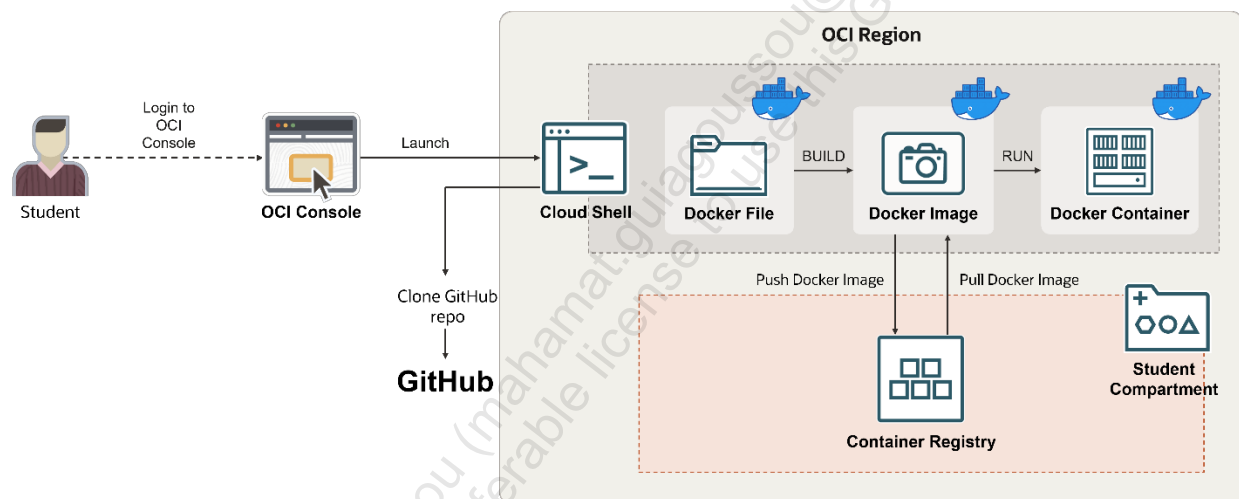
Estimated Time: 60 minutes

Get Started

Overview

There are certain ways for creating, running, and deploying applications in containers using Docker. A Docker image contains application code, libraries, tools, dependencies, and other files needed to make that application run. In addition, the development-to-production workflow can be made simpler with the help of an Oracle-managed registry. For developers, Container Registry makes it simple to store, share, and manage container images (such as Docker images).

In this lab, you will create a Docker image using a Dockerfile, which will further be used to build a container that can run on the Docker platform. You will also create a Container Registry and perform some basic operations such as push and pull a Docker image.



In this lab, you'll:

- Access the Dockerfile
- Build the Docker image
- Run your Docker image as a container
- Access the web application running within the container
- Delete the Docker container
- Create an Auth Token
- Create a new Container Repository

- h. Sign in to Oracle Cloud Infrastructure Registry (OCIR) from the Cloud Shell
- i. Tag the Docker image
- j. Push the tagged Docker image to OCIR Repository
- k. Verify if the image has been pushed
- l. Pull the image from OCIR Repository

For more information on Docker, see the [OCI Docker Documentation](#).

For more information on Oracle Cloud Infrastructure Registry (OCIR), see the [OCI Container Registry Documentation](#).

Assumptions

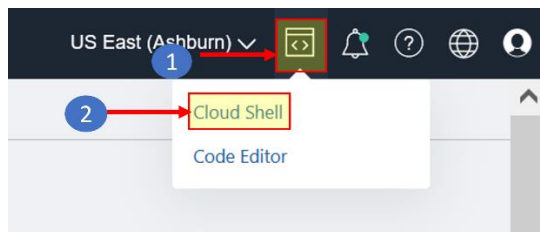
- You are signed into your Oracle Cloud Infrastructure (OCI) account using your credentials.
- You have access to the Git repository link that contains the Dockerfile.
- You will replace the `<userID>` placeholder with your user ID.

Access the Dockerfile

Access the Dockerfile needed to generate the Docker image by cloning a Git repository.

Tasks

1. Open the [Cloud Shell](#) from the Developer tools listed in the OCI console header.



Note: The OCI CLI running in the Cloud Shell will execute commands against the region selected in the Console's region selection menu when the Cloud Shell was started.

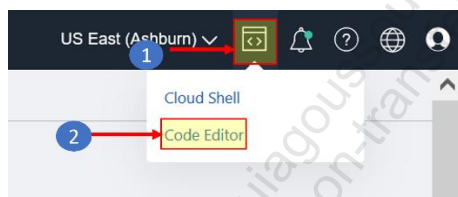
2. Within Cloud Shell, clone the GitHub repository to access the sample Dockerfile which is a simple Nginx HelloWorld application that you will use to build the Docker image.

```
$ git clone https://github.com/ou-developers/docker-helloworld-demo
```

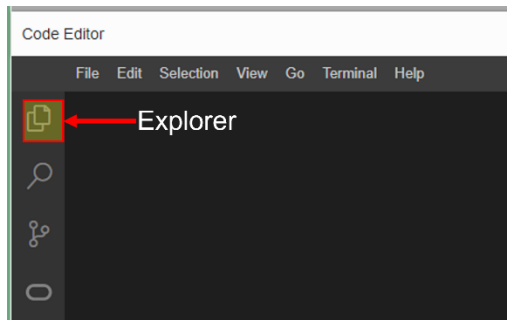
3. Navigate to the cloned directory.

```
$ cd docker-helloworld-demo/
```

4. Open Code Editor from the Developer tools listed in the OCI console header.



5. The tool bar is on the left side of the **Code Editor** window. Click the **Explorer** (top) icon from the left side menu within the Code Editor window.



6. Browse to the cloned Git directory "docker-helloworld-demo" to view the various files you have in the directory including application code and Dockerfile for creating the sample Nginx application.

Build the Docker Image

You're using Cloud Shell as your development environment which comes preinstalled with Docker.

Tasks

1. Check the Docker version using the following command in [Cloud Shell](#). It will return a string with the Docker version installed.

```
$ docker -v
```

For example, Docker version 19.03.11-ol, build 9bb540d

2. Check for existing Docker images in the Cloud Shell.

```
$ docker images
```

3. Create a docker image for the sample Web Application using `docker build` command. This command needs Dockerfile as one of its parameters.

```
$ docker build -t oci_sample_webapp_<userID>:<tag> .
```

For example,

```
$ docker build -t oci_sample_webapp_user22:1.0 .
```

Where,

- `-t` is the switch used to specify the image name.
- Enter an image name using this format: `oci_sample_webapp_<userID>`.
Replace `<userID>` with your user ID.
For example, `oci_sample_webapp_user22`.
- A tag is used to give the image a version. In this lab, you will use `1.0` as tag.
- You are currently in the cloned directory which contains the Dockerfile. Use `."` as the relative path at the end of the command.

4. Upon successful build of a Docker image, verify the image in the local repository using the following command:

```
$ docker images
```

You'll see two entries in the output. One is the base image `"nginx"`, and the other is the custom Docker image for the Web Application `"oci_sample_webapp_<userID>"`.

Run Your Docker Image as a Container

Your Docker image holds the application that you want Docker to run as a container.

Tasks

1. Use the `docker run` command to spin a container based on the image created.

```
$ docker run -d --name webapp-<userID> -p 80:80/tcp  
oci_sample_webapp_<userID>:<tag>
```

Where,

- `-d` flag is used to run container in background and print `CONTAINER_ID`.
- `--name` flag is used to assign a name to the container.
- `-p` flag is used to publish container port 80 to the host machine port 80.
- Replace `<userID>` with your user ID.

For example,

```
$ docker run -d --name webapp-user22 -p 80:80/tcp  
oci_sample_webapp_user22:1.0
```

Note: This command returns the `CONTAINER_ID` of the container started in the background.

2. Check the container that is currently running using the `docker ps` command.

```
$ docker ps
```

You will see a container running with the name `webapp-<userID>` and a corresponding `CONTAINER_ID`.

Access the Web Application Running Within the Container

Verify whether you can access the web application that is running in your container. Once you have verified, stop the running container.

Tasks

1. Use the `curl` command to connect to the local host on port 80 to access the web application.

```
$ curl -k http://127.0.0.1:80
```

The output must display the webpage code. This confirms that your web application is up and running.

2. Get the `CONTAINER_ID` and copy it on a notepad to use it in your next step.

```
$ docker ps -a
```

3. Stop the running container.

```
$ docker stop <CONTAINER_ID>
```

For example,

```
$ docker stop ffab54628f8f
```

4. Use the **curl** command to connect to the localhost on port 80 to access the web application.

```
$ curl -k http://127.0.0.1:80
```

Output: curl: (7) Failed to connect to 127.0.0.1 port 80 after 0 ms:
Couldn't connect to server

This time output will return the above-mentioned error, since the container running the application is no longer active.

Delete the Docker Container

Clean up your resources by removing the container used in this lab.

Tasks

1. Check the status of all the containers in the system.

```
$ docker ps -a
```

The status for the container must show **exited** which means the container is stopped.

2. Delete the existing container using the **rm** flag.

```
$ docker rm webapp-<userID>
```

For example,

```
$ docker rm webapp-user22
```

Output: webapp-user22

Note: On successful deletion it'll return the container name

3. Verify if the container is deleted.

```
$ docker ps -a
```

The container entry should be gone.

Important Note: Do not delete the Docker image created in this lab, as it will be used as an artifact in the upcoming labs.

Congratulations! You have successfully built and containerized a docker image.

Create an Auth Token

Create an auth token to use with Oracle Cloud Infrastructure Registry (OCIR).

Tasks

1. In the top-right corner of the OCI Console, open the **Profile** menu, and then click **User Settings**.

2. On the **Auth Tokens** page, click **Generate Token**.

Note: Each user can only have two auth tokens at a time.

3. Enter **IAD-AP-LAB09-1-AT-01**, as a friendly description for the auth token.
4. Click **Generate Token**. The new auth token is displayed. Here's a sample of how an auth token looks like: `R5kwpS-xxxxxx ([51r]]`. It'll be different in your case.

Note: Copy the auth token to a notepad because you won't see the auth token again in the Console. You'll need this auth token later in this and other labs.

For example,

`R5kwpS-xxxxxx ([51r]]`

5. Click **Close**.

Create a New Container Repository

Create an empty repository in a compartment and give it a name that's unique across all compartments in the tenancy. Having created the new repository, you can push an image to the repository using the Docker CLI.

Tasks

1. Check if you can access Oracle Cloud Infrastructure Registry (OCIR):
 - a. In the Console, open the navigation menu and click **Developer Services**. Under **Containers & Artifacts**, click **Container Registry**.
 - b. Select your *<assigned compartment>* from **List scope** on the left navigation pane.
 - c. Review the repositories that already exist.
2. Click **Create Repository**.
3. Select your *<assigned compartment>* to create a new repository.
4. Enter a name for the new repository: *<region-key>-ap-lab09-1-ocir-1/oci_sample_webapp_<userID>*

Where,

- *<region-key>* is the key for the Oracle Cloud Infrastructure Registry region you're using. For example, *iad* is the region key for US EAST (Ashburn) region. See the [Availability by Region](#) topic in the Oracle Cloud Infrastructure documentation.
- Replace *<userID>* with your user ID.

For example, *iad-ap-lab15-1-ocir-1/oci_sample_webapp_user22*

5. Select the **Private** option to limit access to the new repository.
6. Click **Create Repository**.

Sign In to OCIR from the Cloud Shell

Once you have generated the auth token and created a new repository, sign in to Oracle Cloud Infrastructure Registry (OCIR) from Docker CLI in the cloud shell.

Tasks

1. Click [Cloud Shell](#) at the right of the OCI Console header.

Note: The OCI CLI running in the Cloud Shell will execute commands against the region selected in the Console's region selection menu when the Cloud Shell was started.

2. In the Cloud Shell, log in to OCIR by entering:

```
$ docker login <region-key>.ocir.io
```

For example,

```
$ docker login iad.ocir.io
```

3. When prompted, enter your username in the format given below.

```
<tenancy-namespace>/<username>.
```

Replace the `<tenancy-namespace>` and `<username>` values from the information given in the **Profile** menu.

Where,

`<tenancy-namespace>` is the auto-generated Object Storage namespace string of the tenancy in which to create repositories (as shown on the Tenancy Information page). For username, use the one shown in the profile menu.

For example, `ansh81vrulzp/mahendra@acme.com`. Or `outenancy29/99239886-lab.user16`

Note that for some older tenancies, the namespace string might be the same as the tenancy name in all lowercase letters (for example, `acme-dev`).

If your tenancy is federated with Oracle Identity Cloud Service, use the format `<tenancy-namespace>/oracleidentitycloudservice/<username>`.

Enter the auth token `IAD-AP-LAB09-1-AT-01` (random string) you copied earlier as the password.

For example,

```
R5kwpS-xxxxxx ([51r]]
```

Note: When you enter or paste the password, you'll not see masked characters. Press **Enter** on your keyboard to continue and you should see the "Login Succeeded" message on the screen.

Mahamat Guiagoussou (mahamat.guiagoussou@oracle.com) has a non-transferable license to use this Guide.

Tag the Docker Image

A tag identifies the Oracle Cloud Infrastructure Registry (OCIR) region, tenancy, and repository to which you want to push the image.

This task requires the Docker image `oci_sample_webapp_<userID>:<tag>`, which you created earlier in this lab.

Tasks

1. In the [Cloud Shell](#), run the following command to attach a tag to the image that you're going to push to OCIR repository:

```
$ docker tag oci_sample_webapp_user22:1.0  
<region-key>.ocir.io/<tenancy-namespace>/<repo-name>:<tag>
```

Where,

- `<region-key>` is the key for the Oracle Cloud Infrastructure Registry region you're using. For example, `iad` is the region key for US EAST (Ashburn) region. See the [Availability by Region](#) topic in the Oracle Cloud Infrastructure documentation.
- `ocir.io` is the Oracle Cloud Infrastructure Registry name.
- `<tenancy-namespace>` is the auto-generated Object Storage namespace string of the tenancy (as shown on the Tenancy Information page) to which you want to push the image. For example, `oracletenancy`.
- `<repo-name>` is the name of the target repository to which you want to push the image (for example, `iad-ap-lab09-1-ocir-1/oci_sample_webapp_user22`).
- `<tag>` is an image tag you want to give the image in Oracle Cloud Infrastructure Registry (for example, `latest`).

For example,

```
$ docker tag oci_sample_webapp_user22:1.0  
iad.ocir.io/oracletenancy/iad-ap-lab09-1-ocir-  
1/oci_sample_webapp_user22:latest
```

2. Validate if the new image with the tag is listed.

```
$ docker images
```

Note: Although two tagged images will be shown (`1.0` and `latest`), both are based on the same base image with the same `IMAGE_ID`.

Push the Tagged Docker Image to OCIR Repository

After assigning a tag to the image, you use the Docker CLI to push it to Oracle Cloud Infrastructure Registry repository.

Tasks

1. In the [Cloud Shell](#), run the following command to push the tagged Docker image to OCIR repository:

```
$ docker push <region-key>.ocir.io/<tenancy-namespace>/<repo-name>:<tag>
```

For example,

```
$ docker push iad.ocir.io/oracletenancy/iad-ap-lab09-1-ocir-1/oci_sample_webapp_user22:latest
```

You will see the different layers of the image are pushed in turn and it prints the sha256 digest along with the size of the image on the screen.

Verify if the Image has Been Pushed

Verify if the image has been pushed successfully to the OCIR repository.

Tasks

1. Go back to the OCIR Service page and select your `<assigned compartment>` **from List scope** in the left navigation pane.
2. You'll see the private repository `iad-ap-lab09-1-ocir-1/oci_sample_webapp_<userID>` that you created.
3. Click the name of the repository that contains the image you just pushed from the **dropdown menu** under label **Repositories and images**. You'll see:
 - An image with the tag `latest`
 - A summary page that shows you the details about the repository, including who created it and when, its size, and whether it's a public or a private repository
4. Click the image tag `latest` from the **dropdown menu**.

On the Summary page, you'll see the image size, when it was pushed and by which user, image sha256 digest, and the number of times the image has been pulled.

Pull the Image from OCIR Repository

Perform a pull operation after deleting the existing images from the local docker repository. You will pull the same image that was previously pushed to the OCIR repository.

Tasks

1. Delete the existing images from the local docker repository.

- a. In the [Cloud Shell](#), list all the images.

```
$ docker images
```

- b. Run the `docker rmi` command to delete the tagged image and the original image you created earlier.

```
$ docker rmi oci_sample_webapp_user22:1.0
```

Output: Untagged: oci_sample_webapp_user22:1.0

```
$ docker rmi iad.ocir.io/oracletenancy/iad-ap-lab09-1-ocir-1/oci_sample_webapp_user22:latest
```

This command will first untag the image and delete the image by deleting all the associated layers.

2. Verify if the images are deleted.

```
$ docker images
```

3. Switch to the OCI Console. From the OCIR page, select the repository and the image tag that needs to be pulled.

4. Click **Copy pull command**. The command you copy includes the fully qualified path to the image's location in Container Registry in the following format:

```
<region-key>.ocir.io/<tenancy-namespace>/<repo-name>:<tag>
```

5. Execute the copied command in the Cloud Shell to pull the image to the local repository.

For example,

```
$ docker pull iad.ocir.io/oracletenancy/iad-ap-lab09-1-ocir-1/oci_sample_webapp_user22:latest
```

6. Verify the pulled image from OCIR repository.

```
$ docker images
```

You should see the pulled image listed within the local repository.

Important Note: Do not delete any artifacts and resources created in this lab as they will be required in the upcoming labs.

Congratulations, you have successfully pushed and pulled an image from the OCIR repository.

Mahamat Guiagoussou (mahamat.guiagoussou@oracle.com) has a non-transferable license to use this Guide.

Design Cloud-Native, Microservices, and Serverless Architecture: Deploy a Load-Balanced Web application on an OKE cluster using Kubectl

Lab Practices

Estimated Time: 90 minutes

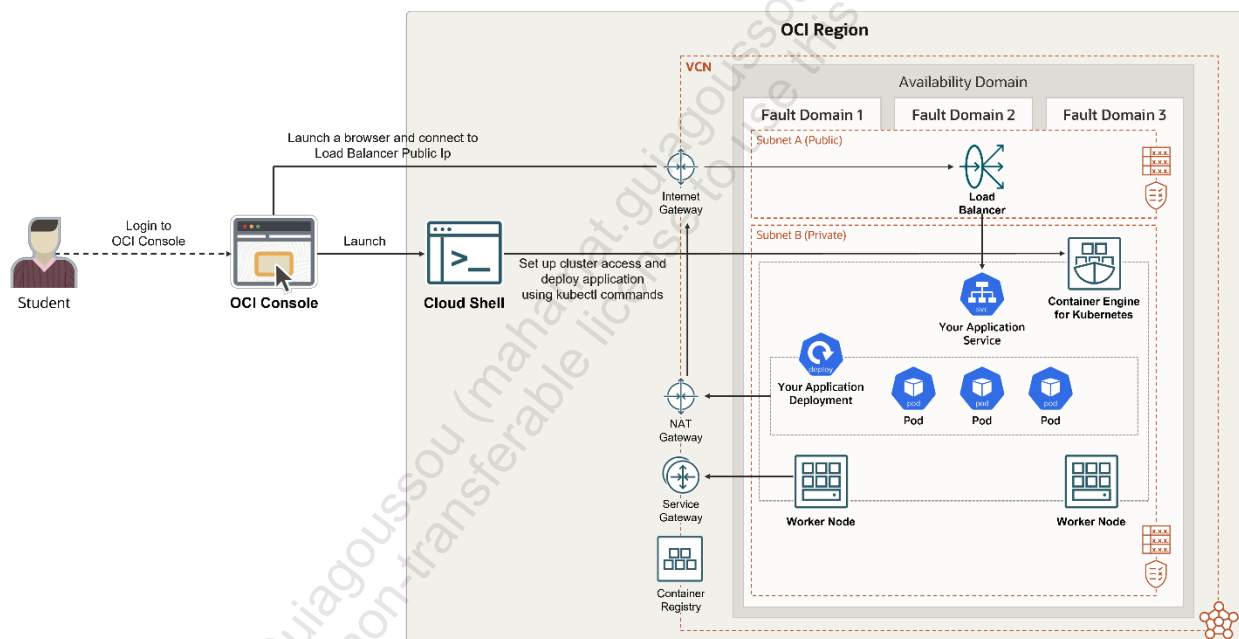
Get Started

Overview

A Kubernetes cluster is a group of nodes (machines running applications). Each node can be a physical machine or a virtual machine.

In this practice, you will set up access to your Kubernetes cluster to deploy your application. The `kubectl` command line client is a versatile way to interact with a Kubernetes cluster, including managing multiple clusters.

Additionally, you will create a named secret which contains your Oracle Cloud Infrastructure (OCI) credentials and add them to a deployment manifest. You will then use this manifest to deploy a sample Web application to an OKE cluster and later verify if the application is accessible.



In this lab, you'll:

- Set up the `kubeconfig` file
- Run `kubectl` commands against Kubernetes cluster
- Create a Kubernetes (OKE) secret
- Add the secret and the image path to the deployment manifest

- e. Deploy the sample Web Application to OKE cluster
- f. Verify if the sample Web Application is accessible
- g. Clean up the resources deployed within OKE cluster

For more information on OCI Container Engine for Kubernetes (OKE), see the [OCI Container Engine Documentation](#).

Prerequisites

You will use the existing Docker image, OCIR repository, Auth token, and Kubernetes namespace from the previous labs to perform tasks for this practice.

- *Design Cloud-Native, Microservices, and Serverless Architecture: Manage OCIR and push and pull images using Docker CLI (Lab09-1)*

Assumptions

- You are signed into your Oracle Cloud Infrastructure (OCI) account using your credentials.
- A pre-created OKE cluster **<EventID>-OCI-ELS-ARCHPRO-OKE** is available in the root compartment. *<EventID>* can be fetched from the **Lab tab** available in the course page.
- You will replace the *<userID>* placeholder with your user ID.

Set Up the kubeconfig File

To access a cluster using `kubectl`, you must set up a Kubernetes configuration file (commonly known as the `kubeconfig` file) for the cluster. The `kubeconfig` file provides the necessary details to access the cluster.

Tasks

1. In the Console, open the navigation menu and click **Developer Services**. Under **Containers and Artifacts**, click **Kubernetes Clusters (OKE)**.
2. Select **root** compartment from **List Scope** on the left navigation pane.

In the table listing Clusters, click the cluster **<EventID>-OCI-ELS-ARCHPRO-OKE** to access using `kubectl`. The Cluster details page shows information on the cluster.

Note: `<EventID>` can be fetched from the Lab tab available in the course page.

3. Click **Access Cluster** to display the **Access Your Cluster** window.
4. Click **Cloud Shell Access** and copy the command to access the `kubeconfig` for your cluster via the VCN-Native public endpoint and paste it on notepad.
5. Launch [Cloud Shell](#) and run the copied command. On successful execution, it will return a new config written to `kubeconfig` file.

For example,

```
$ oci ce cluster create-kubeconfig --cluster-id
ocidl.cluster.oc1.iad.xxxxxaaaziwdigokvlwhuaeslgxi6tdk473xqgodcb
oc6nlgecsyudoxxxxx --file $HOME/.kube/config --region us-
ashburn-1 --token-version 2.0.0 --kube-endpoint PUBLIC_ENDPOINT
```

Note: This is just a representation of the command. Do not use this command to connect with the cluster that's created for this lab.

Run kubectl Commands Against Kubernetes Clusters

Having set up the `kubeconfig` file, you can start using `kubectl` to access the cluster by creating a sample deployment in OKE cluster.

Tasks

1. Verify that `kubectl` can connect to the cluster.

```
$ kubectl get nodes
```

This will return the IP addresses of three worker nodes set up within this OKE cluster.

2. Create a namespace in your Kubernetes cluster to manage your resources.

```
$ kubectl create ns ns-<userID>
```

Where,

`ns-<userID>` - is a unique namespace for your group of resources within a cluster.

Replace `<userID>` with your user ID.

For example:

```
$ kubectl create ns ns-user22
```

3. View the cluster information.

```
$ kubectl cluster-info
```

It dumps relevant information regarding clusters for debugging and diagnosis.

4. Create a sample deployment in OKE cluster.

```
$ kubectl create deployment deploy-<userID> --  
image=iad.ocir.io/ocucictrng5/httpd:latest -n ns-<userID>
```

This command will return `deployment.apps/deploy-<userID> created`.

Where,

- `kubectl create deployment` - is used to create a pod with a single running container.
- `deploy-<userID>` - is a name for your deployment.
- `image= iad.ocir.io/ocucictrng5/httpd:latest` is the fully qualified path to the image in OCIR repository.

- `-n ns-<userID>` - is the namespace where your Kubernetes objects are created.

For example:

```
$ kubectl create deployment deploy-user22 --
image=iad.ocir.io/ocucictrng5/httpd:latest -n ns-user22
```

5. Expose your deployment using service of type load balancer by using the following command:

```
$ kubectl expose deployment deploy-<userID> --type=LoadBalancer
--name=svc-<userID> --port=80 --target-port=80 -n ns-<userID>
```

Where,

- `deploy-<userID>` - is a name for your deployment.
- `--type=LoadBalancer` - exposes the service externally using an OCI load balancer.
- `svc-<userID>` - is the name for your service.
- `--port=80 --target-port=80` - is used to expose the application running within the cluster on port 80.
- `ns-<userID>` - is the namespace where your Kubernetes objects are created.

For example,

```
$ kubectl expose deployment deploy-user22 --type=LoadBalancer
--name=svc-user22 --port=80 --target-port=80 -n ns-user22
```

This command will return `svc-<userID>` exposed.

6. View all the deployments in your namespace.

```
$ kubectl get deploy -n ns-<userID>
```

The output of this command will be a row with the deployment name and ready column set to 1/1. The age column determines the duration of the deployment created.

7. View all the pods in your namespace.

```
$ kubectl get pods -n ns-<userID>
```

The output of this command will be a row with the pod name and ready column set to 1/1. The age column determines the duration of the pod created.

8. View all the services in your namespace.

```
$ kubectl get svc -n ns-<userID>
```

The output of this command is a row with service name and type set to Load Balancer. It shows you the details of CLUSTER-IP and EXTERNAL-IP.

9. Copy the IP address listed under the EXTERNAL-IP column and paste it in a browser to access your httpd application that is deployed within OKE cluster.

The webpage will display:

“It Works!”

10. Check the number of instances of pods running in your deployment.

```
$ kubectl get replicaset -n ns-<userID>
```

The output of this command should display the replicaset name. The desired and current columns specify the number of replicas running. Age column determines the duration of replica created.

11. Scale up the current replicas by three so that Kubernetes can start new pods to scale up your service.

```
$ kubectl scale --replicas=3 deployment/deploy-<userID> -n ns-<userID>
```

On successful execution, this command will return “deployment.apps/deploy-<userID> scaled”.

12. Check if you have three replicas running.

```
$ kubectl get replicaset -n ns-<userID>
```

This shows that the Load Balancer service will now balance the incoming requests among these three pods (replicaset).

13. View all the resources running in your namespace.

```
$ kubectl get all -n ns-<userID>
```

This command shows you all the pods, services, deployments, and replicaset running in your namespace within the OKE cluster.

Notice that the pod count has changed to three after the previous scale-up instruction.

14. View the pod logs. The `kubectl logs` command lets you inspect the logs for a particular pod.

```
$ kubectl logs <podname> -n ns-<userID>
```

Where,

`<podname>` - is the complete pod name to be used from the output of `kubectl get all -n ns-<userID>` command. For example, `pod/deploy-user22-cd95b4455-f8plr`.

15. Delete your deployment.

```
$ kubectl delete deploy deploy-<userID> -n ns-<userID>
```

On successful execution, this command will display “`deployment.apps deploy-<userID> deleted`”.

16. Delete your service object.

```
$ kubectl delete svc svc-<userID> -n ns-<userID>
```

On successful execution, this command will display “`service svc-<userID> deleted`”.

17. Run the following command and you’ll not find any resources in your namespace.

```
$ kubectl get all -n ns-<userID>
```

Output: `No resources found in ns-<userID> namespace.`

18. Since all the resources are deleted, if you go back to your browser and hit refresh on the IP address you pasted earlier. The page will no longer respond.

Important Note: Do not delete the namespace and entry created in the kubeconfig file in this lab, as they will be required in the upcoming practice.

Congratulations! You have successfully deployed a sample web application to the OKE cluster.

Create a Kubernetes (OKE) Secret

To enable Kubernetes to pull an image from OCIR repository when deploying an application, you need to create a Kubernetes secret. The secret contains all the login details you would provide while logging in to OCIR using the `docker login` command, including your auth token.

Tasks

1. Open [Cloud Shell](#) available within the Developer tool in the Console header.

Note: The OCI CLI running in the Cloud Shell will execute commands against the region selected in the Console's region selection menu when the Cloud Shell was started.

2. Run the following command to create a secret:

```
$ kubectl create secret docker-registry <name-of-secret>-<userID> -
--docker-server=<region-key>.ocir.io --docker-username='<tenancy-
name>/<oci-username>' --docker-password='<oci-auth-token>' --
docker-email='<email-address>' -n ns-<userID>
```

Where,

- `<name-of-secret>-<userID>`: A unique name for the secret. For example, `ocir-secret-user22`. Replace `<userID>` with your user ID.
- `<region-key>`: The `<region-key>` is the key for the Oracle Cloud Infrastructure Registry region you're using. For example, `iad` is the region key for US EAST (Ashburn) region. See the [Availability by Region](#) topic in the Oracle Cloud Infrastructure documentation.
- `ocir.io` is the Oracle Cloud Infrastructure Registry name.
- `<tenancy-namespace>` is the auto-generated Object Storage namespace string of the tenancy (as shown on the Tenancy Information page) to which you want to push the image. For example, `oracletenancy`.
- `<oci-auth-token>`: Use the auth token (random string) created in the earlier lab for **IAD-AP-LAB09-1-AT-01**, which was saved in your notepad.
For example, `R5kwpS-xxxxx ([51r]]`.

Note: If you do not have an auth token, create a new one by referring to *Design Cloud-Native, Microservices, and Serverless Architecture: Manage OCIR and push and pull images using Docker CLI* (Lab09-1).

- `<email-address>`: Your email address.

For example,

```
$ kubectl create secret docker-registry ocir-secret-user22 --
docker-server=iad.ocir.io --docker-username='oracletenancy/user22'
--docker-password='R5kwpS-xxxxx([51r]]' --docker-
email='user22@oracle.com' -n ns-user22
```

You will see this confirmation message “secret/ocir-secret-user22 created” for secret creation on the screen.

3. Run the following command to verify if the secret has been created:

```
$ kubectl get secrets -n ns-<userID>
```

For example,

```
$ kubectl get secrets -n ns-user22
```

You will see the secret details displayed with the name, age, and other attributes.

Mahamat Guiagoussou (mahamat.guiagoussou@oracle.com) has a non-transferable license to use this Guide.

Add the Secret and the Image Path to the Deployment Manifest

After the secret is created, you are required to include name of the secret (`<name-of-secret>-<userID>`) and full path of the image (`iad-ap-lab09-1-ocir-1/oci_sample_webapp-<userID>:latest`) pushed to OCIR repository in the deployment manifest which is used for deploying the sample web application to an OKE cluster.

Note: You pushed the image to OCIR repository in *Design Cloud-Native, Microservices, and Serverless Architecture: Manage OCIR and push and pull images using Docker CLI (Lab09-1)*. That's the image you'll be using in this task.

Tasks

1. Open the [Code Editor](#) available within the Developer tool in the Console header. Code Editor allows you to edit files and source codes present in the cloned Git directory within the Cloud Shell.

The Tool Bar is on the left side of the Code Editor window. Click the **Explorer** (top) icon from the left side menu within the code editor window.

- a. Within the Code Editor window, navigate to the cloned Git directory named `docker-helloworld-demo`, which is present in the user's home directory.
- b. Browse to the file `HelloWorld-1b.yaml` in the cloned Git directory and replace the placeholders with relevant values in the **Deployment** section:

- 1) `name: helloworld-deployment-<userID>`
- 2) `namespace: ns-<userID>`
- 3) `image: <region-key>.ocir.io/<tenancy-namespace>/<repo-name>:<tag>`
Where,

- `<region-key>`: The `<region-key>` is the key for the Oracle Cloud Infrastructure Registry region you're using. For example, `iad` is the region key for the US EAST (Ashburn) region. See the [Availability by Region](#) topic in the Oracle Cloud Infrastructure documentation.
- `<tenancy-namespace>`: The auto-generated Object Storage namespace string of the tenancy (as shown on the Tenancy Information page) to which you want to push the image. For example, `oracle-tenancy`.

- `<repo-name>:<tag>`: The repository name 'iad-ap-lab09-1-ocir-1/oci_sample_webapp_<userID>:latest' used to tag and push the image.

4) name: ocir-secret-<userID>

c. Also, replace the placeholders in the **Service** section:

1) name: helloworld-service-<userID>

2) namespace: ns-<userID>

The file will look similar after you've made all the changes:

```

HelloWorld-lb.yaml x
docker-helloworld-demo > HelloWorld-lb.yaml
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: helloworld-deployment-user22
5    namespace: ns-user22
6  spec:
7    selector:
8      matchLabels:
9        app: helloworld
10   replicas: 1
11   template:
12     metadata:
13       labels:
14         app: helloworld
15     spec:
16       containers:
17         - name: helloworld
18           # enter the path to your image, be sure to include the correct region prefix
19           image: iad.ocir.io/oci_sample_webapp_user22:latest
20           ports:
21             - containerPort: 80
22     imagePullSecrets:
23       # enter the name of the secret you created
24       - name: ocir-secret-user22
25   ---

HelloWorld-lb.yaml x
docker-helloworld-demo > HelloWorld-lb.yaml
25   ---
26   apiVersion: v1
27   kind: Service
28   metadata:
29     name: helloworld-service-user22
30     namespace: ns-user22
31   spec:
32     type: LoadBalancer
33     ports:
34       - port: 80
35         protocol: TCP
36         targetPort: 80
37     selector:
38       app: helloworld
39
  
```

3) Click Save from the File menu and exit the Code Editor.

Deploy the Sample Web Application to OKE Cluster

After making changes to manifest, you are ready to deploy the application to the OKE cluster.

Tasks

1. Open [Cloud Shell](#) and change to the docker-helloworld-demo directory.

```
$ cd ~/docker-helloworld-demo
```

2. Run the following command:

```
$ kubectl create -f HelloWorld-lb.yaml
```

A confirmation of deployment and service creation will be displayed.

Note: The HelloWorld Service Load Balancer is implemented as an OCI Load Balancer with a backend set to route incoming traffic to the cluster nodes.

The OKE service creates new Load Balancer in the root compartment. You can see the new Load Balancer in the OCI Console by navigating to the **Load Balancers** page under **Networking** by selecting the root compartment from the **List Scope** menu from the left navigation pane.

Make a note of the overall health and public IP address for the Load Balancer.

Verify if the Sample Web Application Is Accessible

Your deployment should now be running on an OKE cluster node.

Tasks

1. Open Cloud Shell and run the command:

```
$ kubectl get services -n ns-<userID>
```

For example,

```
$ kubectl get services -n ns-user22
```

Note: The status of the `EXTERNAL-IP` column will show `<pending>` initially. Re-run the command at some interval until the IP is allotted.

You'll observe details of the services running on cluster nodes. You'll also observe HelloWorld-Service Load Balancer details such as External/Public IP and Port Number.

2. Launch an Internet Browser and enter the HelloWorld-Service Load Balancer's External/Public IP into the browser's address bar to access the deployed application. The load balancer routes the request to available nodes in the cluster.

In this lab, you'll see one node as the replica count is set to 1 in the Kubernetes manifest. Once the request reaches the node, you'll see the following web page:



3. Now comes the fun part! Let's pretend your sample web application has suddenly gained popularity and you are now required to allocate more resources to it.

The OKE cluster is running on a single node pool with three worker nodes, thus you can easily scale your deployment.

- a. To scale up twice as much and run an additional pod for your current single pod deployment, run the command:

```
$ kubectl -n ns-<userID> scale --replicas=2
deployment/<deploymentname>
```

For example,

```
$ kubectl -n ns-user22 scale --replicas=2 deployment/helloworld-
deployment-user22
```

You will see a confirmation for deployment scaling on screen.

- b. Further, to see pod and deployment details, run the command:

```
$ kubectl get all -n ns-<userID>
```

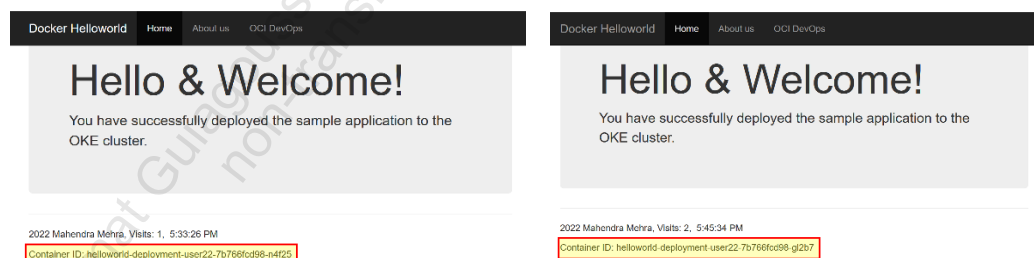
For example,

```
$ kubectl get all -n ns-user22
```

Here, you will observe an additional row for the new pod that has spawned. You can identify the new pod by comparing the Container ID or the value in Age column of the output.

Also, the Deployment row shows '2/2' in the READY column, indicating the deployment is now hosted on two pods.

If you refresh the webpage a few times, you will observe that the two Container IDs alternatively serving your request. This is because the traffic can reach any of these pods via the OCI Load Balancer.



Clean Up the Resources Deployed Within OKE Cluster

Clean up the resources deployed within OKE cluster.

Tasks

1. To delete the sample web application and all other resources you created on the cluster, run the following command:

```
$ kubectl delete -f HelloWorld-lb.yaml -n ns-<userID>
```

For example,

```
$ kubectl delete -f HelloWorld-lb.yaml -n ns-user22
```

2. To confirm the resources are cleared, run the command:

```
$ kubectl get all -n ns-<userID>
```

For example,

```
$ kubectl get all -n ns-user22
```

You will observe that no resources are found in the namespace.

Congratulations! You have successfully deployed a sample web application to the OKE cluster.

Infrastructure As Code: Create a Reusable VCN Configuration with Terraform

Lab Practices

Estimated Time: 30 minutes

Mahamat Guiagoussou (mahamat.guiagoussou@oracle.com) has a
non-transferable license to this guide.

Get Started

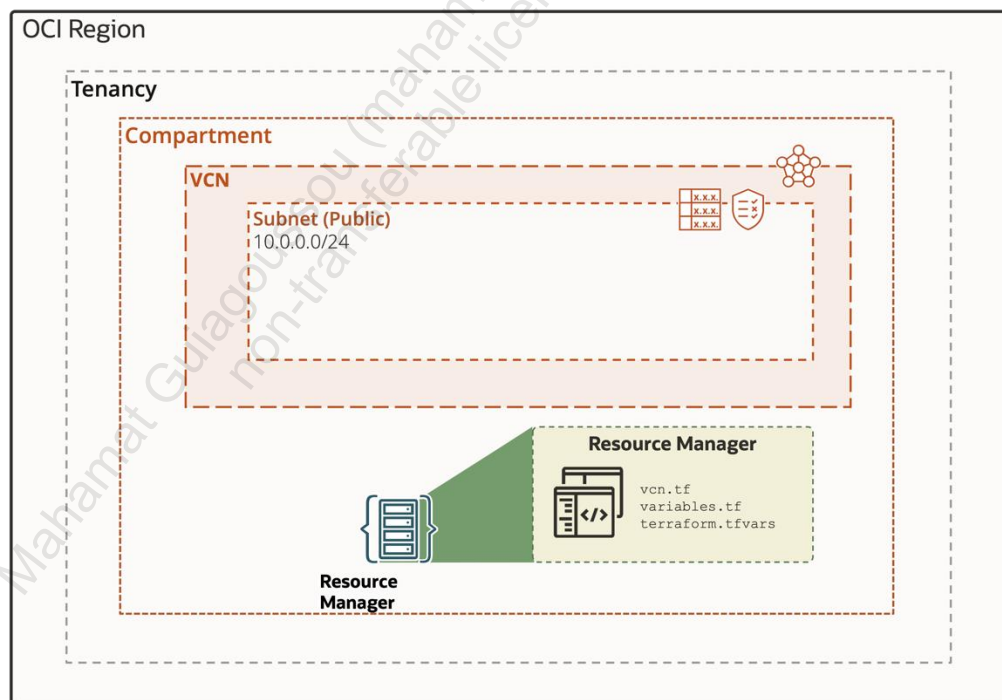
Overview

There are multiple ways to create a VCN and subnet in the Oracle Cloud Console. Particularly if you want to launch several VCNs with the same configuration, it's beneficial to use Terraform or Resource Manager to streamline and automate that process. Terraform can manage low-level components like compute, storage, and networking resources, as well as high-level components like DNS entries and SaaS features.

In this lab, you'll launch and destroy a VCN and subnet by creating Terraform automation scripts and issuing commands in Code Editor. Next, you'll download those Terraform scripts and create a stack by uploading them into Oracle Cloud Infrastructure Resource Manager. You'll then use that service to launch and destroy the same VCN and subnet.

In this lab, you'll:

- Create a Terraform folder and file in Code Editor
- Create and destroy a VCN using Terraform
- Create and destroy a VCN using Resource Manager



Prerequisites

- Required IAM policies have already been set up for you.

Mahamat Guiagoussou (mahamat.guiagoussou@oracle.com) has a non-transferable license to use this Guide.

Create a Terraform Folder and File in Code Editor

In this practice, you'll create a folder and file to hold your Terraform scripts.

Tasks

1. Log in to your tenancy in the Cloud Console and open the [Code Editor](#), whose icon is at the top right, to the right of the CLI Cloud Shell icon.
2. Expand the Explorer panel with the top icon on the left panel. It looks like two overlapping documents.
3. Expand the drop-down for your home directory if it isn't already expanded. It's okay if it is empty.
4. Create a new folder by clicking **File**, then **New Folder**, and name it `terraform-vcn`.
5. Create a file in that folder by clicking **File**, then **New File**, and name it `vcn.tf`. To make Code Editor create the file in the correct folder, click the folder name in your home directory to highlight it.
6. First, you'll set up Terraform and the OCI Provider in this directory. Add these lines to the file:

```
terraform {  
  required_providers {  
    oci = {  
      source  = "oracle/oci"  
      version = ">=4.67.3"  
    }  
  }  
  required_version = ">= 1.0.0"  
}
```

7. Save the changes by clicking **File**, and then **Save**.
8. Now, run this code. Open a terminal panel in Cloud Editor by clicking **Terminal**, then **New Terminal**.
9. Use `pwd` to check that you are in your home directory.
10. Enter `ls` and you should see your `terraform_vcn` directory.

11. Enter `cd terraform_vcn/` to change to that directory with.
12. Use `terraform init` to initialize this directory for Terraform.
13. Use `ls -a` and you should see that Terraform created a hidden directory and file.

Mahamat Guiagoussou (mahamat.guiagoussou@oracle.com) has a non-transferable license to use this Guide.

Create and Destroy a VCN Using Terraform

Terraform uses providers to interface between the Terraform engine and the supported cloud platform. The Oracle Cloud Infrastructure (OCI) Terraform provider is a component that connects Terraform to the OCI services that you want to manage. In this practice, you'll create a Terraform script that will launch a VCN and subnet. You'll then alter your script and create two additional files that will apply a compartment OCID variable to your Terraform script.

Tasks

Write the Terraform

1. Open the [OCI Provider documentation](#) in the Terraform Registry to familiarize yourself with the OCI Terraform provider. As you go along the lab, it may be helpful to try and find the relevant portions of the documentation.
2. Add the following code block to your Terraform script to declare a VCN, replacing `<your_compartment_ocid>` with the proper OCID. The only *strictly* required parameter is the compartment OCID, but you'll add more later.

If you need to retrieve your compartment OCID, navigate to **Identity & Security**, then **Compartments**. Find your compartment, hover over the OCID, and click **Copy**.

```
resource "oci_core_vcn" "example_vcn" {  
  compartment_id = "<your_compartment_ocid>"  
}
```

This snippet declares a *resource block* of type `oci_core_vcn`. The label that Terraform will use for this resource is `example_vcn`.

3. In the terminal, run `terraform plan`, and you should see that Terraform would create a VCN. Since most of the parameters were unspecified, terraform will list their values as "(known after apply)." You can ignore the "-out option to save this plan" warning for this lab.

Note that `terraform plan` parses your Terraform configuration and creates an execution plan for the associated stack, while `terraform apply` applies the execution plan to create (or modify) your resources.

4. Add a display name and CIDR block (the bolded portion) to the code. Note that we want to set the `cidr_blocks` parameter, rather than `cidr_block` (which is deprecated). The region code IAD is used below, for the US East (Ashburn) region.

```
resource "oci_core_vcn" "example_vcn" {
  compartment_id = "<your_compartment_ocid>"
  display_name = "IAD-AP-LAB11-1-VCN-01"
  cidr_blocks = ["10.0.0.0/16"]
}
```

5. Save the changes and run `terraform plan` again. You should see the display name and CIDR block reflected in Terraform's plan.
6. Now add a subnet to this VCN. At the bottom of the file, add the following block:

```
resource "oci_core_subnet" "example_subnet" {
  compartment_id = "<your_compartment_ocid>"
  display_name = "IAD-AP-LAB11-1-SNT-01"
  vcn_id = oci_core_vcn.example_vcn.id
  cidr_block = "10.0.0.0/24"
}
```

Note the line where we set the VCN ID. Here we reference the OCID of the previously declared VCN, using the name we gave it to Terraform: `example_vcn`. This dependency makes Terraform provision the VCN first, wait for OCI to return the OCID, then provision the subnet.

7. Run `terraform plan` to see that it will now create a VCN and subnet.

Add Variables

8. Before moving on there are a few ways to improve the existing code. Notice that the subnet and VCN both need the compartment OCID. We can factor this out into a variable. Create a file named `variables.tf`.
9. In `variables.tf`, declare a variable named `compartment_id`:

```
variable "compartment_id" {
  type = string
}
```

10. In `vcn.tf`, replace all instances of the compartment OCID with `var.compartment_id` as follows:

```
terraform {
  required_providers {
    oci = {
      source  = "oracle/oci"
      version = ">=4.67.3"
    }
  }
  required_version = ">= 1.0.0"
}

resource "oci_core_vcn" "example_vcn" {
  compartment_id = var.compartment_id
  display_name   = "IAD-AP-LAB17-1-VCN-01"
  cidr_blocks    = ["10.0.0.0/16"]
}

resource "oci_core_subnet" "example_subnet" {
  compartment_id = var.compartment_id
  display_name   = "IAD-AP-LAB17-1-SNT-01"
  vcn_id         = oci_core_vcn.example_vcn.id
  cidr_block     = "10.0.0.0/24"
}
```

Save your changes in both `vcn.tf` and `variables.tf`.

11. If you were to run `terraform plan` or `apply` now, Terraform would see a variable and provide you a prompt to input the compartment OCID. Instead, you'll provide the variable value in a dedicated file. Create a file named exactly `terraform.tfvars`.
12. Terraform will automatically load values provided in a file with this name. If you were to use a different name, you would have to provide the file name to the Terraform CLI. Add the value for the compartment ID in this file:

```
compartment_id = "<your_compartment_ocid>"
```

Be sure to save the file.

13. Run `terraform plan` and you should see the same output as before.

Provision the VCN

14. Run `terraform apply` and confirm that you want to make the changes by entering **yes** at the prompt.
15. Navigate to VCNs in the console. Ensure that you have the right compartment selected. You should see your VCN. Click its name to see the details. You should see its subnet listed.

Terminate the VCN

16. Run `terraform destroy`. Enter **yes** to confirm. You should see the VCN terminate. Refresh your browser if needed.

Mahamat Guiagoussou (mahamat.guiagoussou@oracle.com) has published this Guide.
non-transferable license to use this Guide.

Create and Destroy a VCN Using Resource Manager

You can better manage the infrastructure provisioned through Terraform by migrating to Resource Manager instead of running Terraform locally in Cloud Shell or Code Editor. In this section, we will reuse the Terraform code but replace the CLI with Resource Manager.

Tasks

1. Create a folder named `terraform_vcn` on your host machine. Download the `vcn.tf`, `terraform.tfvars`, and `variables.tf` files from Code Editor and move them to the `terraform_vcn` folder to your local machine. To download from Code Editor, right-click the file name in the Explorer panel, and select **Download**. You could download the whole folder at once, but then you would have to delete Terraform's hidden files.

Create a Stack

2. Navigate to Resource Manager in the Console's navigation menu under **Developer Services**. Go to the **Stacks** page.
3. Click **Create stack**.
 - a. The first page of the form will be for stack information.
 - 1) For the origin of the Terraform configuration, keep **My configuration** selected.
 - 2) Under **Stack configuration**, upload your `terraform_vcn` folder.
 - 3) Under **Custom providers**, keep **Use custom Terraform providers** unselected.
 - 4) Name the stack and give it a description.
 - 5) Ensure that your compartment is selected
 - 6) Click **Next**.
 - b. The second page will be for variables.
 - 1) Since you uploaded a `terraform.tfvars` file, Resource Manager will auto-populate the variable for compartment OCID.
 - 2) Click **Next**.

- c. The third page will be for review.
 - 1) Keep **Run apply** unselected.
 - 2) Click **Create**. This will take you to the stack's details page.

Run a Plan Job

4. The stack itself is only a bookkeeping resource—no infrastructure was provisioned yet. You should be on the stack's page. Click **Plan**. A form will pop up.
 - a. Name the job `RM-Plan-01`.
 - b. Click **Plan** again at the bottom to submit a job for Resource Manager to run `terraform plan`. This will take you to the job's details page.
5. Wait for the job to complete, and then view the logs. They should match what you saw when you ran Terraform in Code Editor.

Run an Apply Job

6. Go back to the stack's details page (use the breadcrumbs). Click **Apply**. A form will pop up.
 - a. Name the job `RM-Apply-01`.
 - b. Under **Apply job plan resolution**, select the plan job we just ran (instead of "Automatically approve"). This makes it execute based on the previous plan, instead of running a new one.
 - c. Click **Apply** to submit a job for Resource Manager to run `terraform apply`. This will take you the job's details page.
7. Wait for the job to finish. View the logs and confirm that it was successful.

View the VCN

8. Navigate to VCNs in the Console through the navigation menu under **Networking** and **Virtual Cloud Networks**.
9. You should see the VCN listed in the table. Click on its name to go to its **Details** page.
10. You should see the subnet listed.

Run a Destroy Job

11. Go back to the stack's details page in **Resource Manager**.
12. Click **Destroy**. Click **Destroy** again on the menu that pops up.
13. Wait for the job to finish. View the logs to see that it completed successfully.
14. Navigate back to VCNs in the Console. You should see that it has been terminated.
15. Go back to the stack in Resource Manager. Click the drop-down for **More actions**. Select **Delete stack**. Confirm by selecting **Delete**.

You've now created a Terraform configuration for a VCN; created and destroyed the VCN through Terraform running locally in Cloud Shell/Code Editor; and created and destroyed the VCN through managed Terraform in Resource Manager.

Mahamat Guiagoussou (mahamat.guiagoussou@oracle.com) has a non-transferable license to use this Guide.