Pre-General Availability: 2026-02-26

Oracle® Database

Oracle Blockchain Platform Enterprise Edition for Hyperledger Besu

G50667-02
February 2026

Oracle Database Oracle Blockchain Platform Enterprise Edition for Hyperledger Besu,

G50667-02

Pre-General Availability: *2026-02-26*

Contents

About This Content

Oracle Blockchain Platform Enterprise Edition for Hyperledger Besu lets you build and run smart contracts for private enterprise Ethereum networks. This is a Limited Availability (LA) release, which does not support upgrade or migration.

Audience

This guide is intended for anyone who uses Oracle Blockchain Platform Enterprise Edition for Hyperledger Besu.

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.

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

Oracle Blockchain Platform Enterprise Edition for Hyperledger Besu lets you deploy Besu nodes for maintaining a tamper-resistant distributed ledger for private enterprise Ethereum networks. After deployment you can create multi-partitioned wallets, run predeployed Solidity smart contracts, and deploy your own contracts.

Oracle Blockchain Platform is a preassembled platform designed to manage and facilitate deployment of blockchain networks. Oracle Blockchain Platform Enterprise Edition supports Hyperledger Besu, which is an enterprise Ethereum client based on the Linux Foundation Decentralized Trust open-source project that provides a private Ethereum ledger with multiple Hyperledger Besu nodes running in a high-availability configuration. The platform supports smart contract execution and network management for enterprise use.

Oracle Blockchain Platform Enterprise Edition runs on Kubernetes and is delivered as prebuilt container images for Oracle Cloud Infrastructure (OCI) Kubernetes Engine (OKE). You can install this version of Oracle Blockchain Platform Enterprise Edition by using an OCI Resource Manager stack or by manually installing a distribution package. You can then use the web UI of Blockchain Platform Manager to create a Hyperledger Besu founder instance that starts a private network and connect to the Besu console for operating that instance.

Architecture

As part of the enterprise deployment of Hyperledger Besu, Oracle Blockchain Platform also includes the following components and services:

- Keycloak and OpenLDAP: Provides directory services services and OAuth 2.0/OpenID Connect (OIDC) based strong authentication, user management, fine-grained authorization, and more.
- Istio service mesh: Provides secure, reliable service-to-service communication across Besu nodes and other components, while enabling zero-trust security (mTLS), advanced traffic management, and observability.
- API gateway: Supports the Ethereum JSON-RPC (Web3) API and Besu events and callbacks for integration. In Oracle Blockchain Platform Enterprise Edition permissioned networks this RPC proxy provides an authenticated and authorized access layer for Ethereum JSON-RPC requests. It also provides specialized features for signing transactions, running queries, and managing wallets.
- Prometheus and Grafana: Collects metrics from Besu nodes running in the Kubernetes cluster and provides dashboards for observability. To use these capabilities you need to install kube-prometheus-stack in the cluster using Helm as described in Chapter 7, *Monitor Besu Metrics with Prometheus*.

Key Capabilities

Oracle Blockchain Platform Enterprise Edition support for Hyperledger Besu is a complete platform for permissioned blockchain applications using Solidity smart contracts running on the Ethereum Virtual Machine. It provides node provisioning and lifecycle management, user management and authentication, administration and operations interfaces (Blockchain Platform Manager and Besu Service Console), API capabilities for enterprise integration, custodial wallet and key management, a blockchain explorer for visualizing blocks, transactions, and blockchain network metrics, and a prebuilt smart contract framework to assist developers in

rapidly tailoring and deploying digital asset applications. These features and their capabilities are described in the following details.

- Integrated Blockchain Platform Manager
  - Manages the Besu network life cycle, including provisioning, scaling, configuration, and monitoring.
  - Provides centralized authentication across modules.
- Besu Service Console for network and infrastructure operations
  - Node viewer for Besu peers and health monitoring.
  - Externally owned account (EOA) management to create wallets and run transactions.
  - Predeployed reference smart contracts for common types of digital assets.
  - Log viewing and download for troubleshooting.
  - Built-in block explorer for transaction and block search.
- RPC proxy
  - Supports standard Ethereum/Hyperledger Besu JSON-RPC methods.
  - Includes additional APIs to simplify private chain operations.
- Custodial wallet and key management
  - Creates and manages EOAs using a secure, integrated key vault.
  - Enforces secure key storage and controlled key access.
- Lifecycle transaction service
  - Signs transactions with securely stored EOA keys.
  - Submits transactions to the Besu network and reports status.
- Consortium creation
  - Supports founder and participant node roles.
  - Allows independent authentication and authorization domains per participant.
  - Enables all participants to transact on the same underlying ledger.
- Deployment on OCI
  - Simplifies provisioning into OKE using a provided Resource Manager stack.
- Observability
  - Prometheus metrics scraping.
  - Prebuilt Grafana dashboards for health and performance views.
- Digital Assets smart contract framework
  - Composable software development kit (SDK) with reusable libraries.
  - Enterprise-focused extensions to ERC-20 and ERC-1155 token standards.
  - Prebuilt reference implementations (for example, wrapped CBDC, stablecoin, bond, and a combined token framework).
  - Ability to tailor the provided contracts or build new ones and deploy using Hardhat.

> **Note**
>
> Joining public Ethereum nodes or non-Oracle Blockchain Platform Besu participant nodes is not supported in this Limited Availability release.

Deployment Model

A Besu network consists of multiple Besu nodes connected via peer-to-peer (P2P) networking. Nodes run transactions in the Ethereum Virtual Machine (EVM), persist the blockchain ledger and state, and expose client access through RPC endpoints.

A Besu network includes the following node types.

- Validator nodes: Consensus-participating nodes that propose and validate blocks. Validator nodes must stay current with the canonical chain state to sign and vote correctly.
- Boot nodes: Nodes that support peer discovery so that other nodes can find and join the network.
- RPC nodes: Nodes that serve application traffic via JSON-RPC.
- Archive nodes: Nodes that retain all historical blockchain data and state (no pruning) to support deep queries, audits, and analytics. Archive nodes are not supported in this release of Oracle Blockchain Platform Enterprise Edition for Hyperledger Besu.

Founder/Participant Model

Oracle Blockchain Platform Enterprise Edition for Hyperledger Besu supports two instance types: founder and participant.

A founder instance has the following capabilities.

- Creates and starts the private, permissioned Besu network.
- Operates in a single organization with independent authentication and administrative controls provided by Oracle Blockchain Platform Enterprise Edition.
- Establishes the initial network configuration and provides the baseline network endpoints used to add participants.

Other organizations can create participant instances and join the founder's network. The participant instance contains RPC nodes that connect to and join the founder's Besu network, so that the participant organization can submit transactions and query ledger data using standard Ethereum and RPC APIs. The participant instance operates in its own Oracle Blockchain Platform Enterprise Edition environment, and can be located in the same OCI tenancy or in a different OCI tenancy. In the current release, participant instances use RPC nodes to interact with the founder instance.

The following steps describe the basic operations you complete to work in the founder/participant model.

1. Create the founder network. The founding organization creates a founder Besu instance in Oracle Blockchain Platform Enterprise Edition, which initializes the private permissioned network.
2. Export genesis and node information from the founder. In the instance list in OCI, use the **Actions** menu to export the genesis and node information in `.zip` format.
3. Create a participant instance. The participating organization creates a participant Besu instance in Oracle Blockchain Platform Enterprise Edition by using the exported `.zip` file.

4. Check connectivity and synchronization between the instances. After both instances are running and network connectivity is in place between nodes, the RPC nodes connect and synchronize the ledger. After synchronization, transactions and blockchain data are visible in the participant's service console.
5. Independently administer the instances. Both organizations submit transactions to the same underlying ledger and interact with the same smart contracts, but each retains independent administrative control, which is enforced through the authentication and authorization implemented in their respective blockchain platform manager environments.

Getting Started with Oracle Blockchain Platform Enterprise Edition for Hyperledger Besu

This topic describes the infrastructure requirements, resources, and components needed for an instance.

Supported Compute Shapes

The following compute shapes are supported for Oracle Blockchain Platform Enterprise Edition for Hyperledger Besu:

Compute Shape
VM.Standard.E3.Flex
VM.Standard.E4.Flex
VM.Standard.E5.Flex

For more information on Flex Shapes, see Flexible Shapes.

Resource Estimates

The following table provides details on the minimum service and resource configuration used by Oracle Blockchain Platform Enterprise Edition for Hyperledger Besu. Ensure that these are available in your tenancy or the stack creation will fail.

Service or Resource	Estimated Base Usage
Oracle Kubernetes Engine (OKE)	1 OKE cluster
OKE Node Pool	1 Note: If you're manually creating the stack, the OKE node pool must be set to 39 pods per node. If a new node pool is being created for additional instances, ensure it is set to 14 pods per node minimum.
Compute Instances	1 instance, used by the jump host
Bastion Service	1, used for the jump host connection
Container Registry (OCIR)	10 This is used to store container images for Oracle Blockchain Platform Enterprise Edition components. Ensure 8GB of space is available.
Virtual Cloud Network (VCN)	1 VCN, used by the cluster and jump host
Load Balancer and IP Addresses	2 or more, 1 for Blockchain Platform Manager, 1 for each Besu instance

Cluster Requirement for Hyperledger Besu

Each instance requires a minimum of 1 worker node.

Number of Instances	Worker Node Count	Worker Node Configuration	Worker Node Boot Volume	Load Balancer/IP Address Count
1	1	4 OCPU 64GB memory	150GB	2 total • 1 Blockchain Platform Manager • 1 instance
2	2	4 OCPU 64GB memory	150GB	3 total • 1 Blockchain Platform Manager • 2 instances

Instance Components

A deployed instance of the Besu network provides the following node components.

Component	Default	Minimum	Maximum	Description
Validator nodes	4	4	7	Hyperledger Besu nodes that propose, validate, and add blocks of transactions to the ledger. They use the Quorum Byzantine Fault Tolerance (QBFT) consensus protocol to agree on each block before it is added.
Boot nodes	2	1	2	Boot nodes act as initial connection points, so that new or restarting nodes can discover the network.
RPC nodes	2	1	3	Hyperledger Besu nodes that expose the Ethereum JSON-RPC (JavaScript Object Notation Remote Procedure Call) APIs. They let applications and tools read blockchain data and submit transactions without running their own node.

The instance includes managed components such as an RPC Proxy, a console, wallet storage, and related infrastructure, providing a complete environment to perform lifecycle management (LCM) operations, manage the instance's network, and submit transactions through the RPC Proxy.

Component	Description
RPC proxies	Oracle services that are used to authenticate, manage, and scale access to Web3 JSON-RPC APIs.
Service consoles	Oracle services that let you administer a Besu network through a web console or APIs. A user's access depends on their assigned role and privileges.
Wallet service	An Oracle custodial wallet service that can register and manage user key pairs, sign user transactions, and send transactions to the Besu network for processing.

2

Deploy Oracle Blockchain Platform Enterprise Edition for Hyperledger Besu

The following steps are a high-level overview of how to deploy Oracle Blockchain Platform Enterprise Edition for Hyperledger Besu.

Prerequisites

- Credentials to an OCI tenancy with permissions to create Oracle Kubernetes Engine (OKE) nodes
- OCI Registry credentials (user name and auth token) with permissions to push container images to the repository
- The Oracle Blockchain Platform Enterprise Edition for Hyperledger Besu stack (`.zip` file) or manual distribution package (`.tgz` file) downloaded to your local system

For details when installing using an OCI stack, see Prerequisites.

Deploy the OCI Stack

Use OCI Resource Manager to deploy the stack.

1. Log in to the OCI tenancy and open Resource Manager.
2. Create a stack from the `.zip` file.
3. Enter passwords and OCI Registry credentials.
4. Apply the stack.
5. Review the application information and logs for the stack.
6. Note the admin password you selected, and the ingress gateway IP address for Blockchain Platform Manager.

For complete steps, see Install Using OCI Stacks.
You can also install the Blockchain Platform Manager manually. For more information, see Install Manually.

Create an Instance

Use Blockchain Platform Manager to set up your authentication server, create an administrative user, and create your instance.

1. Log in to the Blockchain Platform Manager using the administrator credentials that you provided during installation.
2. Set the default authentication server.
3. Add an LDAP user with permissions to create instances.
4. Log in as one the new admin user and create an instance.

For complete steps, see Create an Instance.
After you create an instance, you can open the Oracle Blockchain Platform service console to work with Besu nodes and to explore blocks in the ledger. You can also deploy contracts,

create wallets, and send transactions by using the RPC proxy as described in RPC API for Oracle Blockchain Platform Enterprise Edition for Hyperledger Besu.

Install Using OCI Stacks

Complete these steps to install the product by deploying an OCI stack.

Prerequisites

Before you begin the installation process, ensure that you have the following prerequisites:

- Access to an Oracle Cloud Infrastructure (OCI) tenancy and permissions to create and manage Resource Manager stacks, compute and networking resources, and Container Registry and Kubernetes (OKE) resources.
- The Resource Manager stack packages in `.zip` format for Oracle Blockchain Platform Enterprise Edition for Hyperledger Besu.
- A basic understanding of blockchain concepts and Hyperledger Besu. For more information about Hyperledger Besu, see Besu Ethereum client.
- Oracle Cloud Stack prerequisites:
  - A base domain name (for example, `obpee.example.com`) for hosting the Blockchain Platform Manager and service console fully-qualified domain names
  - Passwords for the Blockchain Platform Manager administrator and LDAP administrator
  - An OCI Registry (OCIR) user name and active auth token
  - Cluster details such as worker node count and OCPU and memory specifications

To get your OCIR user name and auth token, complete the following steps.

1. In the OCI console, select **Profile** and then select your user account.
2. Open **Tokens and keys**.
3. Under **Auth tokens**, create a token and then copy it and store it securely.
4. Determine your OCIR user name, which typically uses the following format: `<tenancy-name>/oracleidentitycloudservice/<username>`. For example, `acmeinc/oracleidentitycloudservice/dev.user@example.com`.

For more information, see Logging in to Oracle Cloud Infrastructure Registry.

EVM Compatibility

The platform is based on Hyperledger Besu v25.12, which supports Shanghai and Cancun EVM smart contracts. Use the platform to test and assess product capabilities and develop integrations. Do not use the platform in a production environment. API contracts might change between platform releases.

The following table shows the minimum versions of Solidity and Hardhat that can be used.

EVM Version	Minimum Solidity Version	Minimum Hardhat Version
Shanghai	0.8.20	2.14.0
Cancun	0.8.24	2.21.0

Create the Stack

You use Oracle Cloud Stack to create and deploy Oracle Blockchain Platform Enterprise Edition for Hyperledger Besu

1. Log in to OCI and then navigate to **Developer Services**, and then **Resource Manager**, and then **Stacks**.
2. Click **Create Stack**.
3. On the Stack information page, select **My configuration**, select the Zip file radio button, and then upload the `.zip` file that contains the stack configuration. For the limited availability release, the URL of the `.zip` file is provided by your Oracle point of contact in sales or product management.
4. Optionally customize the **Stack Information** fields. All fields are completed with defaults.
   a. Custom provider

      Custom providers are not supported at this time.

   b. Name

      Name of the stack. It has a default name and provides a date time stamp. You can edit this if desired.

   c. Description

      Description of the stack that you are creating.

   d. Create in compartment

      Defaults to the root compartment of your OCI tenancy.

   e. Terraform version

      Ensure version 1.1.x or later is selected.

   f. Tags

      Optional. Tags are a convenient way to assign a tracking mechanism.

   Click **Next**.
5. Provide a name for the stack and enter the required parameters.

   Blockchain Platform Configurations

   a. Domain Name

      Specify the domain URL for hosting Blockchain Platform Manager and the service console. The default is `Besuee.com`.

   b. Administrative User Password

      This is used to set the Blockchain Platform Manager administrative user's (`obpadmin`) password.

   c. LDAP User Password

      This is used to set the administrative user's password for the LDAP authentication server.

   OCIR Image Configurations

   a. OCIR Username

User name used to log into Oracle Cloud Infrastructure Registry. Enter the name of the user in the format:

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

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 example

```
ansh81vru1zp/jdoe@example.com
```

Note that for some older tenancies, the namespace string might be the same as the tenancy name in all lower-case letters. For example,

```
example-dev
```

If your tenancy is federated with Oracle Identity Cloud Service, use the format

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

See Logging in to Oracle Cloud Infrastructure Registry.

b. OCIR Auth Token

The auth token used to access OCIR.

Kubernetes Cluster Configurations

a. Cluster Name

Name of the OCI Kubernetes Engine cluster that will be created. Subsequent resources will also contain this name. In the final state, the `deployment_id` generated by the apply job will be added to this.

b. Node Pool Name

Name of the node pool for the worker nodes.

c. Enable Cluster Autoscaler for Node Pool

This will enable node pools to autoscale based on resource usage and will add and remove worker nodes as needed. When set to `False`, the Maximum Number of Worker Nodes tab is hidden.

d. Minimum Number of Worker Nodes

The minimum number of nodes in the node pool. If autoscaling has not been enabled, this is the total number of worker nodes available. The default is 1. At a minimum, 1 worker node is required.

e. Maximum Number of Worker Nodes

The maximum number of nodes in the node pool. This is only available if autoscaling has been enabled. The default is 10. The maximum is 100.

f. Worker Nodes Instance Shape

Select the appropriate compute shape. The default is `VM.Standard.E5.Flex`. For information on supported shapes and their configurations, see: Supported Compute Shapes.

g. Worker Node OCPU Count

   The number of OCPUs for each worker node compute instance. The minimum is 4 OCPUs. The maximum is 64 OCPUs.

h. Worker Node Memory (GB)

   The amount of memory for each worker node compute instance. The minimum is 64 GB. The maximum is 1024 GB.

6. Review the information, and then create the stack by selecting **Run Apply** and then **Create**.

   The stack creation process begins, which includes the deployment of resources such as a compute VM (jump host) and Kubernetes clusters.

7. Wait for stack creation to complete, which typically can take 40-45 minutes. Monitor the stack creation job and logs to track the deployment progress.

8. Once the stack job is marked `Succeeded`, the Application Information tab will be available. The information on this tab can be used to configure access to Blockchain Platform Manager.

   For more information about these fields, see Application Information.

Install Manually

You can also install Blockchain Platform Manager manually on Oracle Kubernetes Engine.

Prerequisites for Manual Installation

Before you begin the manual installation process, ensure that you have the following prerequisites.

Kubernetes Platform

Access to a Kubernetes cluster via the `kubeconfig` file. For more information on connecting to Kubernetes, see Connect to Oracle Kubernetes Engine.

Software Prerequisites

- kubectl version 1.31.0 or later - command line tool for Kubernetes
- oci-cli version 3.71.4 or later - command line interface for OCI
- istioctl version 1.28.1 or later - command line tool for Istio, a security and traffic management tool for deployments
- Helm version 3.19.4 or later - a Kubernetes package manager
  - Helm version 4 or later is not supported.
- Podman version 4.9.4-rhel or later
- openssl version 1.1.1 or later
- yq version 4.49.2 - a command line YAML processor
- jq v1.6 or later - a command line JSON processor
- bash version 3.2.57 or later - command shell

Install Oracle Blockchain Platform Enterprise Edition for Hyperledger Besu Manually

Complete the following steps to install the platform manually.

1. Enter the following command to install Istio on the cluster.

```
istioctl install --set profile=default --set
values.pilot.env.ENABLE_TLS_ON_SIDECAR_INGRESS=true --set
components.cni.enabled=true --set values.cni.repair.deletePods="true"
    ## Enter "y" when prompted for "Proceed? (y/N)"
```

2. Enter the following command to extract either the full or the lite distribution package. To push images to the container registry, you must use the full distribution. For the limited availability release, the URL of the `.tgz` file is provided by your Oracle point of contact in sales or product management.

```
tar -xzf <distribution-package-file>.tgz
```

3. Update the `runme-input.yaml` file with the required values. You can use the following example `runme-input.yaml` file as a reference.

```
```
imageRegistryConfiguration:
    registry: ams.ocir.io
    imageTagPrefix: ams.ocir.io/oabcs1/besu
    username: dev.sony@oracle.com

  # Used for unattended mode (Wrap around quotes to prevent certain
symbols from being read as part of yaml)
  ocirpwd: ”"
imageReleaseVersion: 26.1.1-2026XXXXXXXXX

storageClassName
controlPlaneStorage:
  storageClassName: oci-bv
  # Example 500Mi, 5Gi
  size: 1Gi

parentDomainName: example.com

#imagePullTimeout: Use this field to customize the wait time (in seconds)
for pulling the required docker images from the repository. Default is
1800 seconds.
imagePullTimeout: 1800

Used for unattended mode
cpAdminPassword: Welcome1
ldapAdminPassword: Welcome1
idbDbPassword: admin
dbRootPassword: admin
operatordbUser: obp-operator-admin
```
```

```
operatordbPassword: opadmin
```

In the previous example, the variables are defined as shown in the following list:

- **`imageRegistryConfiguration.registry`**: The container registry server to use.
- **`imageRegistryConfiguration.imageTagPrefix`**: Container base repository path in the registry to use for resolving images.
- **`imageRegistryConfiguration.username`**: Container registry login user name.
- **`imageRegistryConfiguration.ocirpwd`**: (Optional, but required in unattended mode) Container registry login password.
- **`imageReleaseVersion`** - Oracle Blockchain Platform Enterprise Edition for Hyperledger Besu release version. Use the release version from the distribution package file name.
- **`controlPlaneStorage.storageClassName`**: Kubernetes storage class to use for PVC (PersistentVolumeClaim). If empty, the default `storageClass` is used.
- **`parentDomainName`**: Domain name to use for the Blockchain Platform Manager deployment.
- **`imagePullTimeout`**: Image pull wait timeout in seconds during installation.
- **`cpAdminPassword`**: (Optional, but required in unattended mode) The password for the default Blockchain Platform Manager administrative user (user name: `obpadmin`).
- **`ldapAdminPassword`**: (Optional, but required in unattended mode) The password for the default LDAP administrative user.
- **`idbDbPassword`**: The default IDB persistence root password.
- **`dbRootPassword`**: The default IDB persistence administrative password.
- **`operatordbUser`**: The user name of the operator user. Operators are read-only users, who do not have access to the Accounts page in the service console
- **`operatordbPassword`**: The password of the operator user.

4. Navigate to the directory where you extracted the package and then run the `runme_oke.sh` script.
   - Enter the following command to run in unattended mode and push the images to the container registry. This command works only with the full distribution package.

     ```
     ./runme_oke -u –p
     ```

   - Enter the following command to run in unattended mode.

     ```
     ./runme_oke.sh -u
     ```

   - Enter the following command to run in interactive mode, following the prompts.

     ```
     ./runme_oke.sh
     ```

   After the script runs, you can log in to Blockchain Platform Manager and create an instance.

Create an Instance

After you install Blockchain Platform Manager, you can create an instance of Oracle Blockchain Platform Enterprise Edition for Hyperledger Besu.

Deploying the stack (or manually installing the distribution package) creates Blockchain Platform Manager, which you can use to provision, configure, and manage instances of Oracle Blockchain Platform Enterprise Edition for Hyperledger Besu. Instance nodes run on a set of available Kubernetes worker nodes.

You use Blockchain Platform Manager to complete the following tasks:

- Create, start, stop, and delete instances of Oracle Blockchain Platform Enterprise Edition for Hyperledger Besu.
- View details about all instances and activity records for all instances.
- Configure, activate, and use the LDAP service. You can use the built-in OpenLDAP service or your own LDAP server.
- Create and save user accounts.

Before You Begin

Review the **Application Information** page, which shows details about the Blockchain Platform Manager, Kubernetes cluster, and Terraform output.

- Note the ingress gateway IP address, which is used for Blockchain Platform Manager, and the ingress IP address for instance services, which is used by the service console and RPC gateway.
- Full job logs, including jump host details and Terraform logs, are available under **Jobs**.

To ensure proper host name resolution, add entries to your local `etc/hosts` or `C:\Windows\System32\drivers\etc\hosts` file for Blockchain Platform Manager and OpenLDAP fully-qualified domain names. For a production environment, use DNS and certificates instead of `etc/hosts` file mappings. Add entries similar to the following example, where the the ingress IP address is the ingress gateway IP address shown in the installation output.

```
<ingress-ip> controlplane.<domainname> openldap.<domainname> auth.<domainname>
```

To create an instance:

1. Open Blockchain Platform Manager (`https://controlplane.`*`domainname`*`/console/index.html`)

   Your browser might warn you about a self-signed certificate.
2. Log in to Blockchain Platform Manager by using the `obpadmin` user name and the password that you provided during installation.
3. Activate and test the default OpenLDAP authentication server.
   a. Select **Configuration**, and then **Authentication Servers**.
   b. Select the `Default` authentication server, then select **Save and Set active**.
4. Add an instance administrative user to the system. Do not use the Oracle Blockchain Platform administrator account for routine tasks. Log out and then log back in as the new user.

a. On the **Authentication Servers** tab of the **Configuration** page of Blockchain Platform Manager, click **Add User**.

b. Once you've entered the user name and password, this user will be added to the LDAP server as an administrative user.

c. You can now log out of Blockchain Platform Manager with your default admin user, and log in with this newly added instance administrative user to create an instance.

> **Note**
>
> Once you've successfully logged into Blockchain Platform Manager with this user and provisioned an instance, you may want to disable the default admin user (`obpadmin`) for security reasons. This can be done from the Configuration page Platform Settings tab.

5. On the **Instances** page, click **Create Instance** to create an instance of the Besu network.

   The Besu Create Instance dialog is displayed.

6. Complete the following fields:

Field	Description
Instance Name	Enter a name for your Oracle Blockchain Platform instance. The service instance name: • Must contain one or more characters. • Must not exceed 15 characters. • Must start with an ASCII letter: a to z. • Must contain only lower case ASCII letters or numbers. • Must not contain a hyphen. • Must not contain any other special characters. • Must be unique within the identity domain.
Description	Optional. Enter a short description of the Oracle Blockchain Platform instance.
Domain Name	Enter the domain name for the cluster that you specified when installing. This must contain only lower case ASCII letters. The hostnames generated for the Blockchain Instance services make use of the domain name and the instance name as parent domain and sub domain respectively.
Role	Select **Founder** to create a complete blockchain environment. This instance becomes the founder organization and you can onboard new participants in the network later. Select **Participant** to create an instance that will join an existing blockchain network created elsewhere before this instance can be used.
Configuration	Select a provisioning shape which meets the needs of your deployment. Only Enterprise is currently supported for Oracle Blockchain Platform Enterprise Edition for Hyperledger Besu.
Chain ID	(Founder instance only) Enter your Ethereum chain ID. The default is `1500`. Provide a unique numeric identifier for this network avoiding common public IDs like 1, 137, 1337, or 56, to ensure transaction signatures are valid only on this specific Besu deployment and protected from replay attacks.
Block Period	(Founder instance only) Minimum time (in seconds) between new block creation.
Validator Node Count	(Founder instance only) Number of nodes responsible for validating and proposing blocks.
BootNode Count	(Founder instance only) Number of nodes used for initial peer discovery.
RPC Node Count	(Founder instance only) Number of nodes providing APIs for external communication.

Field	Description
Rich History Oracle DB Profile	Currently not supported.
Upload Genesis File	(Participant instance only) Upload the genesis metadata `.zip` file from the founder instance. Select **Export Genesis Metadata** from the founder instance **Actions** menu to download the genesis metadata file from the founder.

The instance creation process runs, creating the pods for the nodes and services that make up the Besu network.

7. Once the instance is listed in the `Up` state, add the `Hosts` entries to your `Hosts` file.

   ```
   <ingress-ip> console.<instance-name>.<instance-domain-name>
   rpcproxy.<instance-name>.<instance-domain-name>
   ```

Verify Your Instance

Navigate to your service in Blockchain Platform Manager and sign in to verify that your instance is up and running.

1. Use the instance IP address and service console path to access the blockchain platform service console. Your browser might warn you about a self-signed certificate.

   ```
   https://console.<instance-name>.<domain-name>
   ```

2. Log in to the service console with the instance administrator credentials.

Now that your instance is created, you can explore the pages in the service console:

- Dashboard: View nodes, transactions, blocks, and network statistics.
- Nodes: List and manage Besu network nodes.
- Explorer: View and search transactions and blocks, including block number and transaction hash searches.
- Accounts: Create externally-owned accounts (EOAs) and associate wallets with the accounts.
- Logs: View logs by pod or container and download as needed for troubleshooting.

For more information on the console and using it to manage your network, see Manage the Organization and Network.

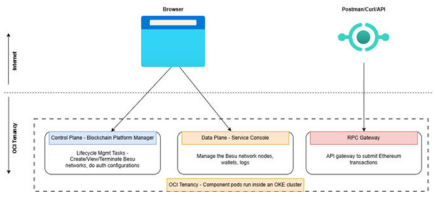
Service Endpoints

These are the service endpoints generated once your instance is created.

The following table shows the endpoints for Blockchain Platform Manager, the service console, and the RPC proxy.

Service	Endpoint Type	URL Format	URL Source
Blockchain Platform Manager	Web URL	`https://controlplane.<domain-name>/console/index.html` Example: `https://controlplane.obpee.example.com/console/index.html`	The domain name that you specified when installing Blockchain Platform Manager.
Service console	Web URL	`https://console.<instance-name>.<instance-domain-name>` Example: `https://console.obpee.example.com`	The instance name and domain name that you specified when creating the instance.
RPC proxy	API endpoint	`https://rpcproxy.<instance-name>.<instance-domain-name>:443` Example: `https://rpcproxy.obpee.example.com:443` Example: `https://rpcproxy.obpee.example.com:443/v1/besu/query`	The instance name and domain name that you specified when creating the instance. A detailed list of APIs is available at the RPC proxy URL in the `/openapi` path.
Authentication service	Fetch access token	`https://auth.<domain-name>:443//auth/realms/master/protocol/openid-connect/token` Example: `https://auth.obpee.example.com:443//auth/realms/master/protocol/openid-connect/token`	The domain name that you specified when installing Blockchain Platform Manager.

Figure 2-1 Service View



3

Blockchain Platform Manager

After you install Oracle Blockchain Platform Enterprise Edition for Hyperledger Besu, you can access Blockchain Platform Manager to configure and provision platform instances.

You use Blockchain Platform Manager to create and manage the life cycle of blockchain platform instances. You an create, start, stop, and delete multiple blockchain platform instances from Blockchain Platform Manager. You can also configure and activate LDAP for user federation, optionally using the built-in OpenLDAP service. Additionally you can create and save user accounts, with information persisted in the LDAP server. The Blockchain Platform Manager interface is divided among three pages: Instances, Activities, and Configuration.

Instances

Use the Instances page to create and manage blockchain platform instances (nodes and networks). Select **Create Instance** to create an instance. For more information, see Create an Instance.

This page lists all instances and the following instance details.

- Role: Founder instances were created as a separate network. Participant instances were created to add nodes to an existing network.
- Shape: The resource configuration of the instance. The Enterprise shape is the only supported shape.
- State: `Creating`, `Up`, `Stopped`, and so on.
- Domain Name: The domain name that you specified when creating the instance.
- Created: Instance creation date/time.
- Version: Version number of the instance.
- Remarks: Additional remarks to indicate instance state.

The **Actions** menu for each instance lets you start, stop, and terminate instances. You can also select **Confidential Client Credentials** to generate or refresh access tokens. Select **Instance URLs** to see the connection URLs for services provided by the Besu instance.

Activities

All lifecycle management actions completed on an instance create a traceable activity record. The activity record logs all lifecycle phase changes the instance goes through from creation to termination. You can filter the activity log by instance, time range, or operation.

This page lists all activities and the following details.

- Instance Name
- Operation: `Start`, `Stop`, `Create`, `Terminate`, and so on.
- Operation Status: The status of the last action on any instance.
- Start Time
- Last Updated Time

- Initiated By: The user who initiated the activity.

Configuration

Use the Configuration page to configure authentication servers and Blockchain Platform Manager itself.

Select **Platform Manager Settings** to view or edit the platform manager name and timeout values.

Select **Authentication Servers** to configure authentication and users. For more information, see User Management.

4

User Management

Blockchain Platform Manager uses an integrated OpenLDAP server for initial Identity and Access Management (IAM) during installation. This OpenLDAP server manages user credentials and enforces Role-Based Access Control (RBAC) using preconfigured groups. All user information such as credentials and group memberships is stored on this OpenLDAP server.

In addition to LDAP-based authentication, Blockchain Platform Manager supports integration with external Identity Management systems (IdMs) via OpenID Connect (OIDC). This enables the use of industry-standard authentication protocols for enhanced interoperability.

LDAP Configuration Management

Blockchain Platform Manager provides a dedicated configuration page for managing LDAP servers. The following actions are supported:

- **Add New:** Configure an external LDAP server for Blockchain Platform Manager, as an alternative to the built-in OpenLDAP server.
- **Save:** Save a new or updated LDAP server configuration.
- **Set Active:** Designate an existing LDAP configuration as active.
- **Save and Set Active:** Save changes and immediately set the updated configuration as active.
- **Test Configuration:** Verify connectivity and accessibility to the specified LDAP server from Blockchain Platform Manager.

Group Creation and Management

For each Blockchain Platform Manager instance that is created, the following groups are provisioned in the OpenLDAP server:

User Role	LDAP Group Name	Description
Platform Management	`OBP_Blockchain Platform Manager<id>_CP_ADMIN`	Users in this group can provision an instance, configure existing instances, set the LDAP configuration, and complete life cycle operations on instances. A user must be a member of this group to be able to log in to Blockchain Platform Manager or create an instance.
Instance Administrator	`BESU_ADMIN_<instance_uuid>`	Users in this group can manage instances by using the console UI.
Instance Operator	`BESU_OPERATOR_<instance_uuid>`	Operators are read-only users. Operators do not have access to the Accounts page in the service console.

User Role	LDAP Group Name	Description
RPC Proxy Client	BESU_RPC_GW_<instance_uuid>	RPC proxy users are typically client applications.

All users provisioned through Blockchain Platform Manager are automatically added to all four groups.

Token Issuance and Group Membership Propagation

When a new instance is created, Blockchain Platform Manager configures the authentication server to enable token issuance with required claims, including user identity and relevant client/party information. Each token includes group membership information, encapsulated in a payload claim. Instance components use these claims to authorize or block external access to workload pods.

You can add users directly to the OpenLDAP server by using OpenLDAP browsers such as jXplorer. Blockchain Platform Manager administrators can use the administrator user name and password that was provided during installation to connect to `openldap.<cp-name>.<cp-domain>:443` with SSL enabled. Once connected, administrators can then add users and assign or modify groups to give the appropriate access levels.

5

Manage the Organization and Network

The Oracle Blockchain Platform Service Console is a comprehensive web interface that enables users to monitor blockchain networks and perform routine administrative tasks. The console also provides specialized tools and functionality for smart contract developers and wallet administrators.

Oracle Blockchain Platform Service Console Overview

After an Oracle Blockchain Platform instance is provisioned, all core functions required to interact with and manage the blockchain network are accessible through the service console. Key capabilities include:

- Managing blockchain nodes
- Configuring network settings and policies
- Deploying smart contracts
- Creating and accessing account wallets
- Using the ledger browser to inspect block details and transactions
- Accessing log files (administrators and operators only)

The console's graphical user interface organizes these features across several tabs:

- **Dashboard:** Presents a summary overview of network health, node status, latest block height, and transaction volume.
- **Nodes:** Displays detailed information about all active nodes, with menus for node management actions.
- **Explorer:** Displays a block explorer to list recent blocks, search by block number and transaction hash, and view details of the transactions.
- **Developer Tools:** Displays predeployed smart contracts, including the source code, Application Binary Interface (ABI), and deployment address.
- **Accounts:** Enables creation and management of Externally Owned Account (EOA) wallets.
- **Logs:** Provides access to log files generated by each pod running Besu nodes.

Note: In the Alpha release, some tabs may not yet be connected to backend services. For operations such as deploying smart contracts, users can utilize the provided RPC Proxy APIs, which act as a Web3 API gateway.

Dashboard

The dashboard shows the following details about your instance.

Summary Header

- Number of nodes
- Total Transactions
- Total number of blocks

- Total number of nodes

Besu Network Information

- Version
- Consensus algorithm
- Chain ID
- Block time

Kubernetes Cluster Details

- Worker node details
- CPU resources
- Memory resources
- Storage resources using OKE persistent volume claims (PVC) disks on Block Storage

Nodes

The Nodes views show a summary by node type and details, which include Name, Organization, Type, Node ID, and Status. There is an Actions menu available for each node action to display the details and, if needed, restart the node.

Explorer

The Explorer tab provides a block explorer to list recent blocks, search by block number and transaction hash, and view details of the transactions.

Accounts

The **Accounts** view provides access to the custodial, multi-partition Wallet Service within the Oracle Blockchain Platform Console. This service enables you to perform essential wallet operations, including:

- Creating up to three wallets per user
- Viewing all wallets owned by the user
- Suspending and reactivating wallets
- Signing transactions with a selected wallet

These wallets are custodial, and designed for use with the Besu blockchain network. You can create new wallets directly from the console, or upload existing wallet keys (by using `.pem` files) to associate external wallets with your account.

Currently, the **Accounts** view supports only a single partition that is linked to the Besu instance.

Logs

The **Logs** view allows you to tail or view the last N lines of logs from each container in any pod in the namespaces of the underlying Kubernetes cluster.

You can search logs by instance, pod, and container. You can also specify the number of lines to display and apply filter expressions to help identify specific log messages.

You can also download log files directly from the console. For advanced log management, you can configure log aggregation using persistent logging with Oracle Cloud Infrastructure (OCI) or integrate with external tools such as `fluentd` for a unified logging approach.

6
RPC Proxy

The RPC proxy is the primary interface for client applications. It forwards requests to the configured upstream node endpoints.

The RPC proxy provides an authenticated and authorized access layer for Ethereum JSON-RPC requests. It also provides specialized features for signing transactions, running queries, and managing wallets.

Authentication

Secured endpoints require a bearer token in the authorization header:

```
Authorization: Bearer <access_token>
```

The proxy validates the token in accordance with the deployment configuration (for example, signature validation, expiration, issuer, and audience).

You can use the following command to generate a bearer token by using the access token endpoint that is available in Blockchain Platform Manager.

```
curl -sS -X POST \
  "<Access Token Endpoint – fetched from Blockchain Platform Manager>" \
  -H "Content-Type: application/x-www-form-urlencoded" \
  -d "grant_type=password" \
  -d "client_id=<CLIENT_ID>" \
  -d "client_secret=<CLIENT_SECRET>" \
  -d "username=<USERNAME>" \
  -d "password=<PASSWORD>"
```

The following text shows an example response. You then use the value of `access_token` as the bearer token for subsequent calls.

```
{
  "access_token": "eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9....",
  "expires_in": 300,
  "token_type": "Bearer"
    ...
}
```

Authorization

Access is controlled using the following group membership, which is conveyed in the token claims.

- JSON-RPC access requires membership in the following group: `BESU_RPC_GW_<instance_id>`
- Instance metadata APIs also require membership in the following group: `OBP_<BPMID>_CP_ADMIN`

OpenAPI Specification

The RPC proxy publishes an OpenAPI specification for its REST endpoints. The following text shows an example request.

```
curl -sS \
  "<RPC_PROXY_BASE_URL>/openapi" \
  -H "Authorization: Bearer <access token>" \
  -H "Accept: application/yaml"
```

The following types of APIs are exposed by the RPC proxy. For full details see the OpenAPI specification at RPC API for Oracle Blockchain Platform Enterprise Edition for Hyperledger Besu.

- JSON-RPC proxy API: Provides an authenticated and authorized entry point for Ethereum JSON-RPC methods such as `eth_call`, `eth_getBalance`, `eth_sendRawTransaction`, and `eth_getTransactionReceipt`.
- Instance/runtime information API: Provides instance metadata intended for viewing the genesis file or downloading the metadata `.zip` file to use when creating a participant instance.
- Contract registry API: Provides endpoints for managing and querying stored contract metadata. For example, Application Binary Interface (ABI), bytecode, manifests, implementations, and storage layouts.
- Wallet API: Provides endpoints for managing wallets, incluiding wallet creation and listing, key upload, and wallet activation.

7

Monitor Besu Metrics with Prometheus

You can use Prometheus and kube-prometheus-stack to retrieve metrics from the Besu nodes running in Kubernetes clusters.

To install monitoring, you use Helm, kube-prometheus-stack, and the predefined values file (`monitoring.yml`) that is maintained in the `Consensys/quorum-kubernetes` repository. In this scenario, Prometheus operates inside an Istio service mesh and securely scrapes metrics via mutual TLS. The kube-prometheus-stack package installs the following software.

- Prometheus
- Prometheus Operator
- Grafana
- Alertmanager
- Standard Kubernetes exporters

In the kube-prometheus-stack architecture, Prometheus discovers scrape targets by using a Kubernetes custom resource called a `ServiceMonitor`. This resource defines the services to scrape, the ports and paths that expose metrics, and configuration for TLS and mutual TLS. `ServiceMonitor` resources for components such as kube-state-metrics and node-exporter are automatically created when you use Helm to install kube-prometheus-stack. However, to scrape Oracle Blockchain Platform Enterprise Edition for Hyperledger Besu metrics, you must create and manage your own `ServiceMonitor` resources.

Install the following prerequisites.

- Helm v3.x. You can verify your Helm version by running the following command.

  ```
  helm version
  ```

- kubectl

You must also have command-line access to your Kubernetes cluster. For more information, see Connect to Oracle Kubernetes Engine.

1. Install kube-prometheus-stack by using the `monitoring.yml` file that is compatible with Besu.
   a. Enter the following command to download the file. The monitoring.yml file is maintained in the Consensys/quorum-kubernetes repository.

      ```
      curl -o monitoring.yml \
      https://raw.githubusercontent.com/Consensys/quorum-kubernetes/master/
      helm/values/monitoring.yml
      ```

   b. If needed, update the Grafana administrator password and configure alert receivers (for example, email or Slack) in the `monitoring.yml` before deployment.

c. If it is not already present, add the Prometheus community Helm repository

```
helm repo add prometheus-community https://prometheus-
community.github.io/helm-charts
helm repo update
```

d. Install the monitoring stack in a namespace dedicated for monitoring. In the following command, the namespace is called `monitoring`.

```
helm install monitoring prometheus-community/kube-prometheus-stack \
  --version 34.10.0 \
  --namespace monitoring \
  --create-namespace \
  --values monitoring.yml
```

e. Run the following command to check deployment status.

```
kubectl get pods -n monitoring -l release=monitoring
```

The installation process deploys the monitoring stack into the `monitoring` namespace, applies overrides from the monitoring.yml file for Besu compatibility, and creates the standard Kubernets `ServiceMonitor` resources that the stack requires.

2. Add Istio annotations to the `monitoring.yml` file to enable Istio sidecar injection for Prometheus. Prometheus must join the Istio mesh to scrape Besu endpoints via mutual TLS. Open the `monitoring.yml` file for editing, and find the following section.

```
prometheus:
  prometheusSpec:
    podMetadata:
```

Add the following Istio annotations.

```
prometheus:
  prometheusSpec:
    podMetadata:
      annotations:
        sidecar.istio.io/inject: "true"
        sidecar.istio.io/userVolumeMount: |
          [{"name": "istio-certs", "mountPath": "/etc/istio-certs",
"readOnly": true}]
        proxy.istio.io/config: |
          proxyMetadata:
            OUTPUT_CERTS: /etc/istio-certs
          proxyMetadata.INBOUND_CAPTURE_PORTS: ""
```

These annotations add Prometheus to the Istio mesh, configures Istio to write workload mTLS certificates to a shared volume, prevents Envoy from intercepting Prometheus inbound traffic, and makes certificates generated by Istio available to the Prometheus container.

3. Add volumes and volume mounts information to the `monitoring.yml` file to mount Istio certificates into Prometheus. Find the following text in the `monitoring.yml` file.

Find the `volumes` section, which is initially empty.

```
volumes: []
```

Update the `volumes` section as shown in the following text.

```
volumes:
  - name: istio-certs
    emptyDir:
      medium: Memory
```

Find the `volumeMounts` section, which is initially empty.

```
volumeMounts: []
```

Update the `volumeMounts` section as shown in the following text.

```
volumeMounts:
  - name: istio-certs
    mountPath: /etc/prom-certs
    readOnly: true
```

Certificates generated by Istio are now available inside Prometheus at `/etc/prom-certs/`.

4. Apply the updated configuration.
   a. Upgrade the existing Helm release with the modified values file.

      ```
      helm upgrade monitoring prometheus-community/kube-prometheus-stack \
        --namespace monitoring \
        --values monitoring.yml
      ```

   b. Verify that the Prometheus pod restarted and is running with an Istio sidecar.

      ```
      kubectl get pods -n monitoring | grep prometheus
      ```

5. Create a `ServiceMonitor` resource for Besu metrics, using the following example.

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: obp-besu-rpc-metrics
  namespace: <besu-namespace>
  labels:
    release: monitoring   # Must match Prometheus serviceMonitorSelector
spec:
  selector:
    matchLabels:
      app: besu
      besu-role: rpc
  namespaceSelector:
    matchNames:
```

```
      - <besu-namespace>
  endpoints:
    - port: metrics
      path: /metrics
      interval: 30s
      scheme: https
      tlsConfig:
        caFile: /etc/prom-certs/root-cert.pem
        certFile: /etc/prom-certs/cert-chain.pem
        keyFile: /etc/prom-certs/key.pem
        insecureSkipVerify: true
```

The certificate paths must match the mounted location (`etc/prom-certs`). The `release` label must match the `serviceMonitorSelector` value in Prometheus. The previous example scrapes metrics from RPC nodes (`besu-role: rpc`). To scrape metrics from boot nodes, validator nodes, or archive nodes, create separate `ServiceMonitor` resources and edit the `besu-role` label accordingly (`bootnode`, `validator`, or `archive`).

6. Verify that metrics are being collected in Prometheus.
   a. Run the following command to forward the port of the Prometheus service.

      ```
      kubectl port-forward -n <namespace> prometheus-monitoring-kube-
      prometheus-prometheus-
      ```

   b. Open `http://localhost:9090` and then select **Status**, and then **Targets**. Confirm that the Besu targets are in the `UP` state.
   c. Run a sample query, such as the following example.

      ```
      besu_blockchain_chain_head_transaction_count
      ```

7. Access Grafana to confirm that metrics are being collected. Run the following command to forward the port of the Grafana service.

   ```
   kubectl port-forward svc/monitoring-grafana -n <namespace> 3000:80
   ```

   - URL: `http://localhost:3000`
   - User name: `admin`
   - Run the following command to retrieve the password:

     ```
     kubectl get secret -n <namespace> monitoring-grafana \
     -o jsonpath="{.data.admin-password}" | base64 --decode ; echo
     ```

8

Secure Your Besu Network

The network is configured to allow peer-to-peer communication so that participants can join easily. You can restrict access further if needed.

By default, the network exposes a small set of ports that are used for Besu peer-to-peer communication. These ports are required so that nodes in different clusters can discover and connect to each other. Ports are exposed by using the cloud load balancer. Worker nodes remain in private subnets. This configuration prioritizes onboarding and reliability. You can further restrict access as needed by updating cloud network security rules (for example, with security lists or network security groups).

1. Identify the IP address or IP address range of the participant's Besu instance. In an OCI environment, this is the NAT IP address of the participant OKE cluster.
2. Update your network security rules to allow inbound traffic from only approved participant IP addresses and to block all other sources.
   By default, when a load balancer service is created in OKE, the security list automatically adds rules to open the node ports that are associated with this service. Therefore, on instance creation the port range from 30303 to 30310 is automatically opened by means of rules in the security list.

   For maximum isolation, the founder instance can remove these rules from the security list. Security lists are applied at the subnet level, so if an instance or load balancer service is created in the same subnet, the rules related to node ports might be applied again automatically. When this happens, you must remove those rules again.
3. If you want to join a participant instance to the founder network, get the NAT IP address mentioned in step one and create a network security group ingress rule that allows traffic from the NAT IP address as the source address to the specified participant address with a destination port range from 30303 to 30310.
4. Identify the dedicated load balancer that is associated with only this instance, and associate the network security group with that load balancer.
   After you associate the network security group with the load balancer, only the explicitly allowed participate instance can discover and connect to the founder instance.

9

Develop and Use Smart Contracts

Use Oracle Blockchain Platform Enterprise Edition for Hyperledger Besu to download sample Solidity contracts, work with tokenized assets, and call Solidity methods for account and token contracts.

- Sample Smart Contracts
  - Find sample contracts on the Developer Tools page in the service console. Download the Solidity source code or the application binary interface (ABI).
  - Use the ABI with RPC proxy API calls to interact with deployed smart contracts.
  - Import the source into a Solidity development environment (for example, Hardhat or Remix) to modify or extend the contracts.
- Install Smart Contracts for Digital Assets
  - Use the provided digital assets packages to work with tokenized assets: one extends the ERC-20 standard and one extends the ERC-1155 standard.
- Solidity Smart Contract API
  - Call Solidity methods that support tokens based on the ERC-20 and ERC-1155 standards.

Sample Smart Contracts

Oracle Blockchain Platform Enterprise Edition for Hyperledger Besu includes sample smart contracts, which are available on the Developer Tools page in the service console.

Select the **Download options** menu for each contract to download the Solidity source code or the application binary interface (ABI). You can use this information to interact with deployed smart contracts by using RPC proxy API calls. You can also import the source code into a Solidity development environment such as Hardhat or Remix and make changes or enhancements as needed.

Hash Time-Locked Contract (HTLC)

This contract is deployed when you create an instance. The Hash Time-Locked Contract enables atomic exchanges of ETH and ERC-20 tokens, where a receiver claims a payment prior to a deadline or the payment is refunded to the sender. The contract supports the following operations.

- Exchange ETHor ERC-20 tokens.
- Lock funds for a receiver under a secret condition for a specified amount of time.
- Claim locked funds by providing the correct secret before the lock expires.
- Refund unclaimed locked funds to the sender after the lock expires.
- Get the status of a locked exchange, including whether is has been claimed or refunded, the locked amount, and the lock deadline.

Soul Bound Token (SBT)

This contract is not predeployed. You must download and compile the source code and then deploy the contract by using tools such as Hardhat or the RPC proxy API. The Soul Bound Token contract implements a non-transferable NFT-style credential that is tied to a wallet identity. Tokens can only be issued, verified, revoked, or burned by the token holder. They cannot be sold or moved. The contract supports the following operations.

- The contract owner (administrator) can issue or revoke credentials (mint or burn soul-bound tokens in a wallet).
- Applications can check if a wallet contains a soul-bound token and read available token metadata (URI).
- The token holder can burn their soul-bound token.

LockBox

This contract is deployed when you create an instance. The LockBox contract implements a time-locked vault that holds ETH and ERC-20 tokens and then releases the contents to a designated owner at a specific time. The contract supports the following operations.

- Deposit funds into the vault by sending ETH directly or transferring ERC-20 tokens.
- Get vault details, including who the beneficiary/owner is, what the release time is, and what tokens/ETH are in the vault. Any user can get this information.
- Release funds at a specified time. After the specified time stamp, the owner can call the release action to retrieve the funds.

Install Smart Contracts for Digital Assets

Oracle Blockchain Platform Enterprise Edition for Hyperledger Besu includes smart contracts that you can use to work with tokenized assets.

Oracle Blockchain Platform Enterprise Edition for Hyperledger Besu provides two digital assets packages: one that supports an extended version of the ERC-20 standard, and one that supports an extended version of the ERC-1155 standard. ERC-20 is the standard for fungible tokens that are created on the Ethererum network. ERC-1155 is a standard that supports both fungible and non-fungible tokens (NFTs).

Install ERC-20 Smart Contracts

You can use the smart contracts provided with the platform to explore stablecoin and wholesale central bank digital currency (CBDC) scenarios.

The following instructions have been tested on macOS and Linux systems.

1. Enter the following command to extract the ERC-20 package.

   ```
   unzip obp-da-erc20.zip -d <path/to/directory>
   ```

2. Use nvm to install Node.js and npm. For best results, use Node.js version 20.17.0 and npm version 10.8.2.

   For more information about nvm, see Node Version Manager on GitHub.

3. Run the following command to confirm that version 2.28.0 of Hardhat is installed.

```
npx hardhat –version
```

4. Navigate to the directory where you extracted the package, run the following command to compile the reference implementations of the smart contracts.

```
npx hardhat clean & npx hardhat compile
```

5. Run the following command to run unit tests on the contracts.

```
npx hardhat test
```

6. Deploy the wholesale CBDC reference implementation.
   a. Run the following commands to set the Hardhat configuration variables, as described in the following table.

```
npx hardhat vars set BESU_CHAIN_ID <chain_id>
npx hardhat vars set BESU_RPC_URL <rpc_url>
npx hardhat vars set BESU_RPX_PROXY_AUTH_TOKEN "Bearer <auth-token>"
npx hardhat vars set BESU_ACCOUNTS <account keys>
npx hardhat vars set BESU_ALLOW_SELF_SIGNED <true or false>
```

Variable	Description	Default value
BESU_CHAIN_ID	Chain ID of the Besu network	1337
BESU_RPC_URL	URL of the RPC proxy with a passthrough endpoint	http://127.0.0.1:8545
BESU_RPX_PROXY_AUTH_TOKEN	Auth token of the Oracle Blockchain Platform RPC proxy	''
BESU_ACCOUNTS	Account keys for deployment	""
BESU_ALLOW_SELF_SIGNED	Flag that allows Hardhat to communicate with Hyperledger Besu by using self-signed certificates without providing CA certificatess (NODE_TLS_REJECT_UNAUTHORIZED=0)	true

   In the limited availability version of the platform, the contract is deployed by using the account keys that are configured in the Hardhat variable. Use one of the pre-funded account keys when you deploy.

   b. Run the following command to set the path for the OpenZeppelin manifest file, adjusting the path as needed for your system.

```
export MANIFEST_DEFAULT_DIR=.openzeppelin/tests
```

   c. Run the Hardhat script to deploy the account and the wholesale CBDC smart contract.

```
npx hardhat run scripts/deploy-wcbdc.ts –-network besu
```

The script runs and shows the contract addresses for the account and the wholesale CBDC contract. Record the contract addresses and the OpenZeppelin manifest files for use when interacting with the contract via RPC proxy HTTP calls.

7. Deploy the stablecoin reference implementation.
   a. Set the Hardhat configuration variables as described in the previous step.
   b. Run the following command to set the path for the OpenZeppelin manifest file, adjusting the path as needed for your system.

      ```
      export MANIFEST_DEFAULT_DIR=.openzeppelin/tests
      ```

   c. Run the Hardhat script to deploy the account and the stablecoin smart contract.

      ```
      npx hardhat run scripts/deploy-stablecoin.ts –-network besu
      ```

      The script runs and shows the contract addresses for the account and the wholesale CBDC contract. Record the contract addresses and the OpenZeppelin manifest files for use when interacting with the contract via RPC proxy HTTP calls.

Install ERC-1155 Smart Contracts

You can use the smart contracts provided with the platform to explore scenarios that support a bond marketplace and art collection marketplace that uses loyalty points.

The following instructions have been tested on macOS and Linux systems.

1. Enter the following command to extract the ERC-1155 package.

   ```
   unzip obp-da-erc1155.zip -d <path/to/directory>
   ```

2. Use nvm to install Node.js and npm. For best results, use Node.js version 20.17.0 and npm version 10.8.2.

   For more information about nvm, see Node Version Manager on GitHub.

3. Run the following command to confirm that version 2.28.0 of Hardhat is installed.

   ```
   npx hardhat –version
   ```

4. Navigate to the directory where you extracted the package, and then run the following command to compile the reference implementations of the smart contracts.

   ```
   npx hardhat clean & npx hardhat compile
   ```

5. Run the following command to run unit tests on the contracts.

   ```
   npx hardhat test
   ```

6. Deploy the art collection reference implementation.
   a. Run the following commands to set the Hardhat configuration variables, as described in the following table.

      ```
      npx hardhat vars set BESU_CHAIN_ID <chain_id>
      npx hardhat vars set BESU_RPC_URL <rpc_url>
      npx hardhat vars set BESU_RPX_PROXY_AUTH_TOKEN "Bearer <auth-token>"
      ```

```
npx hardhat vars set BESU_ACCOUNTS <account keys>
npx hardhat vars set BESU_ALLOW_SELF_SIGNED <true or false>
```

Variable	Description	Default value
BESU_CHAIN_ID	Chain ID of the Besu network	1337
BESU_RPC_URL	URL of the RPC proxy with a passthrough endpoint	http://127.0.0.1:8545
BESU_RPX_PROXY_AUTH_TOKEN	Auth token of the Oracle Blockchain Platform RPC proxy	''
BESU_ACCOUNTS	Account keys for deployment	""
BESU_ALLOW_SELF_SIGNED	Flag that allows Hardhat to communicate with Hyperledger Besu by using self-signed certificates without providing CA certificatess (NODE_TLS_REJECT_UNAUTHORIZED=0)	true

In the limited availability version of the platform, the contract is deployed by using the account keys that are configured in the Hardhat variable. Use one of the pre-funded account keys when you deploy.

b. Run the following command to set the path for the OpenZeppelin manifest file, adjusting the path as needed for your system.

```
export MANIFEST_DEFAULT_DIR=.openzeppelin/tests
```

c. Run the Hardhat script to deploy the account and the art collection smart contract.

```
npx hardhat run scripts/art-loyalty/deploy-art-loyalty.ts --network besu
```

The script runs and shows the contract addresses for the account and the art collection contract. Record the contract addresses and the OpenZeppelin manifest files for use when interacting with the contract via RPC proxy HTTP calls.

7. Deploy the bond marketplace reference implementation.

a. Set the Hardhat configuration variables as described in the previous step.

b. Run the following command to set the path for the OpenZeppelin manifest file, adjusting the path as needed for your system.

```
export MANIFEST_DEFAULT_DIR=.openzeppelin/tests
```

c. Run the Hardhat script to deploy the account and the bond marketplace smart contract.

```
npx hardhat run scripts/bond/deploy-bond.ts --network besu
```

The script runs and shows the contract addresses for the account and the bond marketplace contract. Record the contract addresses and the OpenZeppelin manifest files for use when interacting with the contract via RPC proxy HTTP calls.

Solidity Smart Contract API

Oracle Blockchain Platform Enterprise Edition for Hyperledger Besu provides Solidity methods that you can use to work with Solidity smart contracts.

The API supports tokens based on the ERC-20 standard and the ERC-1155 standard. The ERC-20 and ERC-1155 implementations comprise an account smart contract and a token smart contract. The account smart contract provides identity and access control methods. The token smart contract provides functions specifically related to tokens.

ERC-20 Account Contract Methods

Methods for Account Management

`createAccount`
This method creates an account for a specified user. Accounts track a user's token balance and on-hold balance, and must be created for all users who will have tokens at any point. This method can be called only by a `Token Admin` or an `Org Admin` of the specified organization.

Parameters:

- `userId: string` – The user name or email ID of the user. The user ID string cannot be empty.
- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization. The organization ID cannot be empty.
- `userAddress: address` – The wallet address of the user. The address cannot be zero.
- `maxDailyAmount: int256` – The maximum amount of tokens that can be used in transactions daily. If `maxDailyAmount` is `-1`, there is no limit.
- `maxDailyTransactions: int256` – The maximum number of transactions that can be completed daily. If `maxDailyTransactions` is `-1`, there is no limit.

`deleteAccount`
This method deletes the account of a specified user. An account can be deleted only if the token balance is zero. This method can be called only by a `Token Admin` or an `Org Admin` of the specified organization.

Parameters:

- `userAddress: address` – The wallet address of the user. The address cannot be zero.

`getAccountStatus`
This method gets the current status of the specified account. This method can be called only by a `Token Admin`, `Token Auditor`, `Org Admin` of the specified organization, or by the specified user.

Parameters:

- `userAddress: address` – The wallet address of the user. The address cannot be zero.

Returns:

- A value for the `enum AccountStatus` object, `0`, `1`, or `2`.

```
enum AccountStatus {
active,
suspended,
deleted
}
```

`getAccountByAddress`
This method gets the account information for a specified user. This method can be called only by a `Token Admin`, `Token Auditor`, `Org Admin` of the specified organization, or by the specified user.

Parameters:

- `userAddress: address` – The wallet address of the user. The address cannot be zero.

Return Value Example:

```
{
    "userId": "userA",
    "orgId": "orgA",
    "accountAddress": "0x0fbdc686b912d7722dc86510934589e0aaf3b55a",
    "dailyLimits": {
          "isMaxDailyTxnEnabled": true,
          "isMaxDailyAmtEnabled": true,
          "dailyAmount": 100,
          "dailyTransactions": 5,
          "maxDailyAmount": 100000,
          "maxDailyTransactions": 50
    },
    "status":0
}
```

`getAllAccounts`
This method gets the account information for all accounts that fall in a specified range. This method can be called only by a `Token Admin` or `Token Auditor`.

Parameters:

- `offset: uint256` – The offset index to get the accounts from.
- `limit: uint256` – The number of items to retrieve.

Return Value Example:

```
Return (UserAccount[] memory items, uint256 total, uint256 nextOffset)


[{
    "userId": "userA",
    "orgId": "orgA",
    "accountAddress": "0x0fbdc686b912d7722dc86510934589e0aaf3b55a",
    "dailyLimits": {
          "isMaxDailyTxnEnabled": true,
```

```
            "isMaxDailyAmtEnabled": true,
            "dailyAmount": 100,
            "dailyTransactions": 5,
            "maxDailyAmount": 100000,
            "maxDailyTransactions": 50
        },
        "status":0
}]


total: 15


nextoffset: 6
```

`activateAccount`
This method activates a user account. This method can be called only by a `Token Admin` or `Org Admin` of the specified organization.

Parameters:

- `userAddress: address` – The wallet address of the user. The address cannot be zero.

`suspendAccount`
This method suspends a user account. To delete an account, the account balance must be zero. This method can be called only by a `Token Admin` or `Org Admin` of the specified organization.

Parameters:

- `userAddress: address` – The wallet address of the user. The address cannot be zero.

`setMaxDailyAmount`
This method sets the maximum amount of tokens that can be used in transactions daily. This method can be called only by a `Token Admin` or `Org Admin` of the specified organization.

Parameters:

- `count: int256` – The maximum amount of tokens that can be used in transactions daily.
- `userAddress: address` – The wallet address of the user. The address cannot be zero.

`setMaxDailyTransactionCount`
This method sets the maximum number of transactions that can be completed daily. This method can be called only by a `Token Admin` or `Org Admin` of the specified organization.

Parameters:

- `value: int256` – The maximum number of transactions that can be completed daily.
- `userAddress: address` – The wallet address of the user. The address cannot be zero.

Methods for Administrator Management

`addTokenAdmin`
This method adds a user as a `Token Admin`. This method can be called only by a `Token Admin`.

Parameters:

- `adminDetails: JSON` – An object that contains details about the user to add as a `Token Admin`, as shown in the following example.

```
{
    "userId": "tokenAdmin1",
    "orgId": "orgA",
    "accountAddress": "0x0fbdc686b912d7722dc86510934589e0aaf3b55a"
}
```

`removeTokenAdmin`
This method removes a user as a `Token Admin`. This method can be called only by a `Token Admin`.

Parameters:

- `userAddress: address` – The wallet address of the user. The address cannot be zero.

`getAllTokenAdmins`
This method returns a list of all users who are a `Token Admin`. This method can be called only by the `Token Admin` or `Token Auditor`.

Parameters:

- none

Return Value Example:

```
[{
    "userId": "tokenadmin1",
    "orgId": "orgA",
    "accountAddress": "0x0fbdc686b912d7722dc86510934589e0aaf3b55a"
},
{
    "userId": "tokenadmin2",
    "orgId": "orgB",
    "accountAddress": "0xfe3b557e8fb62b89f4916b721be55ceb828dbd73"
}]
```

`isTokenAdmin`
This method checks whether the specified user is a `Token Admin`. This method can be called only by a `Token Admin`, `Token Auditor`, `Org Admin`, or `Org Auditor`.

Parameters:

- `userAddress: address` – The wallet address of the user. The address cannot be zero.

Returns:

- The method returns `true` if the specified user is a `Token Admin`, otherwise it returns `false`.

`addOrgAdmin`
This method adds a user as a `Org Admin`. This method can be called only by a `Token Admin` or `Org Admin` of the specified organization.

Parameters:

- `adminDetails: JSON` – An object that contains details about the user to add as an `Org Admin`, as shown in the following example.

```
{
    "userId": "tokenAdmin1",
    "orgId": "orgA",
    "accountAddress": "0x0fbdc686b912d7722dc86510934589e0aaf3b55a"
}
```

`removeOrgAdmin`
This method removes a user as an `Org Admin`. This method can be called only by a `Token Admin` or `Org Admin` of the specified organization.

Parameters:

- `userAddress: address` – The wallet address of the user. The address cannot be zero.

`getAllOrgAdmins`
This method returns a list of all users who are an `Org Admin`. This method can be called only by the `Token Admin`, `Token Auditor`, `Org Admin`, or `Org Auditor`.

Parameters:

- none

Return Value Example:

```
[{
    "userId": "orgadmin1",
    "orgId": "orgA",
    "accountAddress": "0x0fbdc686b912d7722dc86510934589e0aaf3b55a"
},
{
    "userId": "orgadmin2",
    "orgId": "orgB",
    "accountAddress": "0xfe3b557e8fb62b89f4916b721be55ceb828dbd73"
}]
```

`isOrgAdmin`
This method checks whether the specified user is an `Org Admin`. This method can be called only by a `Token Admin`, `Token Auditor`, `Org Admin`, or `Org Auditor`.

Parameters:

- `userAddress: address` – The wallet address of the user. The address cannot be zero.

Returns:

- The method returns `true` if the specified user is a `Org Admin`, otherwise it returns `false`.

Methods for Auditor Management

`addTokenAuditor`
This method adds a user as a `Token Auditor`. This method can be called only by a `Token Admin`.

Parameters:

- `adminDetails: JSON` – An object that contains details about the user to add as a `Token Auditor`, as shown in the following example.

```
{
    "userId": "tokenAuditor1",
    "orgId": "orgA",
    "accountAddress": "0x0fbdc686b912d7722dc86510934589e0aaf3b55a"
}
```

`removeTokenAuditor`
This method removes a user as a `Token Auditor`. This method can be called only by a `Token Admin`.

Parameters:

- `userAddress: address` – The wallet address of the user. The address cannot be zero.

`getAllTokenAuditors`
This method returns a list of all users who are a `Token Auditor`. This method can be called only by the `Token Admin` or `Token Auditor`.

Parameters:

- none

Return Value Example:

```
[{
    "userId": "tokenAuditor1",
    "orgId": "orgA",
    "accountAddress": "0x0fbdc686b912d7722dc86510934589e0aaf3b55a"
},
{
    "userId": "tokenAuditor2",
    "orgId": "orgB",
    "accountAddress": "0xfe3b557e8fb62b89f4916b721be55ceb828dbd73"
}]
```

`isTokenAuditor`
This method checks whether the specified user is a `Token Auditor`. This method can be called by any user.

Parameters:

- `userAddress: address` – The wallet address of the user. The address cannot be zero.

Returns:

- The method returns `true` if the specified user is a `Token Auditor`, otherwise it returns `false`.

`addOrgAuditor`
This method adds a user as a `Org Auditor`. This method can be called only by a `Token Admin` or `Org Admin` of the specified organization.

Parameters:

- `adminDetails: JSON` – An object that contains details about the user to add as an `Org Auditor`, as shown in the following example.

```
{
    "userId": "tokenAuditor1",
    "orgId": "orgA",
    "accountAddress": "0x0fbdc686b912d7722dc86510934589e0aaf3b55a"
}
```

`removeOrgAuditor`
This method removes a user as an `Org Auditor`. This method can be called only by a `Token Admin` or `Org Admin` of the specified organization.

Parameters:

- `userAddress: address` – The wallet address of the user. The address cannot be zero.

`getAllOrgAuditors`
This method returns a list of all users who are an `Org Auditor`. This method can be called only by the `Token Admin`, `Token Auditor`, `Org Admin`, or `Org Auditor`.

Parameters:

- none

Return Value Example:

```
[{
    "userId": "orgAuditor1",
    "orgId": "orgA",
    "accountAddress": "0x0fbdc686b912d7722dc86510934589e0aaf3b55a"
},
{
    "userId": "orgAuditor2",
    "orgId": "orgB",
    "accountAddress": "0xfe3b557e8fb62b89f4916b721be55ceb828dbd73"
}]
```

`isOrgAuditor`
This method checks whether the specified user is an `Org Auditor`. This method can be called by any user.

Parameters:

- `userAddress: address` – The wallet address of the user. The address cannot be zero.

Returns:

- The method returns `true` if the specified user is a `Org Auditor`, otherwise it returns `false`.

Methods for Role Management

`addRole`
This method adds a role to a specified user. This method can be called only by a `Token Admin` or `Org Admin` of the specified organization.

Parameters:

- `userAddress: address` – The wallet address of the user. The address cannot be zero.
- `role: string` – The name of the role to add to the specified user. For example, `minter`, `burner`, or `notary`.

`removeRole`
This method removes a role from a specified user. This method can be called only by a `Token Admin` or `Org Admin` of the specified organization.

Parameters:

- `userAddress: address` – The wallet address of the user. The address cannot be zero.
- `role: string` – The name of the role to remove from the specified user. For example, `minter`, `burner`, or `notary`.

`accountHasRole`
This method checks whether a user has a specified role. This method can be called by any user.

Parameters:

- `role: string` – The Keccak-256 hash of the name of the role (`minter`, `burner`, or `notary`) to search for.
- `userAddress: address` – The wallet address of the user. The address cannot be zero.

Returns:

- The method returns `true` if the user has the specified role, otherwise it returns `false`.

ERC-20 Token Contract Methods

Methods for Token Configuration Management

`__ERC20Token_init`
This method is called when the token contract is deployed. This method can be called only by a `Token Admin`.

Parameters:

- `name: string` – The name of the token.
- `symbol: string` – The symbol of the token.
- `accountContract: address` – The address of the account contract.

`initializeERC20Token`
This method initializes an ERC-20 token. This method can be called only by a `Token Admin`.

Parameters:

- `token: JSON` – An object that defines the token, as shown in the following example.

```
{
    "tokenId": "WCBDC-100",
    "tokenName": "wholesale cbdc",
    "tokenDesc": "this is wcbdc contract",
```

```
    "tokenStandard": "ttf+",
    "tokenType": "fungible",
    "tokenUnit": "fractional",
    "behaviors": ["mintable", "burnable", "transferable", "roles",
"holdable", "pausable"],
    "decimals": 2,
    "mintable": { "maxMintQuantity": 1000000, "mintApprovalRequired":
false },
    "burnable": { "burnApprovalRequired": false }
}
```

`getToken`
This method gets details for a token. This method can be called only by a `Token Admin`, `Token Auditor`, `Org Admin`, or `Org Auditor`.

Parameters:

- none

Return Value Example:

```
{
    "tokenId": "WCBDC-100",
    "tokenName": "wholesale cbdc",
    "tokenDesc": "this is wcbdc contract",
    "tokenStandard": "ttf+",
    "tokenType": "fungible",
    "tokenUnit": "fractional",
    "behaviors": ["mintable", "burnable", "transferable", "roles",
"holdable", "pausable"],
    "decimals": 2,
    "mintable": { "maxMintQuantity": 1000000, "mintApprovalRequired": false },
    "burnable": { "burnApprovalRequired": false }
}
```

`decimals`
This method gets the decimal value for a token. This method can be called only by a `Token Admin`, `Token Auditor`, `Org Admin`, or `Org Auditor`.

Parameters:

- none

Returns

- A `uint8` value of the number of decimal places.

`__ERC20Token_init`
This method is called when token contract is deployed. This method can be called only by a `Token Admin`.

Parameters:

- `name: string` – The name of the token.
- `symbol: string` – The symbol of the token.

- `accountContract: address` – The address of the account contract.

`initializeERC20Token`
This method initializes an ERC-20 token. This method can be called only by a `Token Admin`.

Parameters:

- `token: JSON` – An object that defines the token, as shown in the following example.

```
{
    "tokenId": "WCBDC-100",
    "tokenName": "wholesale cbdc",
    "tokenDesc": "this is wcbdc contract",
    "tokenStandard": "ttf+",
    "tokenType": "fungible",
    "tokenUnit": "fractional",
    "behaviors": ["mintable", "burnable", "transferable", "roles",
"holdable", "pausable"],
    "decimals": 2,
    "mintable": { "maxMintQuantity": 1000000, "mintApprovalRequired":
false },
    "burnable": { "burnApprovalRequired": false }
}
```

`getToken`
This method gets details for a token. This method can be called only by a `Token Admin`, `Token Auditor`, `Org Admin`, or `Org Auditor`.

Parameters:

- none

Return Value Example:

```
{
    "tokenId": "WCBDC-100",
    "tokenName": "wholesale cbdc",
    "tokenDesc": "this is wcbdc contract",
    "tokenStandard": "ttf+",
    "tokenType": "fungible",
    "tokenUnit": "fractional",
    "behaviors": ["mintable", "burnable", "transferable", "roles",
"holdable", "pausable"],
    "decimals": 2,
    "mintable": { "maxMintQuantity": 1000000, "mintApprovalRequired": false },
    "burnable": { "burnApprovalRequired": false }
}
```

`cap`
This method gets the maximum token supply (cap). This method can be called only by a `Token Admin`, `Token Auditor`, `Org Admin`, or `Org Auditor`.

Parameters:

- none

Returns

- A `uint256` value of the token supply cap.

`balanceOf`
This method gets the token balance for the specified user. This method can be called only by a `Token Admin`, `Token Auditor`, `Org Admin`, `Org Auditor`, or by the specified user.

Parameters:

- `userAddress: address` – The wallet address of the user. The address cannot be zero.

Returns

- A `uint256` value of the token balance.

Token Behavior Management - Mintable Behavior

`mint`
This method creates (mints) tokens. This method can be called by any user with the minter role.

Parameters:

- `to: address` – The wallet address of the user. The address cannot be zero.
- `value: uint256` – The quantity of tokens to mint.

`batchMint`
This method creates (mints) tokens for more than one user at a time. This method can be called by any user with the minter role.

Parameters:

- `toList: address[]` – The list of wallet addresses of the users. The addresses cannot be zero.
- `amounts: uint256[]` – The list of quantities of tokens to mint.

`requestMint`
This method can be called by a minter to send a request to the notary to create a specified amount of tokens.

Parameters:

- `notary: address` – The wallet address of the notary. The address cannot be zero.
- `amount: uint256` – The quantity of tokens to mint.
- `expiration: uint256` – The expiration time for the request in epoch format.
- `opId: string` – The operation ID of the request.
- `info: InfoDetails` – An object specifying the category (`category`) and description (`description`) of the request.

`approveMint`
This method can be called by a notary to approve a minting request.

Parameters:

- `opId: string` – The operation ID of the request.

`rejectMint`
This method can be called by a notary to reject a minting request.

Parameters:

- `opId: string` – The operation ID of the request.

Token Behavior Management - Burnable Behavior

`burn`
This method deactivates (burns) tokens. This method can be called by any user with the burner role.

Parameters:

- `account: address` – The wallet address of the user. The address cannot be zero.
- `value: uint256` – The quantity of tokens to burn.

`batchBurn`
This method burns tokens for more than one user at a time. This method can be called by any user with the burner role.

Parameters:

- `toList: address[]` – The list of wallet addresses of the users. The addresses cannot be zero.
- `amounts: uint256[]` – The list of quantities of tokens to burn.

`requestBurn`
This method can be called by a burner to send a request to the notary to burn a specified amount of tokens.

Parameters:

- `notary: address` – The wallet address of the notary. The address cannot be zero.
- `amount: uint256` – The quantity of tokens to burn.
- `expiration: uint256` – The expiration time for the request in epoch format.
- `opId: string` – The operation ID of the request.
- `info: InfoDetails` – An object specifying the category (`category`) and description (`description`) of the request.

`approveBurn`
This method can be called by a notary to approve a burning request.

Parameters:

- `opId: string` – The operation ID of the request.

`rejectBurn`
This method can be called by a notary to reject a burning request.

Parameters:

- `opId: string` – The operation ID of the request.

Token Behavior Management - Transferable Behavior

`transfer`
This method transfers tokens to a specified user. This method can be called by any user with tokens.

Parameters:

- `to: address` – The wallet address of the receiver. The address cannot be zero.
- `value: uint256` – The quantity of tokens to transfer.

`batchTransfer`
This method transfers tokens to a specified list of users. This method can be called by any user with tokens.

Parameters:

- `toList: address[]` – The list of wallet addresses of the receivers. The addresses cannot be zero.
- `amounts: uint256[]` – The list of quantities of tokens to transfer.

Token Behavior Management - Delegable Behavior

`allowance`
This method delegates token spending to a specified user. This method can be called by any user with tokens.

Parameters:

- `owner: address` – The wallet address of the owner who is delegating the spending of tokens. The address cannot be zero.
- `spender: address` – The wallet address of the token spender. The address cannot be zero.

Returns

- A `uint256` amount of the tokens that were delegated for spending.

`approve`
This method approves the amount of tokens for a specified delegated spender. This method can be called by any user with tokens.

Parameters:

- `spender: address` – The wallet address of the token spender. The address cannot be zero.
- `value: uint256` – The quantity of tokens the spender is allowed to spend.

`transferFrom`
Delegated spenders use this method to transfer tokens.

Parameters:

- `from: address` – The wallet address of the sender. The address cannot be zero.
- `to: address` – The wallet address of the receiver. The address cannot be zero.
- `value: uint256` – The quantity of tokens to transfer.

Token Behavior Management - Pausable Behavior

`isPaused`
This method checks whether the contract is paused. This method can be called only by a `Token Admin`, `Token Auditor`, `Org Admin`, or `Org Auditor`.

Parameters:

- none

`pause`
This method pauses the contract. This method can be called only by a `Token Admin`.

Parameters:

- none

`unpause`
This method resumes the contract. This method can be called only by a `Token Admin`.

Parameters:

- none

Token Behavior Management - Holdable Behavior

`hold`
This method creates a hold on behalf of the token owner. This method can be called by any user with tokens.

Parameters:

- `to: address` – The wallet address of the receiver. The address cannot be zero.
- `notary: address` – The wallet address of the notary. The address cannot be zero.
- `amount: uint256` – The quantity of tokens to transfer.
- `expiration: uint256` – The expiration time for the request in epoch format.
- `opId: string` – The operation ID of the request.
- `holdType: string` – The type of hold. For example, `transfer`.
- `info: InfoDetails` – An object specifying the category (`category`) and description (`description`) of the request.

`executeHold`
This method approves a hold request. This method can be called only by the previously specified notary.

Parameters:

- `amount: uint256` – The quantity of tokens to approve.
- `opId: string` – The operation ID of the request.

`releaseHold`
This method rejects a hold request. This method can be called only by the previously specified notary.

Parameters:

- `opId: string` – The operation ID of the request.

`getOnHoldBalanceWithOperationId`
This method returns the on-hold balance for a specified operation ID. This method can be called by a `Token Admin` or `Token Auditor`, `Org Admin` or `Org Auditor` of the specified organization, or by a transaction participant (sender, recipient, notary).

Parameters:

- `opId: string` – The operation ID of the request.

Returns

- A `uint256` amount of the on-hold balance.

`getAccountOnHoldBalance`
This method returns the on-hold balance for a specified account. This method can be called by a `Token Admin` or `Token Auditor`, `Org Admin` or `Org Auditor` of the specified organization, or by a transaction participant (sender, recipient, notary).

Parameters:

- `account: address` – The wallet address of the account. The address cannot be zero.

Returns

- A `uint256` amount of the on-hold balance.

ERC-1155 Account Contract Methods

Methods for Account Management

`createAccount`
This method creates an account for a specified user. Accounts track a user's token balance, and must be created for all users who will have tokens at any point. This method can be called only by a `Token Admin`.

Parameters:

- `userId: string` – The user name or email ID of the user. The user ID string cannot be empty.
- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization. The organization ID cannot be empty.
- `userAddress: address` – The wallet address of the user. The address cannot be zero.

`deleteAccount`
This method deletes the account of a specified user. An account can be deleted only if the token balance is zero. This method can be called only by a `Token Admin`.

Parameters:

- `userAddress: address` – The wallet address of the user. The address cannot be zero.

`getAccountStatus`
This method gets the current status of the specified account. This method can be called only by a `Token Admin` or by the specified user.

Parameters:

- `userAddress: address` – The wallet address of the user. The address cannot be zero.

Returns:

- A value for the `enum AccountStatus` object, `0`, `1`, or `2`.

```
enum AccountStatus {
active,
suspended,
deleted
}
```

`getAccountByAddress`
This method gets the account information for a specified user. This method can be called only by a `Token Admin` or by the specified user.

Parameters:

- `userAddress: address` – The wallet address of the user. The address cannot be zero.

Return Value Example:

```
{
    "userId": "userA",
    "orgId": "orgA",
    "accountAddress": "0x0fbdc686b912d7722dc86510934589e0aaf3b55a",
    "status":0
}
```

`getAllAccounts`
This method gets the account information for all accounts that fall in a specified range. This method can be called only by a `Token Admin`.

Parameters:

- `offset: uint256` – The offset index to get the accounts from.
- `limit: uint256` – The number of items to retrieve.

Return Value Example:

```
Return (UserAccount[] memory items, uint256 total, uint256 nextOffset)


[{
    "userId": "userA",
    "orgId": "orgA",
    "accountAddress": "0x0fbdc686b912d7722dc86510934589e0aaf3b55a",
    "status":0
}]


total: 15


nextoffset: 6
```

`activateAccount`
This method activates a user account. This method can be called only by a `Token Admin`.

Parameters:

- `userAddress: address` – The wallet address of the user. The address cannot be zero.

`suspendAccount`
This method suspends a user account. To delete an account, the account balance must be zero. This method can be called only by a `Token Admin`.

Parameters:

- `userAddress: address` – The wallet address of the user. The address cannot be zero.

Methods for Administrator Management

`addTokenAdmin`
This method adds a user as a `Token Admin`. This method can be called only by a `Token Admin`.

Parameters:

- `adminDetails: JSON` – An object that contains details about the user to add as a `Token Admin`, as shown in the following example.

```
{
    "userId": "tokenAdmin1",
    "orgId": "orgA",
    "accountAddress": "0x0fbdc686b912d7722dc86510934589e0aaf3b55a"
}
```

`removeTokenAdmin`
This method removes a user as a `Token Admin`. This method can be called only by a `Token Admin`.

Parameters:

- `userAddress: address` – The wallet address of the user. The address cannot be zero.

`getAllTokenAdmins`
This method returns a list of all users who are a `Token Admin`. This method can be called only by the `Token Admin`.

Parameters:

- none

Return Value Example:

```
[{
    "userId": "tokenadmin1",
    "orgId": "orgA",
    "accountAddress": "0x0fbdc686b912d7722dc86510934589e0aaf3b55a"
},
{
    "userId": "tokenadmin2",
```

```
    "orgId": "orgB",
    "accountAddress": "0xfe3b557e8fb62b89f4916b721be55ceb828dbd73"
}]
```

`isTokenAdmin`
This method checks whether the specified user is a `Token Admin`. This method can be called only by a `Token Admin`.

Parameters:

- `userAddress: address` – The wallet address of the user. The address cannot be zero.

Returns:

- The method returns `true` if the specified user is a `Token Admin`, otherwise it returns `false`.

Methods for Role Management

`addRole`
This method adds a role to a specified user. This method can be called only by a `Token Admin`.

Parameters:

- `userAddress: address` – The wallet address of the user. The address cannot be zero.
- `role: string` – The name of the role to add to the specified user. For example, `minter`, `burner`, or `notary`.
- `scopeId: uint256` – The non-fungible token (NFT) class ID or the fungible token ID.

`removeRole`
This method removes a role from a specified user. This method can be called only by a `Token Admin` or `Org Admin` of the specified organization.

Parameters:

- `userAddress: address` – The wallet address of the user. The address cannot be zero.
- `role: string` – The name of the role to remove from the specified user. For example, `minter`, `burner`, or `notary`.
- `scopeId: uint256` – The non-fungible token (NFT) class ID or the fungible token ID.

`accountHasRole`
This method checks whether a user has a specified role. This method can be called by any user.

Parameters:

- `role: string` – The Keccak-256 hash of the name of the role (`minter`, `burner`, or `notary`) to search for.
- `userAddress: address` – The wallet address of the user. The address cannot be zero.
- `scopeId: uint256` – The non-fungible token (NFT) class ID or the fungible token ID.

Returns:

- The method returns `true` if the user has the specified role, otherwise it returns `false`.

`addTokenSysRole`
This method adds the `TOKEN_SYS_VAULT_ROLE` role to a specified user. This method can be called only by a `Token Admin`.

Parameters:

- `userAddress: address` – The wallet address of the user. The address cannot be zero.
- `role: string` – The only supported role is `TOKEN_SYS_VAULT_ROLE`.

Returns:

- The method returns `true` if the role was added successfully, otherwise it returns `false`.

`removeTokenSysRole`
This method removes the `TOKEN_SYS_VAULT_ROLE` role from a specified user. This method can be called only by a `Token Admin`.

Parameters:

- `userAddress: address` – The wallet address of the user. The address cannot be zero.
- `role: string` – The only supported role is `TOKEN_SYS_VAULT_ROLE`.

Returns:

- The method returns `true` if the role was removed successfully, otherwise it returns `false`.

`transferTokenSysRole`
This method transfers the `TOKEN_SYS_VAULT_ROLE` role from one user to another. This method can be called only by a `Token Admin`.

Parameters:

- `fromAddress: address` – The wallet address of the sender. The address cannot be zero.
- `toAddress: address` – The wallet address of the receiver. The address cannot be zero.
- `role: string` – The only supported role is `TOKEN_SYS_VAULT_ROLE`.

Returns:

- The method returns `true` if the role was tramsferred successfully, otherwise it returns `false`.

ERC-1155 Token Contract Methods

Methods for Token Configuration Management

`__ERC1155Token_init`
This method is called when the token contract is deployed. This method can be called only by a `Token Admin`.

Parameters:

- `accountContract: address` – The address of the account contract.
- `uri: string` – The token type URI.

`saveNFTClass`
This method saves NFT class information to the ledger. This method can be called only by a `Token Admin`.

Parameters:

- `classinfo: JSON` – The NFT account class information object. The following code shows an example for a whole NFT.

```
{
    "nftClassId": (BigInt(1)),
    "tokenName": "ArtCollection2",
    "tokenDesc": "this is art collection contract",
    "tokenStandard": "erc1155",
    "tokenUnit": 0,
    "behaviors": ["mintable", "burnable", "transferable", "roles",
"indivisible"],
    "divisible": { "decimals": 0 },
    "mintable": { "maxMintQuantity": 2 }
}
```

The following code shows an example for a fractional NFT.

```
{
  "nftClassId": (BigInt(0)),
  "tokenName": "ArtCollection",
  "tokenDesc": "this is art collection contract",
  "tokenStandard": "erc1155",
  "tokenUnit": 1,
  "behaviors": ["mintable", "burnable", "transferable", "roles",
"divisible"],
  "divisible": { "decimals": 0 },
  "mintable": { "maxMintQuantity": 1 }
}
```

`createNonFungibleToken`
This method mints NFTs. This method can be called only by a user with the minter role.

Parameters:

- `tokenId: uint256` – The token ID of the NFT, which contains NFT class information. The top 128 bits of the ID represent the NFT class ID and the bottom 128 bits represent the unique index of the NFT.
- `quantity: uint256` – The amount of NFTs to mint (create).

`createFungibleToken`
This method is called when the token contract is deployed. This method can be called only by a `Token Admin`.

Parameters:

- `token` – The fungible token definition to record in the ledger, as shown in the following example.

```
 "tokenId": BigInt(0),
 "tokenName": "wcbdc",
```

```
 "tokenDesc": "this is wcbdc token",
 "tokenStandard": "erc1155",
 "tokenType": 1,
 "tokenUnit": 1,
 "behaviors": ["mintable", "burnable", "transferable", "roles",
"divisible"],
 "divisible": { "decimals": 2 },
 "quantity": 1,
 "mintable": { "maxMintQuantity": 1 }
```

`getTokenById`
This method gets details for a token. This method can be called only by a `Token Admin` or token owner.

Parameters:

- `token` – The fungible token definition, as shown in the following example.

```
 "tokenId": BigInt(0),
 "tokenName": "wcbdc",
 "tokenDesc": "this is wcbdc token",
 "tokenStandard": "erc1155",
 "tokenType": 1,
 "tokenUnit": 1,
 "behaviors": ["mintable", "burnable", "transferable", "roles",
"divisible"],
 "divisible": { "decimals": 2 },
 "quantity": 1,
 "mintable": { "maxMintQuantity": 1 }
```

- `metaInfo: bytes` – If the token is fungible, specify an empty value. If the token is a whole NFT, specify an encoded version of the following `struct` object.

```
struct ERC1155WholeNFT {
    bool isBurned;
    bool isLocked;
    uint256 creationDate;
    uint256 quantity;
    address createdBy;
    address owner;
}
```

If the token is a fractional NFT, specify an encoded version of the following `struct` object.

```
struct ERC1155FractionalNFT {
    bool isBurned;
    bool isLocked;
    uint256 creationDate;
    uint256 quantity;
    address createdBy;
    Owners[] owners;
}
```

`geTokenDecimals`
This method gets the decimal value for a token. This method can be called only by a `Token Admin`.

Parameters:

- `id: uint256` – The token ID.

Returns

- A `uint8` value of the number of decimal places.

`tokenIdOf`
This method gets the ID of a token. This method can be called only by a `Token Admin`.

Parameters:

- `classId: uint256` – The NFT class ID.
- `serialId: uint256` – The serial ID in the NFT class.

Returns

- A `uint256` token ID.

`balanceOf`
This method gets the token balance for the specified user. This method can be called only by a `Token Admin` or by the token owner.

Parameters:

- `account: address` – The wallet address of the user. The address cannot be zero.
- `id: uint256` – The token ID.

Returns

- A `uint256` value of the token balance.

`balanceOfBatch`
This method gets the token balance for a list of users. This method can be called only by a `Token Admin` or by the token owner.

Parameters:

- `account: address[]` – A list of the wallet addresses of users.
- `id: uint256[]` – A list of token IDs.

Returns

- A `uint256[]` list of token balances.

`exists`
This method checks whether a specified token exists. This method can be called only by a `Token Admin`.

Parameters:

- `id: uint256` – A token ID.

Returns:

- The method returns `true` if the specified token exists, otherwise it returns `false`.

`totalSupply`
This method checks gets the total supply of all tokens in the contract. This method can be called only by a `Token Admin`.

Parameters:

- none

Returns:

- A `uint256[]` value of the total token supply.

`totalSupply`
This method checks gets the total supply of a specified token in the contract. This method can be called only by a `Token Admin`.

Parameters:

- `id: uint256` – A token ID.

Returns:

- A `uint256[]` value of the total supply of the specified token.

Token Behavior Management - Mintable Behavior

`mintBatch`
This method creates (mints) ERC-1155 tokens in batch mode. The tokens must be initialized. This method can be called by any user with the minter role.

Parameters:

- `tokenIds: uint256[]` – The list of tokens to mint.
- `quantity: uint256[]` – The list of token quantities to mint.
- `data: byte` – Additional bytes of data.

Token Behavior Management - Burnable Behavior

`burnBatch`
This method destroys (burns) ERC-1155 tokens in batch mode. The tokens must be initialized. This method can be called by any user with the burner role.

Parameters:

- `tokenIds: uint256[]` – The list of tokens to burn.
- `quantity: uint256[]` – The list of token quantities to burn.
- `data: byte` – Additional bytes of data.

`burnNFT`
This method destroys (burns) a non-fungible token. The tokens must be initialized. This method can be called by any user with the burner role.

Parameters:

- `tokenId: uint256` – The ID of the token to burn.

Token Behavior Management - Transferable Behavior

`safeTransferFrom`
This method transfers tokens from a sender to a receiver. This method can be called by any user who holds tokens.

Parameters:

- `from: address` – The wallet address of the sender. The address cannot be zero.
- `to: address` – The wallet address of the receiver. The address cannot be zero.
- `id: uint256` – The ID of the ERC-1155 token to transfer.
- `value: uint256` – The amount of tokens to transfer.
- `data: byte` – Additional bytes of data.

`safeBatchTransferFrom`
This method transfers tokens from a sender to a receiver. This method can be called by any user who holds tokens.

Parameters:

- `from: address` – The wallet address of the sender. The address cannot be zero.
- `to: address` – The wallet address of the receiver. The address cannot be zero.
- `id: uint256` – The ID of the ERC-1155 token to transfer.
- `value: uint256` – The amount of tokens to transfer.
- `data: byte` – Additional bytes of data.

Token Behavior Management - Delegable Behavior

`setApprovalForAll`
This method grants or revokes permission for an operator to transfer the caller's tokens, based on the `approved` parameter. This method can be called by any user who holds tokens.

Parameters:

- `operator: address` – The wallet address of the operator who is being granted or revoked permissions. The address cannot be zero.
- `approved: bool` – A Boolean flag indicated whether the operator has permission to transfer the caller's tokens.

`isApprovedForAll`
This method checks whether an operator is approved to transfer tokens for a specified account. This method can be called by any user who holds tokens.

Parameters:

- `account: address` – The wallet address of the users who granted or revoked permissions to the operator for transferring their tokens. The address cannot be zero.
- `operator: address` – The wallet address of the operator who was granted or revoked permissions. The address cannot be zero.

Returns:

- A Boolean value indicating whether the operator is approved to transfer tokens for the specified account.

Token Behavior Management - Pausable Behavior

`paused`
This method checks whether the contract is paused. This method can be called by any user.

Parameters:

- none

`pause`
This method pauses the contract. This method can be called only by a `Token Admin`.

Parameters:

- none

`unpause`
This method resumes a paused contract. This method can be called only by a `Token Admin`.

Parameters:

- none

Token Behavior Management - Lockable Behavior

`lockNFT`
This method locks a non-fungible token. This method can be called only by a user with the `TOKEN_SYS_VAULT_ROLE` role.

Parameters:

- `tokenId: uint256` – The ID of the token to lock.

`isNFTLocked`
This method checks whether a non-fungible token is locked. This method can be called only by a user with the `TOKEN_SYS_VAULT_ROLE` role or by the `Token Admin`.

Parameters:

- `tokenId: uint256` – The ID of the token to check.

Returns:

- A Boolean value indicating whether the token is locked.

10

Known Issues

Use the following information to troubleshoot problems with Oracle Blockchain Platform Enterprise Edition for Hyperledger Besu.

Key file must be in `.pem` format
Use the following Python commands to convert a private hexadecimal key to `.pem` format. In this example, the output file is `ethkey.pem`.

```
============================================================
python3 - <<'EOF'
from cryptography.hazmat.primitives.asymmetric import ec
from cryptography.hazmat.primitives import serialization
k=int("PASTE-PRIVATE-KEY-HERE",16)
pem=ec.derive_private_key(k,ec.SECP256K1()).private_bytes(
    serialization.Encoding.PEM,
    serialization.PrivateFormat.TraditionalOpenSSL,
    serialization.NoEncryption()
)
open("ethkey.pem","wb").write(pem)
EOF
============================================================
```

Incorrect Soulbound contract address
The contract address of the Soulbound smart contract on the Developer Tools page is incorrect. This contract is not deployed. You can download the source code and deploy it separately as needed.

`Object Object` in service console logs
The service console logs shown on the Logs page for the front-end container might include `Object Object` entries. You can safely ignore this behavior.

Rich history database not supported
The rich history database function shown in the user interface is inoperative and is not supported in this release.

A
Application Information

After creating a stack in Oracle Cloud Infrastructure, all information about the created stack is available in its Application Information tab. You can get to this page from the OCI Resource Manager.

Table A-1 General

Variable Name	Description	Default
Title	Clickable text that redirects the user to the marketplace listing.	Oracle Blockchain Platform on Oracle Cloud Infrastructure Container Engine for Kubernetes
Description	Simple text describing the stack.	The stack deploys a new OCI Kubernetes Engine Cluster and hosts the Blockchain Platform on it

Informational Text	Simple informational text to guide the user once they have run Apply on the stack.	After deployment, copy the `Domain Host Entry` and add it to your client machine's hosts file. For created instances, append the hosts entry with
		
		`console.<instance-name>.<instance-domain>`
		`rpcproxy.<instance-name>.<instance-domain>`
		
		Navigate to the Blockchain Platform Manager for creating instances.

Table A-2 Blockchain Platform Details

Variable Name	Description	Default
Ingress Gateway IP	IP address of the gateway created by the stack's OKE Cluster, used for DNS resolution.	
Domain Host Entry	Entry to append to your `hosts` file. • MacOS and Linux: `/etc/hosts` • Microsoft Windows: `C:\Windows\System32\drivers\etc\hosts`	`<ingress-gateway-ip> controlplane.<instance-domain> openldap.<instance-domain>` Example: `203.0.113.204 controlplane.example.com openldap.example.com auth.example.com`
Documentation	Documentation	

Table A-2 (Cont.) Blockchain Platform Details

Variable Name	Description	Default
Blockchain Platform Manager	URL to Blockchain Platform Manager on Kubernetes.	`https://controlplane.<instance-domain>/console/index.html` Example: `https://controlplane.example.com/console/index.html`

Table A-3 Kubernetes Cluster Details

Variable Name	Description	Default
OKE Cluster Name	OKE cluster name, with the dynamic deployment ID of the stack apply job appended	`<cluster-name>(<deployment-id>)` Example: `TestCluster01(tusf)`, `TestCluster02(Gd45)`
OKE Cluster Compartment OCID	Compartment OCID of the OKE cluster	Example: `ocid1.compartment.oc1..aaaaaaaa4wsjx5rugizog2i2bqlvd7ppcz7xmq4nybdsb5vgrwqsxigt2sia`
OKE Cluster OCID	OCID of the OKE cluster	Example: `ocid1.cluster.oc1.eu-milan-1.aaaaaaaa5tkfntba7ncdx774gl7yq4qvfhgwtncfy4cduglmlck7jkfm3y2q`
Kubernetes Configuration	The Kubernetes configuration of the cluster. Note: OCI CLI is necessary for the authentication for this configuration on the machine where it is used.	

Table A-4 Terraform Deployment Details

Variable Name	Variable Name	Description
Deployment Id	Randomly generated 4 character deployment ID, unique for each terraform job triggered.	Example: `tusf`, `Gd45`
Stack Version	Version of the stack.	
Deployed Region	Corresponding region where the stack and its resources have been deployed (created).	

Table A-5 Jump Host and Bastion Details

Variable Name	Description	Default
Bastion Name	Name of the bastion service used for accessing the jump host VM.	Example: `jumphost-bastion-43vg`

Table A-5 (Cont.) Jump Host and Bastion Details

Variable Name	Description	Default
Bastion OCID	OCID of the bastion service used for accessing the jump host VM.	Example: `ocid1.bastion.oc1.eu-milan-1.amaaaaaahwdhddqasib2ahia3djn2ceoeqtkzfu6euexxefxylcumxnvc7mq`
Jump Host Name	Name of the jump host VM instance.	Sample: `besu-jump-host-43vg`
Jump Host Public IP	Public IP to access the jump host VM instance.	Example: `203.0.113.178`

B

Connect to Oracle Kubernetes Engine

You can use the OCI command line interface (CLI) to access an Oracle Kubernetes Engine (OKE) cluster.

The following instructions assume that a `kubeconfig` file is already available.

1. Install the OCI CLI.
   - On Linux or macOS, enter the following command:

     ```
     bash -c "$(curl -L https://raw.githubusercontent.com/oracle/oci-cli/
     master/scripts/install/install.sh
     ```

   - On Microsoft Windows, use the `.msi` installer linked to from Installing the CLI.

   For more information about the OCI CLI, see Working with the CLI.
2. Install kubectl.
   - On macOS, enter the following homebrew command.

     ```
     brew install kubectl
     ```

   - On Linux, enter the following curl command:

     ```
     VERSION=(curl-L-shttps://dl.k8s.io/release/stable.txt)curl-LO"https://
     dl.k8s.io/release/{VERSION}/bin/linux/amd64/kubectl"; chmod +x kubectl
     && sudo mv kubectl /usr/local/bin/
     ```

   - On Microsoft Windows, use the following Chocolatey command.

     ```
     choco install kubernetes-cli
     ```

   For more information about installing kubectl, see Install Tools in the Kubernetes documentation.
3. Enter the following command to configure your OCI CLI credentials.

   ```
   oci setup config
   ```

   Provide the following information to the command.
   - The tenancy OCID
   - Your user OCID
   - The region (for example, `ap-mumbai-1`)
   - Your API key (fingerprint/private key path)

To get a new API key, enter the following command, and then select **Identity & Security**, and then **Users**, and then ***Your User Account***, and then **API Keys** to upload the public key to the console.

```
oci setup keys
```

4. Add the `kubeconfig` file to the default path, so that it is accessible by kubectl.
   - On macOS and Linux, add the `kubeconfig` file to the following directory.

     ```
     ~/.kube/config
     ```

   - On Microsoft Windows, add the `kubeconfig` file to the following directory.

     ```
     %USERPROFILE%.kube\config
     ```

5. Generate the kubeconfig file for the Kubernetes cluster.
   a. From the cluster page in OCI, select **Actions**, and then **Access cluster**. The Access cluster window is displayed.
   b. Select **Local access**.
   c. Select **Copy** for **Public endpoint access command**.
   d. Run the copied command to generate the `kubeconfig` file.
6. Run the following commands to validate connectivity to the cluster.

   ```
   kubectl cluster-info
   kubectl get nodes -o wide
   kubectl get ns
   ```

   Private endpoint clusters require network access to the virtual cloud network (VCN). Public endpoint clusters require internet connectivity to the API server endpoint.
7. If you use multiple OCI profiles, set the following environment variable.
   - Linux and macOS: `export OCI_CLI_PROFILE=`
   - Microsoft Windows: `$env:OCI_CLI_PROFILE=""`

If kubectl returns a 401 or 403 error, verify that your identity and access management (IAM) policies allow you to use and manage a cluster family in the compartment, and that your `kubeconfig` file was generated for the correct cluster and region.
To scale your OKE cluster, complete the following steps.

1. In the OCI console, select your OKE cluster and then select **Node pools**.
2. Select the node pool that you want to scale.
3. From the **Actions** menu, select **Edit**. The Edit node pool page is displayed.
4. Make changes as needed to the node count or node shape. To change the node shape (which affects the number of OCPUs and the amount of memory), select **Change shape**.
5. Select **Update** to confirm your changes.
6. From the **Actions** menu, select **Cycle nodes** to apply the updated configuration. The Cycle nodes page is displayed.

7. Under **Cycling options**, select **Replace nodes**. Enter numbers for **Maximum surge** and **Maximum unavailable**, then select **Cycle nodes**. When the state of the node pool is updated to `Active`, the scaling is complete.

Alternately, if you installed by deploying a stack, you can click **Destroy** to delete your current stack in the OCI console. and then select **Edit**, and then **Edit stack** to edit the stack definition to change the node count or node shape as needed. Select **Run apply** and **Save changes** to create a stack that uses the new parameters.