ORACLE

Cloud Infrastructure

User Guide

Pre-General Availability Draft: 3/15/2019

between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

Pre-GA Draft Documentation Notice

This documentation is in pre-General Availability status and is intended for demonstration and preliminary use only. It may not be specific to the hardware on which you are using the software. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to this documentation and will not be responsible for any loss, costs, or damages incurred due to the use of this documentation.

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.

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.

CONTENTS

CHAPTER 1 Overview of Functions

Oracle Functions is a fully managed, highly scalable, on-demand, Functions-as-a-Service platform, built on enterprise-grade Oracle Cloud Infrastructure and powered by the Fn Project open source engine. Use Oracle Functions (sometimes abbreviated to just Functions) when you want to focus on writing code to meet business needs. You don't have to worry about the underlying infrastructure because Oracle Functions will ensure your app is highly-available, scalable, secure, and monitored. With Oracle Functions, you can deploy your code, call it directly or in response to triggers, and get billed only for the resources consumed during the execution.

Oracle Functions is based on Fn Project. Fn Project is an open source, container native, serverless platform that can be run anywhere - any cloud or on-premises. Fn Project is easy to use, supports every programming language, and is extensible and performant. You can download and install the open source distribution of Fn Project, develop and test a function locally, and then use the same tooling to deploy that function to Oracle Functions.

You can access Oracle Functions using the Console, a CLI, and a REST API. You can invoke the functions you deploy to Oracle Functions using the CLI or by making signed HTTP requests.

Oracle Functions is integrated with Oracle Cloud Infrastructure Identity and Access Management (IAM), which provides easy authentication with native Oracle Cloud Infrastructure identity functionality.

Ways to Access Oracle Cloud Infrastructure

You can access Oracle Cloud Infrastructure using the Console (a browser-based interface) or the REST API. Instructions for the Console and API are included in topics throughout this guide. For a list of available SDKs, see SDKs and Other Tools.

To access the Console, you must use a supported browser. You can use the **Console** link at the top of this page to go to the sign-in page. You will be prompted to enter your cloud tenant, your user name, and your password.

To access the Console, you must use a supported browser. Oracle Cloud Infrastructure supports the latest desktop versions of Google Chrome, Microsoft Edge, Internet Explorer 11, Safari, Firefox, and Firefox ESR. Note that private browsing mode is not supported for Firefox, Internet Explorer, or Edge. Mobile browsers are not supported.

For general information about using the REST API, see About the API.

Resource Identifiers

Most types of Oracle Cloud Infrastructure resources have a unique, Oracle-assigned identifier called an Oracle Cloud ID (OCID). For information about the OCID format and other ways to identify your resources, see Resource Identifiers.

Authentication and Authorization

Each service in Oracle Cloud Infrastructure integrates with IAM for authentication and authorization, for all interfaces (the Console, SDK or CLI, and REST API).

An administrator in your organization needs to set up groups, compartments, and policies that control which users can access which services, which resources, and the type of access. For example, the policies control who can create new users, create and manage the cloud network, launch instances, create buckets, download objects, etc. For more information, see Get Started with Policies. For specific details about writing policies for each of the different services, see Policy Reference.

If you're a regular user (not an administrator) who needs to use the Oracle Cloud Infrastructure resources that your company owns, contact your administrator to set up a user ID for you. The administrator can confirm which compartment or compartments you should be using.

Oracle Functions Capabilities and Limits

In your tenancy you can create a maximum of:

- 10 applications (Limited Availability Release limit)
- 20 functions (Limited Availability Release limit)
- 50 triggers (Limited Availability Release limit)

See Service Limits for a list of applicable limits and instructions for requesting a limit increase.

Some other Oracle Functions capabilities and limits are also fixed. However, there are also a number that you can change. See Changing Oracle Functions Default Behavior.

Required IAM Service Policy

To use Oracle Cloud Infrastructure, you must be given the required type of access in a policy written by an administrator, whether you're using the Console or the REST API with an SDK, CLI, or other tool. If you try to perform an action and get a message that you don't have permission or are unauthorized, confirm with your administrator the type of access you've been granted and which compartment you should work in.

If you're new to policies, see Getting Started with Policies and Common Policies.

For more information about policies for Oracle Functions, see:

- Create Policies to Control Access to Function-Related Resources
- Details for Functions

Oracle Functions Concepts

This topic describes key concepts you need to understand when using Oracle Functions.

Functions Developers

Oracle Cloud Infrastructure users who use Oracle Functions to create and deploy functions are referred to as 'functions developers'. To use Oracle Functions, functions developers must have Oracle Cloud Infrastructure user accounts. Their accounts must belong to groups to which appropriate policies grant access to function-related resources.

See Create Groups and Users to use with Oracle Functions, if these don't exist already.

Applications

In Oracle Functions, an application is:

- a logical grouping of functions
- a common context to store configuration variables that are available to all functions

When you define an application in Oracle Functions, you specify the public subnets in which to run the functions in the application.

Oracle Functions shows applications and their functions in the Console.

See Creating Applications.

Functions

In Oracle Functions, functions are:

- small but powerful blocks of code that generally do one simple thing
- grouped into applications
- stored as Docker images in a specified Docker registry
- invoked in response to a CLI command or signed HTTP request

When you deploy a function to Oracle Functions using the Fn Project CLI, the function is built as a Docker image and pushed to a specified Docker registry.

A definition of the function is stored as metadata in the Oracle Functions server. The definition describes how the function is to be executed and includes:

- the Docker image to pull when the function is invoked
- the maximum length of time the function is allowed to execute for
- the name and source of any triggers associated with the function

When the function is invoked for the first time, Oracle Functions pulls the function's Docker image from the specified Docker registry, runs it as a Docker container, and executes the function. If there are subsequent requests to the same function, Oracle Functions directs those requests to the same running container. After a period being idle, the Docker container is stopped.

Oracle Functions shows functions, the applications into which they are grouped, and the triggers for each function in the Console.

See Creating, Deploying, and Invoking Your First Function.

Triggers (Not available in the Limited Availability Release)

A trigger is one way of invoking a function, in addition to the function being invoked directly using the Fn Project CLI.

A function need not be associated with any triggers, or it can be associated with one or multiple triggers.

There are three types of trigger:

- HTTP triggers. An HTTP trigger invokes a function in response to an HTTP request. You can specify any HTTP request method (GET, POST, PUT, etc) when using an HTTP trigger to invoke a function. (Not available in the Limited Availability Release.)

- Event-based triggers. An event-based trigger invokes a function in response to events occurring, for example, in other cloud services. (Not available in the Limited Availability Release.)
- Scheduled triggers. A scheduled trigger invokes a function on a defined, time-based schedule. (Not available in the Limited Availability Release.)

Invocations

In Oracle Functions, a function's code is run (or executed) when the function is called (or invoked). You can invoke a function that you've deployed to Oracle Functions in different ways:

- Using the Fn Project CLI.
- Using the Oracle Cloud Infrastructure SDKs.
- Making a signed HTTP request to the function's invoke endpoint. Every function has an invoke endpoint.

Oracle Functions shows information about function invocations in the Console. (Not available in the Limited Availability Release.)

See Invoking Functions.

How Oracle Functions Works

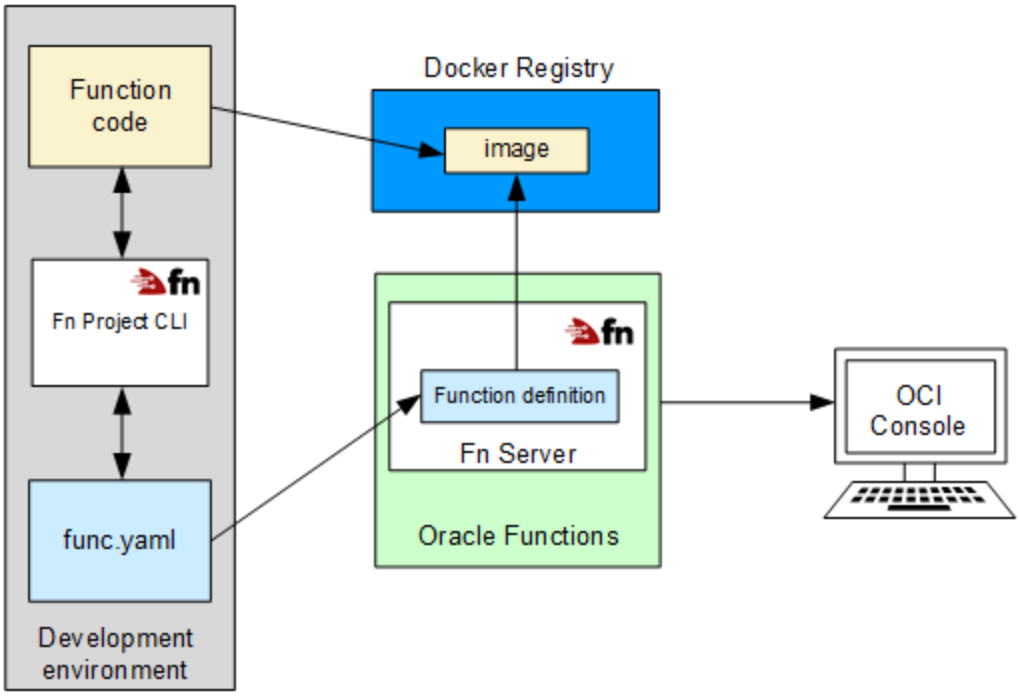
This topic describes how Oracle Functions works when you deploy a function, and when you invoke a function.

What Happens When You Deploy a Function to Oracle Functions?

When you have written the code for a function and it's ready to deploy, you can use a single Fn Project CLI command to perform all the deploy operations in sequence:

- building a Docker image from the function
- providing a definition of the function in a func.yaml file that includes:
    - the Docker image to pull when the function is invoked
    - the maximum length of time the function is allowed to execute for
- pushing the image to the specified Docker registry
- uploading function metadata (from the function definition in the func.yaml file) to the Fn Server
- adding the function to the list of functions shown in the Console

The above process of deploying a function to Oracle Functions is shown in the diagram.



What Happens When You Invoke a Function?

You can invoke a function that you've deployed to Oracle Functions:

- Using the Fn Project CLI.
- Using the Oracle Cloud Infrastructure SDKs.
- Making a signed HTTP request to the function's invoke endpoint. Every function has an invoke endpoint.
- Using a trigger associated with the function, fired from other Oracle Cloud services or from external services. (Not available in the Limited Availability Release.)

When a function is invoked for the first time, Oracle Functions first verifies the request with the IAM service. Assuming the request passes authentication and authorization checks, Oracle Functions then passes the request to the Fn Server, which uses the function definition to:

- identify the Docker image of the function to pull from the Docker registry
- execute the function by running the function's image as a container on an instance in a public subnet associated with the application to which the function belongs

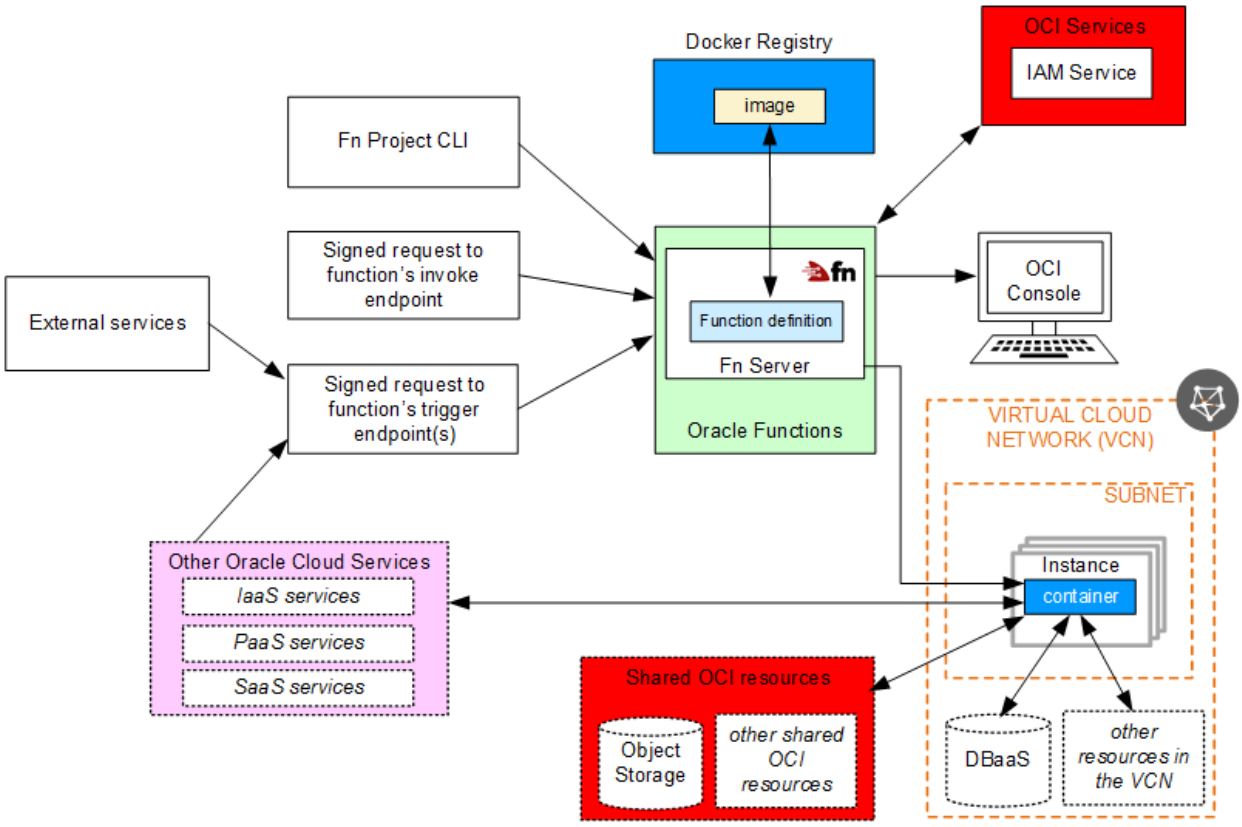
When the function is executing inside the container, the function can read from and write to other resources and services running in the same subnet (for example, Database as a Service). The function can also read from and write to other shared resources (for example, Object Storage), and other Oracle Cloud Services. You can specify the maximum length of time the function is allowed to execute by setting a timeout in the func.yaml file.

The function's execution log files are written to the Logging Service. (Not available in the Limited Availability Release.)

When the function has finished executing and after a period being idle, the Docker container is stopped. If Oracle Functions receives another call to the same function before the container is stopped, the second request is routed to the same running container. If Oracle Functions receives a call to a function that is currently executing inside a running container, a second Docker container is started.

Oracle Functions shows information about function invocations in the Console. (Not available in the Limited Availability Release.)

The above process of invoking a function is shown in the diagram.



Preparing for Oracle Functions

Before you can deploy functions to Oracle Functions:

- A tenancy administrator must have configured your tenancy for function development. This will give you access, via a suitable policy and user account, to an Oracle Cloud Infrastructure tenancy that has a VCN with at least one public subnet and an internet gateway. You must also have access to a Docker registry in which to store images (this documentation assumes you will be using Oracle Cloud Infrastructure Registry as your Docker registry and provides instructions accordingly). There are a number of different configuration tasks to complete. For more information, see Configuring Your Tenancy

for Function Development.

- You must have configured your client environment for functions development. There are a number of different configuration tasks to complete. For more information, see Configuring Your Client Environment for Function Development.

Configuring Your Tenancy for Function Development

Before you can start using Oracle Functions to create and deploy functions, you have to set up your tenancy for function development.

When a tenancy is created, an Administrators group is automatically created for the tenancy. Users that are members of the Administrators group can perform any operation on resources in the tenancy. Oracle Functions users are typically not members of the Administrators group, and do not have to be. However, a member of the Administrators group does need to perform a number of administrative tasks to enable users to use Oracle Functions.

To set up your tenancy for function development, you have to complete the following tasks in the order shown in this checklist (the instructions in the topics below assume that you are a tenancy administrator):

Task #	Tenancy Configuration Task	Done?
0	Limited Availability Release: Subscribe Tenancy to Phoenix Region, if not already subscribed	
1	Create Groups and Users to use with Oracle Functions, if these don't exist already	
2	Create a Compartment to Own Oracle Functions Resources in the Tenancy, if one doesn't exist already	

Task #	Tenancy Configuration Task	Done?
3	Create a VCN with Public Subnets to Use with Oracle Functions, if one doesn't exist already	
4	Create Policies to Control Access to Function-Related Resources, and more specifically: • Create a Policy in the Root Compartment to Give Oracle Functions Users Access to Oracle Cloud Infrastructure Registry Repositories • Create a Policy to Give Oracle Functions Users Access to Function-Related Resources and Virtual Network Resources • Create a Policy to Give the Oracle Functions Service Access to Virtual Network Resources • Create a Policy to Give the Oracle Functions Service Access to Repositories in Oracle Cloud Infrastructure Registry	

Click each of the links in turn, and follow the instructions.

When you have set up your tenancy for function development, the next step is to set up your client development environment (see Configuring Your Client Environment for Function Development).

Limited Availability Release: Subscribe Tenancy to Phoenix Region, if not already subscribed

Before users can start using the Limited Availability Release of Oracle Functions to create and deploy functions, your tenancy must be subscribed to the Phoenix region.

To confirm that your tenancy is subscribed to the Phoenix region:

1. Log in to the Console as a tenancy administrator.
2. Open the navigation menu. Under **Governance and Administration**, go to **Administration** and click **Tenancy Details**.
3. On the **Regions** page, confirm that your tenancy is already subscribed to the Phoenix region.
4. If your tenancy is not yet subscribed to the Phoenix region, click the **Subscribe To This Region** button beside us-phoenix-1.

Create Groups and Users to use with Oracle Functions, if these don't exist already

Before users can start using Oracle Functions to create and deploy functions, as a tenancy administrator you have to create Oracle Cloud Infrastructure user accounts, along with a group to which the user accounts belong. Later on, you'll define policies to give the group (and the user accounts that belong to it) access to function-related resources. If a suitable group and user accounts already exist, there's no need to create new ones.

To create groups and users to use with Oracle Functions:

1. Log in to the Console as a tenancy administrator.
2. If a suitable group for Oracle Functions users doesn't exist already, create such a group as follows:
    a. Open the navigation menu. Under **Governance and Administration**, go to **Identity** and click **Groups**. A list of the groups in your tenancy is displayed.
    b. Click **Create Group** and create a new group (see To create a group). Give the group a meaningful name (for example, `acme-functions-developers`) and description. Avoid entering confidential information.
3. If suitable user accounts for Oracle Functions users don't exist already, create users as follows:
    a. Open the navigation menu. Under **Governance and Administration**, go to **Identity** and click **Users**. A list of the users in your tenancy is displayed.
    b. Click **Create User** and create one or more new users (see To create a user).

4. If they haven't been added already, add users to the group to use Oracle Functions as follows:
    a. Open the navigation menu. Under **Governance and Administration**, go to **Identity** and click **Users**. A list of the users in your tenancy is displayed.
    b. Select one or more users and add them to the group authorized to use Oracle Functions (see To add a user to a group).

Create a Compartment to Own Oracle Functions Resources in the Tenancy, if one doesn't exist already

Before users can start using Oracle Functions to create and deploy functions, as a tenancy administrator you have to create an Oracle Cloud Infrastructure compartment to own function-related resources. If a suitable compartment already exists, there's no need to create a new one.

To create a compartment to own function-related resources in the tenancy:

1. Log in to the Console as a tenancy administrator.
2. Open the navigation menu. Under **Governance and Administration**, go to **Identity** and click **Compartments**. A list of the compartments in your tenancy is displayed.
3. Click **Create Compartment** and create a new compartment (see To create a compartment). Give the compartment a meaningful name (for example, `acme-functions-compartment`) and description. Avoid entering confidential information.

Create a VCN with Public Subnets to Use with Oracle Functions, if one doesn't exist already

Before users can start using Oracle Functions to create and deploy functions, as a tenancy administrator you have to create a VCN owned by the same compartment to which other function-related resources will belong.

Each subnet in the VCN must have a CIDR block that provides at least a certain minimum number of free IP addresses, as follows:

- AD-specific subnets must have a minimum of 12 free IP addresses
- regional subnets must have a minimum of 32 free IP addresses

Note that Oracle strongly recommends each subnet has a CIDR block that provides more than the minimum number of free IP addresses.

To support the largest possible number of concurrent connections, Oracle also strongly recommends that the security lists used by subnets in the VCN only have stateless rules.

If a suitable VCN already exists, there's no need to create a new one.

To create a VCN with public subnets to use with Oracle Functions:

1. Log in to the Console as a tenancy administrator.
2. Open the navigation menu. Under **Core Infrastructure**, go to **Networking** and click **Virtual Cloud Networks**.
3. Choose the compartment that will own function-related resources (on the left side of the page). The page updates to display only the resources in that compartment.
4. Click **Create Virtual Cloud Network** to create a new VCN.
5. In the **Create Virtual Cloud Network** dialog box, enter the following:
    a. **Name:** A meaningful name for the cloud network, such as `acme-functions-vcn`. The name doesn't have to be unique, but it cannot be changed later in the Console. Avoid entering confidential information.
    b. **Create Virtual Cloud Network Plus Related Resources:** For convenience, select this option to create the VCN and associated resourses (all with default properties) in a single operation.
    c. **Use DNS Hostnames in this VCN:** Select this option.
6. Click **Create Virtual Cloud Network** to create the VCN, along with the related resources (three public subnets and an internet gateway).

Create Policies to Control Access to Function-Related Resources

Before users can start using Oracle Functions to create and deploy functions, as a tenancy administrator you have to create a number of Oracle Cloud Infrastructure policies to grant

access to function-related and network resources. You have to:

- Create a Policy in the Root Compartment to Give Oracle Functions Users Access to Oracle Cloud Infrastructure Registry Repositories
- Create a Policy to Give Oracle Functions Users Access to Function-Related Resources and Virtual Network Resources
- Create a Policy to Give the Oracle Functions Service Access to Virtual Network Resources
- Create a Policy to Give the Oracle Functions Service Access to Repositories in Oracle Cloud Infrastructure Registry

See Details for Functions for more information about policies.

Create a Policy in the Root Compartment to Give Oracle Functions Users Access to Oracle Cloud Infrastructure Registry Repositories

To create a policy in the root compartment to give users access to repositories in Oracle Cloud Infrastructure Registry:

1. Log in to the Console as a tenancy administrator.
2. In the Console, open the navigation menu. Under **Governance and Administration**, go to **Identity** and click **Policies**. A list of the policies in the compartment you're viewing is displayed.
3. Select the tenancy's root compartment from the list on the left. Note that you must select the root compartment.
4. Click **Create Policy**.
5. Enter the following:
    - **Name:** A meaningful name for the policy (for example, `acme-functions-developers-ocir-access`). The name must be unique across all policies in your tenancy. You cannot change this later. Avoid entering confidential information.
    - **Description:** A meaningful description (for example, `Gives functions developers access to create repos and push/pull images`). You can change this later if you want to. Avoid entering confidential information.

- **Policy Versioning:** Select **Keep Policy Current** if you'd like the policy to stay current with any future changes to the service's definitions of verbs and resources. Or if you'd prefer to limit access according to the definitions that were current on a specific date, select **Use Version Date** and enter that date in YYYY-MM-DD format. For more information, see Policy Language Version.
- **Statement:** A policy statement to enable the group to access repositories in Oracle Cloud Infrastructure Registry. To access a repository, users must belong to a group that has been given permission to manage the repository, by specifying a statement such as:

  ```
  Allow group <group-name> to manage repos in tenancy
  ```

  where `<group-name>` is the name of the group to which users using Oracle Functions belong.
  For example:

  ```
  Allow group acme-functions-developers to manage repos in tenancy
  ```

  The above policy statement gives the group permission to manage all repositories in the tenancy. If you consider this to be too permissive, then you can restrict the repositories to which the group has access by including a where clause in the `manage repos` statement. Note that if you do include a where clause, you must also include a second statement in the policy to enable the group to inspect all repositories in the tenancy (when using the Console).
  For example, the following policy statements restrict the group to accessing only repositories with names that start 'acme-web-app', but also enables the group to inspect all repositories in the tenancy:

  ```
  Allow group acme-functions-developers to inspect repos in tenancy

  Allow group acme-functions-developers to manage repos in tenancy where all
  {target.repo.name=/acme-web-app*/ }
  ```

- **Tags:** Optionally, you can apply tags. If you have permissions to create a resource, you also have permissions to apply free-form tags to that resource. To apply a defined tag, you must have permissions to use the tag namespace. For

more information about tagging, see Resource Tags. If you are not sure if you should apply tags, skip this option (you can apply tags later) or ask your administrator.

6. Click **Create** to create the policy giving Oracle Functions users access to repositories in Oracle Cloud Infrastructure Registry.

CREATE A POLICY TO GIVE ORACLE FUNCTIONS USERS ACCESS TO FUNCTION-RELATED RESOURCES AND VIRTUAL NETWORK RESOURCES

To create a policy to give users access to function-related and virtual network resources belonging to the compartment that will own function-related resources:

1. Log in to the Console as a tenancy administrator.
2. In the Console, open the navigation menu. Under **Governance and Administration**, go to **Identity** and click **Policies**. A list of the policies in the compartment you're viewing is displayed.
3. Select the compartment that will own Oracle Functions resources from the list on the left.
4. Click **Create Policy**.
5. Enter the following:
   - **Name:** A meaningful name for the policy (for example, `acme-functions-developers-manage-access`). The name must be unique across all policies in your tenancy. You cannot change this later. Avoid entering confidential information.
   - **Description:** A meaningful description (for example, `Gives functions developers access to all resources in the acme-functions compartment`). You can change this later if you want to. Avoid entering confidential information.
   - **Policy Versioning:** Select **Keep Policy Current** if you'd like the policy to stay current with any future changes to the service's definitions of verbs and resources. Or if you'd prefer to limit access according to the definitions that were

current on a specific date, select **Use Version Date** and enter that date in YYYY-MM-DD format. For more information, see Policy Language Version.

- **Statement:** Enter the following policy statements to give the group access to the necessary resources in the compartment in which functions development will take place:
    - As **Statement 1:**, enter the following policy statement to give the group access to all function-related resources belonging to the compartment:

      ```
      Allow group <group-name> to manage functions-family in compartment <compartment-name>
      ```

      For example:

      ```
      Allow group acme-functions-developers to manage functions-family in compartment acme-functions-compartment
      ```

    - As **Statement 2:**, enter the following policy statement to give the group access to all VNICs belonging to the compartment:

      ```
      Allow group <group-name> to manage vnics in compartment <compartment-name>
      ```

      For example:

      ```
      Allow group acme-functions-developers to manage vnics in compartment acme-functions-compartment
      ```

    - As **Statement 3:**, enter the following policy statement to give the group access to all subnets belonging to the compartment:

      ```
      Allow group <group-name> to inspect subnets in compartment <compartment-name>
      ```

      For example:

      ```
      Allow group acme-functions-developers to inspect subnets in compartment acme-functions-compartment
      ```

- **Tags:** Optionally, you can apply tags. If you have permissions to create a resource, you also have permissions to apply free-form tags to that resource. To apply a defined tag, you must have permissions to use the tag namespace. For more information about tagging, see Resource Tags. If you are not sure if you

should apply tags, skip this option (you can apply tags later) or ask your administrator.

6. Click **Create** to create the policy giving Oracle Functions users access to function-related resources and virtual network resources in the compartment.

Create a Policy to Give the Oracle Functions Service Access to Virtual Network Resources

To create a policy to give the Oracle Functions service access to virtual network resources belonging to the compartment that will own function-related resources:

1. Log in to the Console as a tenancy administrator.
2. In the Console, open the navigation menu. Under **Governance and Administration**, go to **Identity** and click **Policies**. A list of the policies in the compartment you're viewing is displayed.
3. Select the compartment that will own Oracle Functions resources from the list on the left.
4. Click **Create Policy**.
5. Enter the following:
   - **Name:** A meaningful name for the policy (for example, `functions-service-network-access`). The name must be unique across all policies in your tenancy. You cannot change this later. Avoid entering confidential information.
   - **Description:** A meaningful description (for example, `Gives Oracle Functions access to virtual network resources in the acme-functions compartment`). You can change this later if you want to. Avoid entering confidential information.
   - **Policy Versioning:** Select **Keep Policy Current** if you'd like the policy to stay current with any future changes to the service's definitions of verbs and resources. Or if you'd prefer to limit access according to the definitions that were current on a specific date, select **Use Version Date** and enter that date in YYYY-MM-DD format. For more information, see Policy Language Version.

- **Statement:** The following policy statement to give the Oracle Functions service access to all virtual network resources in the compartment in which functions development will take place:

  ```
  Allow service FaaS to use virtual-network-family in compartment <compartment-name>
  ```

  For example:

  ```
  Allow service FaaS to use virtual-network-family in compartment acme-functions-compartment
  ```

- **Tags:** Optionally, you can apply tags. If you have permissions to create a resource, you also have permissions to apply free-form tags to that resource. To apply a defined tag, you must have permissions to use the tag namespace. For more information about tagging, see Resource Tags. If you are not sure if you should apply tags, skip this option (you can apply tags later) or ask your administrator.

6. Click **Create** to create the policy giving the Oracle Functions service access to virtual network resources in the compartment.

Create a Policy to Give the Oracle Functions Service Access to Repositories in Oracle Cloud Infrastructure Registry

To create a policy to give the Oracle Functions service access to repositories in Oracle Cloud Infrastructure Registry:

1. Log in to the Console as a tenancy administrator.
2. In the Console, open the navigation menu. Under **Governance and Administration**, go to **Identity** and click **Policies**. A list of the policies in the compartment you're viewing is displayed.
3. Select the tenancy's root compartment from the list on the left. Note that you must select the root compartment.
4. Click **Create Policy**.
5. Enter the following:
   - **Name:** A meaningful name for the policy (for example, `functions-service-repos-access`). The name must be unique across all policies in your tenancy. You

cannot change this later. Avoid entering confidential information.

- **Description:** A meaningful description (for example, `Gives Oracle Functions access to repositories in Oracle Cloud Infrastructure Registry`). You can change this later if you want to. Avoid entering confidential information.
- **Policy Versioning:** Select **Keep Policy Current** if you'd like the policy to stay current with any future changes to the service's definitions of verbs and resources. Or if you'd prefer to limit access according to the definitions that were current on a specific date, select **Use Version Date** and enter that date in YYYY-MM-DD format. For more information, see Policy Language Version.
- **Statement:** The following policy statement to give the Oracle Functions service access to all repositories in the tenancy:

  ```
  Allow service FaaS to read repos in tenancy
  ```

  The above policy statement gives the Oracle Functions service access to all repositories in the tenancy. If you consider this to be too permissive, then you can restrict the repositories to which Oracle Functions has access by including a where clause in the `read repos` statement.

  For example, the following policy statement restricts Oracle Functions to accessing only repositories with names that start 'acme-web-app':

  ```
  Allow service FaaS to read repos in tenancy where all {target.repo.name=/acme-web-app*/ }
  ```

- **Tags:** Optionally, you can apply tags. If you have permissions to create a resource, you also have permissions to apply free-form tags to that resource. To apply a defined tag, you must have permissions to use the tag namespace. For more information about tagging, see Resource Tags. If you are not sure if you should apply tags, skip this option (you can apply tags later) or ask your administrator.

6. Click **Create** to create the policy giving the Oracle Functions service access to repositories in Oracle Cloud Infrastructure Registry.

Configuring Your Client Environment for Function Development

Before you can start using Oracle Functions to create and deploy functions, you have to set up your client environment for function development. Note that prior to setting up your client environment, you must already have set up your tenancy (see Configuring Your Tenancy for Function Development).

To set up your client environment for function development, you have to complete the following tasks in the order shown in this checklist:

Task #	Development Environment Configuration Task	Done?
1	1. Set up an Oracle Cloud Infrastructure API Signing Key for Use with Oracle Functions	
2	2. Create a Profile in the Oracle Cloud Infrastructure CLI Configuration File	
3	3. Create and Configure a Copy of oci-curl	
4	4. Install Docker for Use with Oracle Functions	
5	5. Install the Fn Project CLI	
6	6. Create an Fn Project CLI Context to Connect to Oracle Cloud Infrastructure	
7	7. Set the Context for the Fn Project CLI Using the oracle.profile Parameter	
8	8. Generate an Auth Token to Enable Login to Oracle Cloud Infrastructure Registry	
9	9. Start Docker	
10	10. Log in to Oracle Cloud Infrastructure Registry	

Click each of the links in the checklist in turn, and follow the instructions.

When you have completed all of the development environment configuration tasks, confirm your configuration is correct (see Verifying Your Configuration for Function Development).

Oracle Functions Configuration Settings

When you configure your development environment for Oracle Functions, you have to specify values for a number of different configuration settings, as shown in this section.

URL TO CALL THE API

URL	Value
api-url	`https://functions.<region-name>.oraclecloud.com` where `<region-name>` is one of: • `us-ashburn-1` • `us-phoenix-1` • `eu-frankfurt-1` • `uk-london-1` **Example:** `https://functions.us-phoenix-1.oraclecloud.com` **Description:** The endpoint to use when calling the API.

SETTINGS FOR ORACLE CLOUD INFRASTRUCTURE RESOURCES

Setting	Format
Tenancy	`<tenancy-name>` or `<tenancy-ocid>` **Example:** • `acme-tenancy` • `ocid1.tenancy.oc1..aa______` **Description:** Depending on the situation, either the name or the OCID of the tenancy in which to create and deploy functions. See Where to Get the Tenancy's OCID and User's OCID.
Region	`<region-name>` where `<region-name>` is one of: • `us-ashburn-1` • `us-phoenix-1` • `eu-frankfurt-1` • `uk-london-1` **Example:** `us-phoenix-1` **Description:** Region in which to create and deploy functions.

Setting	Format
Compartment	`<compartment-name>` or `<compartment-ocid>` **Example:** • `acme-functions-compartment` • `ocid1.compartment.oc1..aa_____` **Description:** Depending on the situation, either the name or the OCID of the compartment in which to create and deploy functions. Your tenancy administrator might have created a compartment specifically for use with Oracle Functions.
VCN	`<vcn-name>`or `<vcn-ocid>` **Example:** • `acme-vcn-01` • `ocid1.vcn.oc1..aa_____` **Description:** Depending on the situation, either the name or the OCID of the VCN containing the subnets in which to run functions. Your tenancy administrator might have created a VCN specifically for use with Oracle Functions.

Setting	Format
Subnet	`<subnet-name>` or `<subnet-ocid>` **Example:** • `Public Subnet IHsY:US-PHOENIX-AD-1` • `ocid1.subnet.oc1..aa_____` **Description:** Depending on the situation, either the name or the OCID of the public subnet in which to run functions. Your tenancy administrator might have created a subnet specifically for use with Oracle Functions. If a regional subnet has been defined, best practice is to select that subnet to make failover across availability domains simpler to implement. If a regional subnet has not been defined and you need to meet high availability requirements, select multiple subnets.

SETTINGS FOR ORACLE CLOUD INFRASTRUCTURE IDENTITY

Setting	Format
User	`<user-name>` or `<user-ocid>` **Example:** • `john.doe` • `ocid1.user.oc1..aa_____` **Description:** Depending on the situation, either the name or the OCID of the Oracle Cloud Infrastructure user account you will be using to create and deploy functions. See Where to Get the Tenancy's OCID and User's OCID.
Private key file path	`<private-key-file-path>` **Example:** `/Users/johndoe/.oci/john_api_key_private.pem` **Description:** File name and location you specify when you create a private key.
Fingerprint	`<public-key-fingerprint>` **Example:** `d1:b2:32:53:d3:5f:cf:68:2d:6f:8b:5f:77:8f:07:13` **Description:** Value returned when you upload a public key to Oracle Cloud Infrastructure.

Setting	Format
Passphrase	`<passphrase>` **Description:** A string of characters you specify for additional protection when you create a private key.
Auth token	`<auth-token>` **Example:** `6<!)N_____6MqX` **Description:** An Oracle Cloud Infrastructure auth token value you obtain from the Console, for use when logging into a Docker registry.

SETTINGS FOR DOCKER

Setting	Format
Docker registry address	`<region-code>.ocir.io/<tenancy>/<repo-name>` where `<region-code>` is one of: • `fra` (Frankfurt) • `iad` (Ashburn) • `lhr` (London) • `phx` (Phoenix) **Example:** `phx.ocir.io/acme-dev/acme-repo` **Description:** The url to use when specifying the Docker registry as the Oracle Functions context.
Docker registry login url	`<region-code>.ocir.io` where `<region-code>` is one of: • `fra` (Frankfurt) • `iad` (Ashburn) • `lhr` (London) • `phx` (Phoenix) **Example:** `phx.ocir.io` **Description:** The url to use when logging into a Docker registry.

Setting	Format
Docker registry user	`<tenancy-name>/<username>` **Example:** `acmedev/jdoe@acme.com` **Description:** The name of an Oracle Cloud Infrastructure user who will be using Oracle Functions to create and deploy functions. If your tenancy is federated with Oracle Identity Cloud Service, use the format `<tenancy-name>/oracleidentitycloudservice/<username>`.
Docker registry password	`<auth-token>` **Example:** `6<!)N_____6MqX` **Description:** An Oracle Cloud Infrastructure auth token value you obtain from the Console, for use when logging into a Docker registry.

1. Set up an Oracle Cloud Infrastructure API Signing Key for Use with Oracle Functions

Before using Oracle Functions, you have to set up an Oracle Cloud Infrastructure API signing key.

The instructions in this topic assume:

- you are using Linux
- you are following Oracle's recommendation to provide a passphrase to encrypt the private key

For more information and other options, see Required Keys and OCIDs.

The instructions below describe how to create a new ~/.oci directory, how to generate a new private key file and public key file in that ~/.oci directory, how to upload the public key to Oracle Cloud Infrastructure to create a new API signing key, and how to obtain a fingerprint for the public API key. Be aware that instructions and examples elsewhere in this documentation assume the ~/.oci directory exists and contains the private and public key files.

If your user account already has an API signing key, create the ~/.oci directory if it doesn't exist, and then go straight to 2. Create a Profile in the Oracle Cloud Infrastructure CLI Configuration File.

If your user account doesn't already have an API signing key, follow the steps below, but note the following:

- If the ~/.oci directory doesn't exist, create it.
- If the ~/.oci directory already exists, go straight to the step below that instructs you to generate a private key file.
- If the ~/.oci directory already exists and already contains a private key file and public key file, and you know the passphrase that was used to encrypt the existing files, there's no need to create new private and public key files. Instead, go straight to the step below that instructs you to create a new API signing key and upload the public key value to Oracle Cloud Infrastructure to obtain a fingerprint.
- If you already have a private key file and public key file but they are not in the ~/.oci directory, and you know the passphrase that was used to encrypt the existing files, there's no need to create new private and public key files. Having created the ~/.oci directory if it doesn't exist, go straight to the step below that instructs you to create a new API signing key and upload the public key value to Oracle Cloud Infrastructure to obtain a fingerprint.

To set up an API signing key:

1. Log in to your development environment as a functions developer.
2. In a terminal window, confirm that the ~/.oci directory does not already exist. For example, by entering:

```
ls  ~/.oci
```

3. Assuming the ~/.oci directory does not already exist, create it. For example, by entering:

```
mkdir ~/.oci
```

4. Generate a private key encrypted with a passphrase that you provide by entering:

```
$ openssl genrsa -out ~/.oci/<private-key-file-name>.pem -aes128 2048
```

where <private-key-file-name> is a name of your choice for the private key file (for example, john_api_key_private.pem).
For example:

```
$ openssl genrsa -out ~/.oci/john_api_key_private.pem -aes128 2048

Generating RSA private key, 2048 bit long modulus
....+++
.....................................................................+++
e is 65537 (0x10001)

Enter pass phrase for /Users/johndoe/.oci/john_api_key_private.pem:
```

5. When prompted, enter a passphrase to encrypt the private key file. Be sure to make a note of the passphrase you enter, as you will need it later.
6. When prompted, re-enter the passphrase to confirm it.
7. Confirm that the private key file has been created in the directory you specified. For example, by entering:

```
$ ls -l ~/.oci/john_api_key_private.pem

-rw-r--r-- 1 johndoe staff 1766 Jul 14 00:24 /Users/johndoe/.oci/john_api_key_private.pem
```

8. Change permissions on the file to ensure that only you can read it. For example, by entering:

```
$ chmod go-rwx ~/.oci/john_api_key_private.pem
```

9. Generate a public key (encrypted with the same passphrase that you provided when creating the private key, and in the same location as the private key file) by entering:

```
$ openssl rsa -pubout -in ~/.oci/<private-key-file-name>.pem -out ~/.oci/<public-key-file-
name>.pem
```

where:

- <private-key-file-name> is what you specified earlier as the name of the private key file (for example, john_api_key_private.pem)
- <public-key-file-name> is a name of your choice for the public key file (for example, john_api_key_public.pem)

For example:

```
$ openssl rsa -pubout -in ~/.oci/john_api_key_private.pem -out ~/.oci/john_api_key_public.pem

Enter pass phrase for /Users/johndoe/.oci/john_api_key_private.pem:
```

10. When prompted, enter the same passphrase you previously entered to encrypt the private key file.
11. Confirm that the public key file has been created in the directory you specified. For example, by entering:

```
$ ls -l ~/.oci/

-rw------- 1 johndoe staff 1766 Jul 14 00:24 john_api_key_private.pem
-rw-r--r-- 1 johndoe staff 451 Jul 14 00:55 john_api_key_public.pem
```

12. Copy the contents of the public key file you just created. For example, by entering:

```
$ cat ~/.oci/john_api_key_public.pem | pbcopy
```

13. Having created the API key pair, upload the public key value to Oracle Cloud Infrastructure:
    a. Log in to the Console as the Oracle Cloud Infrastructure user who will be using Oracle Functions to create and deploy functions.
    b. In the top-right corner of the Console, open the **User** menu () and then click **User Settings** to view the details.
    c. On the **API Keys** page, click **Add Public Key**.
    d. Paste the public key's value into the window and click **Add**.

The key is uploaded and its fingerprint is displayed (for example, d1:b2:32:53:d3:5f:cf:68:2d:6f:8b:5f:77:8f:07:13).

e. (Optional) Note the fingerprint value. You'll use the fingerprint in a subsequent configuration task, so you might want to copy it to a convenient and secure location.

When you have completed the steps in this topic, go on to 2. Create a Profile in the Oracle Cloud Infrastructure CLI Configuration File.

2. Create a Profile in the Oracle Cloud Infrastructure CLI Configuration File

Before using Oracle Functions, you must have an Oracle Cloud Infrastructure CLI configuration file that contains the credentials of the user account that you will be using to create and deploy functions. These user account credentials are referred to as a 'profile'.

By default, the Oracle Cloud Infrastructure CLI configuration file is located at `~/.oci/config`. You might already have a configuration file as a result of installing the Oracle Cloud Infrastructure CLI. However, you don't need to have installed the Oracle Cloud Infrastructure CLI in order to use Oracle Functions.

The Oracle Cloud Infrastructure CLI configuration file can contain several profiles. If you already have a configuration file containing one or more profiles, you have to add a new profile to the existing file for the Oracle Cloud Infrastructure user who will be using Oracle Functions to create and deploy functions

The instructions in this topic assume:

- you are using Linux
- you have already completed the steps in 1. Set up an Oracle Cloud Infrastructure API Signing Key for Use with Oracle Functions

To create a profile in the Oracle Cloud Infrastructure CLI configuration file for the user account that you will be using to create and deploy functions:

1. Log in to your development environment as a functions developer.
2. In a terminal window, confirm the contents of the ~/.oci directory. For example, by

entering:

```
$ ls -l ~/.oci/

-rw------- 1 johndoe staff 1766 Jul 14 00:24 john_api_key_private.pem
-rw-r--r-- 1 johndoe staff 451 Jul 14 00:55 john_api_key_public.pem
```

3. Do one of the following, depending on whether the ~/.oci directory already contains a file called config:
    - If the ~/.oci directory already contains a file called config, open the file in a text editor.
    - If the ~/.oci directory doesn't yet contain a file called config, create the file and open it in a text editor. For example, by entering:

      ```
      $ vim ~/.oci/config
      ```

4. Add a new profile to the ~/.oci/config file as follows:

    ```
    [<profile-name>]
    user=<user-ocid>
    fingerprint=<public-key-fingerprint>
    key_file=<private-key-pem-file>
    tenancy=<tenancy-ocid>
    region=<region-name>
    pass_phrase=<passphrase>
    ```

    where:

    - `<profile-name>` is a name of your choosing for the profile.
    - `<user-ocid>` is the OCID of the Oracle Cloud Infrastructure user account you will be using to create and deploy functions. See Where to Get the Tenancy's OCID and User's OCID.
    - `<public-key-fingerprint>` is the fingerprint of the public API key value that you uploaded in the Console in 1. Set up an Oracle Cloud Infrastructure API Signing Key for Use with Oracle Functions .
    - `<private-key-pem-file>` is the full path to the private key file that you created in 1. Set up an Oracle Cloud Infrastructure API Signing Key for Use with Oracle Functions .

- `<tenancy-ocid>` is the OCID of the tenancy in which you will be creating and deploying functions. See Where to Get the Tenancy's OCID and User's OCID.
- `<region-name>` is the name of the Oracle Cloud Infrastructure region in which you will be creating and deploying functions, as follows:
  - `us-ashburn-1` (Not available in the Limited Availability release)
  - `us-phoenix-1`
  - `eu-frankfurt-1` (Not available in the Limited Availability release)
  - `uk-london-1` (Not available in the Limited Availability release)
- `<passphrase>` is the passphrase you entered in 1. Set up an Oracle Cloud Infrastructure API Signing Key for Use with Oracle Functions).

For example:

```
[john-oci-profile]
user=ocid1.user.oc1..aaaaaaaas...7ap
fingerprint=d1:b2:32:53:d3:5f:cf:68:2d:6f:8b:5f:77:8f:07:13
key_file=~/.oci/john_api_key_private.pem
tenancy=ocid1.tenancy.oc1..aaaaaaaap...keq
region=us-phoenix-1
pass_phrase=<your-passphrase>
```

When you have completed the steps in this topic, go on to 3. Create and Configure a Copy of oci-curl.

3. Create and Configure a Copy of oci-curl

You can use a bash script provided by Oracle (commonly referred to as oci-curl) to invoke a function. The oci-curl script creates a signed request, based on credentials you provide in the body of the script. For more information about oci-curl, see https://docs.cloud.oracle.com/iaas/Content/API/Concepts/signingrequests.htm#Bash

To use oci-curl to invoke a function, you must provide the credentials of an Oracle Cloud Infrastructure user that has been granted access to resources in the same tenancy and belonging to the same compartment as the function (see Create Policies to Control Access to Function-Related Resources).

Typically you'll want to invoke a function as the functions developer that's configured for your development environment. The instructions below assume that is the case.

The instructions in this topic assume you have already completed the steps in 2. Create a Profile in the Oracle Cloud Infrastructure CLI Configuration File.

To create and configure a copy of oci-curl:

1. Log in to your development environment as a functions developer.
2. Create a copy of the oci-curl script file in your development environment and add your credentials to the file as follows:
    a. In a browser navigate to https://docs.cloud.oracle.com/iaas/Content/Resources/Assets/signing_sample_bash.txt to see the oci-curl code as raw text.
    b. Select all the text and copy it.
    c. In a text editor, open a new file in a convenient location. For example, in a terminal window, you might create a new subdirectory in your home directory and open a new file in that directory by entering:

    ```
    $ cd ~

    $ mkdir oci-curl

    $ vim ~/oci-curl/oci-curl.sh
    ```

    The name and location of the new file is up to you, but the following instructions assume ~/oci-curl/oci-curl.sh.
    d. Paste the oci-curl script code that you copied earlier into the new file.
    e. Save the file but leave it open so you can add your credentials.
3. Replace the sample credentials in the oci-curl.sh file with those of the user account that you want to invoke functions, as follows:
    a. Locate the following lines in the oci-curl.sh file that contain sample credential values:

```
# TODO: update these values to your own
local tenancyId="ocid1.tenancy.oc1..aaaaaaaab______dsq";
local authUserId="ocid1.user.oc1..aaaaaaaas______o3r";
local keyFingerprint="20:3b:97:13:55:1c:5b:0d:d3:37:d8:50:4e:c5:3a:34";
local privateKeyPath="/Users/someuser/.oci/oci_api_key.pem";
```

Tip

Typically you'll want to invoke a function as the functions developer that's configured for your development environment. If that's the case, open the ~/.oci/config file so you can easily copy values from there to replace the sample values in the oci-curl.sh file.

b. Change the value of the `tenancyId` parameter in the oci-curl.sh file by replacing the sample value in quotes with the OCID of the tenancy in which the function has been deployed. For example:

```
local tenancyID="ocid1.tenancy.oc1..aaaaaaaap______keq";
```

c. Change the value of the `authUserID` parameter in the oci-curl.sh file by replacing the sample value in quotes with the OCID of the user account that you want to run the function. The user account must have access to resources in the same tenancy and belonging to the same compartment as the function. For example:

```
local authUserId="ocid1.user.oc1..aaaaaaaas______7ap";
```

d. Change the value of the `keyFingerprint` parameter in the oci-curl.sh file by replacing the sample value in quotes with the fingerprint of the user's public key uploaded to Oracle Cloud Infrastructure. For example:

```
local keyFingerprint="d1:b2:32:53:d3:5f:cf:68:2d:6f:8b:5f:77:8f:07:";
```

e. Change the value of the `privateKeyPath` parameter in the oci-curl.sh file by replacing the sample value in quotes with the full path to the private key file that is paired with the public key for which you provided the fingerprint. For example:

```
local privateKeyPath="/Users/johndoe/.oci/john_api_key_private.pem";
```

4. Save and close the oci-curl.sh file.

Note

When you've deployed functions to Oracle Functions and you want to use oci-curl to invoke them, you will first have to run the `source` command to set up the current shell environment for oci-curl. See Invoking Functions.

When you have completed the steps in this topic, go on to 4. Install Docker for Use with Oracle Functions.

4. Install Docker for Use with Oracle Functions

Before using Oracle Functions, a version of Docker supported by Fn Project must be installed in your development environment. If Docker is not already installed, or the installed version of Docker is not supported, you'll have to install or upgrade Docker.

The instructions in this topic assume you have already completed the steps in 3. Create and Configure a Copy of oci-curl.

To confirm that a supported version of Docker is installed in your development environment:

1. Log in to your development environment as a functions developer.
2. In a terminal window, confirm that Docker is installed by entering:

```
$ docker version
```

3. Do one of the following, depending on the message you see:
   - If you see an error message indicating that Docker is not installed, you have to install Docker before proceeding to the next step. See the Docker documentation for information about installing Docker on your platform. If your platform is Oracle Linux, see Oracle Container Runtime for Docker User's Guide.

- If you see a message indicating the version of Docker that's installed, go to the next step.

4. Assuming Docker is installed, go to the Fn Project home page on GitHub to confirm that the installed version of Docker is at least the minimum version specified in the Pre-requisites section.
   If the installed version of Docker is not supported by Fn Project, you have to upgrade the version of Docker before proceeding. See the Docker documentation for information about upgrading Docker on your platform. If your platform is Oracle Linux, see Oracle Container Runtime for Docker User's Guide.

When you have completed the steps in this topic, go on to 5. Install the Fn Project CLI.

5. Install the Fn Project CLI

Before using Oracle Functions, the Fn Project CLI must be installed in your development environment.

You can install the Fn Project CLI in a number of different ways according to your environment.

The instructions in this topic assume you have already completed the steps in 4. Install Docker for Use with Oracle Functions.

To install the Fn Project CLI:

1. Log in to your development environment as a functions developer.
2. Open the README.md file in the fnproject/cli repository on GitHub and follow the appropriate instructions for installing the Fn Project CLI in your development environment. As a convenient overview, the instructions are summarized below:
   - In a MacOS environment using Homebrew, install the Fn Project CLI by entering:

     ```
     $ brew install fn
     ```

   - In a Linux or MacOS environment, install the Fn Project CLI by entering:

     ```
     $ curl -LSs https://raw.githubusercontent.com/fnproject/cli/master/install | sh
     ```

If prompted for a password, enter the superuser's password.

- In a Linux, MacOS, or Windows environment, install the Fn Project CLI by downloading the binary from the Releases page and running it.

3. In a terminal window, confirm that the CLI has been installed by entering:

```
$ fn version
```

Assuming the Fn Project CLI has been installed correctly, you'll see a message indicating the version of the CLI that has been installed.

When you have completed the steps in this topic, go on to 6. Create an Fn Project CLI Context to Connect to Oracle Cloud Infrastructure.

6. Create an Fn Project CLI Context to Connect to Oracle Cloud Infrastructure

Before using Oracle Functions, you have to configure the Fn Project CLI to connect to your Oracle Cloud Infrastructure tenancy.

When the Fn Project CLI is initially installed, it's configured for a local development 'context'. To configure Fn Project CLI to connect to your Oracle Cloud Infrastructure tenancy instead, you have to create a new context. The context specifies Oracle Functions endpoints, the OCID of the compartment to which deployed functions will belong, and the address of the Docker registry to and from which to push and pull images.

You can define multiple contexts, each stored in a different context file in .yaml format. By default, the individual context files are stored in the `~/.fn/contexts` directory. The `~/.fn/config.yaml` file specifies which context file Fn Project uses.

To create a new context, you can create a new context file manually and edit the `~/.fn/config.yaml` file by hand to point to that file. Alternatively, you can use the Fn Project CLI to interactively create the new context file and instruct the Fn Project CLI to start using that file, as described below.

The instructions in this topic assume you have already completed the steps in 5. Install the Fn Project CLI.

To create a new context file using the Fn Project CLI:

1. Log in to your development environment as a functions developer.
2. In a terminal window, create the new Fn Project CLI context for Oracle Cloud Infrastructure by entering:

   ```
   $ fn create context <my-context> --provider oracle
   ```

   where <my-context> is a name of your choosing. For example:

   ```
   $ fn create context johns-oci-context --provider oracle
   ```

3. Specify that the Fn Project CLI is to use the new context by entering:

   ```
   $ fn use context <my-context>
   ```

   where <my-context> is the name you specified in the previous step. For example:

   ```
   $ fn use context johns-oci-context
   ```

4. Configure the new context with the OCID of the compartment you want to own deployed functions (you might have created a new compartment specifically for this purpose, see Create a Compartment to Own Oracle Functions Resources in the Tenancy, if one doesn't exist already) by entering:

   ```
   $ fn update context oracle.compartment-id <compartment-ocid>
   ```

   For example:

   ```
   $ fn update context oracle.compartment-id ocid1.compartment.oc1..aaaaaaaarvdfa72n...
   ```

5. Configure the new context with the api-url endpoint to use when calling the API by entering:

   ```
   $ fn update context api-url <api-endpoint>
   ```

   where:

   - `<api-endpoint>` is in the format `https://functions.<region-name>.oraclecloud.com`
   - `<region-name>` in `<api-endpoint>` is the name of the Oracle Cloud Infrastructure region in which you'll be creating and deploying functions, as follows:

- us-ashburn-1 (Not available in the Limited Availability release)
- us-phoenix-1
- eu-frankfurt-1 (Not available in the Limited Availability release)
- uk-london-1 (Not available in the Limited Availability release)

For example:

```
$ fn update context api-url https://functions.us-phoenix-1.oraclecloud.com
```

6. Configure the new context with the address of the Docker registry that you want to use with Oracle Functions by entering:

```
$ fn update context registry <region-code>.ocir.io/<tenancy-name>/<repo-name>
```

where:

- <region-code> is the Oracle Cloud Infrastructure Registry region, as follows:

Region Code	Region Name
fra	Frankfurt
iad	Ashburn
lhr	London
phx	Phoenix

  Oracle recommends that the Docker registry you specify is in the same region as the public subnet on which you intend functions to run.
- <tenancy-name> is the name of the tenancy in which to create repositories
- <repo-name> is a repository name to pre-pend to the names of functions that you deploy

For example:

```
$ fn update context registry phx.ocir.io/acme-dev/acme-repo
```

7. (Optional) Verify the Fn Project CLI context you've created by viewing the context file.

For example, by entering:

```
$ more ~/.fn/contexts/johns-oci-context.yaml

api-url: https://functions.us-phoenix-1.oraclecloud.com
provider: oracle
registry: phx.ocir.io/acme-dev/acme-repo
```

When you have completed the steps in this topic, go on to 7. Set the Context for the Fn Project CLI Using the oracle.profile Parameter.

7. Set the Context for the Fn Project CLI Using the oracle.profile Parameter

Before using Oracle Functions, you have to configure the Fn Project CLI to use the new profile you added to the Oracle Cloud Infrastructure CLI configuration file `~/.oci/config` (see 2. Create a Profile in the Oracle Cloud Infrastructure CLI Configuration File). The profile you added contains the credentials of the user account you'll be using to create and deploy functions.

Note that unless you specify otherwise, the Fn Project CLI will attempt to use a profile in the `~/.oci/config` file named `default`.

The instructions in this topic assume you have already completed the steps in 6. Create an Fn Project CLI Context to Connect to Oracle Cloud Infrastructure.

To configure the Fn Project CLI to use the profile you've created for use with Oracle Functions:

1. Log in to your development environment as a functions developer.
2. In a terminal window, configure the Fn Project CLI context with the name of the profile you've created for use with Oracle Functions by entering:

   ```
   $ fn update context oracle.profile <profile-name>
   ```

   For example:

   ```
   $ fn update context oracle.profile john-oci-profile
   ```

When you have completed the steps in this topic, go on to 8. Generate an Auth Token to Enable Login to Oracle Cloud Infrastructure Registry.

8. Generate an Auth Token to Enable Login to Oracle Cloud Infrastructure Registry

Before using Oracle Functions, the user account you'll be using to create and deploy functions must have an Oracle Cloud Infrastructure auth token. You use the auth token as the password when logging Docker in to Oracle Cloud Infrastructure Registry

The instructions in this topic assume you have already completed the steps in 7. Set the Context for the Fn Project CLI Using the oracle.profile Parameter.

If the user account already has an auth token, go straight on to 9. Start Docker. Otherwise, if the user account does not have an auth token, generate an auth token now.

To generate an auth token for the user account you'll be using to create and deploy functions:

1. Log in to the Console as a functions developer.
2. In the top-right corner of the Console, open the **User** menu () and then click **User Settings** to view the details.
3. On the **Auth Tokens** page, click **Generate Token**.
4. In the **Generate Token** dialog:
    a. Enter a meaningful description for the auth token. For example, `John's auth token for use with Oracle Functions`. Avoid entering confidential information.
    b. Click **Generate Token**. The new auth token is displayed. For example, `6<!)N___ _____6MqX`.
5. Copy the auth token immediately to a secure location from where you can retrieve it later, because you won't see the auth token again in the Console.
6. Close the **Generate Token** dialog.

When you have completed the steps in this topic, go on to 9. Start Docker.

9. Start Docker

Before using Oracle Functions, Docker must be running in your development environment. If it is not running, you must start Docker before proceeding.

The instructions in this topic assume you have already completed the steps in 8. Generate an Auth Token to Enable Login to Oracle Cloud Infrastructure Registry.

To verify that Docker is running:

1. Log in to your development environment as a functions developer.
2. In a terminal window, launch the standard hello-world Docker image as a container to confirm that Docker is running by entering:

   ```
   $ docker run hello-world
   ```

3. Do one of the following, depending on the message you see:
   - If you see an error message indicating that Docker is not running, you have to start the Docker daemon before proceeding. See the Docker documentation for information about starting Docker on your platform.
   - If you see a message like the one shown below, Docker is already running and you can proceed:

     ```
     Hello from Docker.

     This message shows that your installation appears to be working correctly.
     ```

When you have completed the steps in this topic, go on to 10. Log in to Oracle Cloud Infrastructure Registry .

10. Log in to Oracle Cloud Infrastructure Registry

Before using Oracle Functions, you have to log Docker in to the Docker registry in which you are going to store your functions as Docker images. This is the Docker registry specified in the Fn Project CLI context (see 6. Create an Fn Project CLI Context to Connect to Oracle Cloud Infrastructure).

You can store functions in public and private repositories in Oracle Cloud Infrastructure Registry, an Oracle-managed registry built on top of Oracle Cloud Infrastructure.

When you log Docker into a Docker registry, you have to provide the appropriate authentication details. For example, in the case of Oracle Cloud Infrastructure Registry, you have to provide the tenancy name, the user name, and the user's auth token.

The instructions in this topic assume you have already completed the steps in 9. Start Docker.

To log Docker into Oracle Cloud Infrastructure Registry:

1. Log in to your development environment as a functions developer.
2. In a terminal window, log in to Oracle Cloud Infrastructure Registry by entering:

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

   where `<region-code>` is the code for the Oracle Cloud Infrastructure Registry region specified in the Fn Project CLI context (see 6. Create an Fn Project CLI Context to Connect to Oracle Cloud Infrastructure), as follows:

Region Code	Region Name
`fra`	Frankfurt
`iad`	Ashburn
`lhr`	London
`phx`	Phoenix

   For example, `docker login phx.ocir.io`

3. When prompted, enter the name of the user you will be using with Oracle Functions to create and deploy functions, in the format `<tenancy-name>/<username>`. For example, `acme-dev/jdoe@acme.com`. If your tenancy is federated with Oracle Identity Cloud Service, use the format `<tenancy-name>/oracleidentitycloudservice/<username>`. You must have already generated an Oracle Cloud Infrastructure auth token for the user you specify (see 8. Generate an Auth Token to Enable Login to Oracle Cloud Infrastructure Registry).
4. When prompted for a password, enter the user's Oracle Cloud Infrastructure auth token. Having entered the password, Docker might warn you that the password is stored unencrypted in the Docker configuration file. The warning includes a link to the Docker documentation where you can find out how to configure a credential helper. Oracle recommends you review the information in the Docker documentation and consider using an external credentials store for increased security.

When you have completed the steps in this topic, you have completed the configuration tasks for your client environment. Go on to Verifying Your Configuration for Function Development to confirm that the Fn Project CLI can communicate with the API endpoint.

Verifying Your Configuration for Function Development

Before using Oracle Functions, it's a good idea to confirm that you have successfully completed:

- the tasks for Configuring Your Tenancy for Function Development
- the tasks for Configuring Your Client Environment for Function Development

If you have successfully completed the configuration tasks, the Fn Project CLI will be able to communicate with the API endpoint.

To confirm that the Fn Project CLI can communicate with the API endpoint:

1. Log in to your development environment as a functions developer.
2. In a terminal window, try and view a list of applications that have been defined in Oracle Functions by entering:

```
$ fn list apps
```

3. If you see either of the following, you can proceed to create and deploy functions because your system is configured correctly:
    - A message indicating that no applications have been found, which is expected if this is the first time the tenancy has been configured for Oracle Functions.
    - A list of applications that have already been created, which is expected if other users are already using the tenancy for functions development.
4. If you see an error message, it's likely that the Fn Project CLI cannot communicate with the API endpoint due to some incorrect configuration. Review the configuration tasks to confirm you completed them as instructed.

Creating, Deploying, and Invoking Your First Function

You can start off using Oracle Functions by using Fn Project CLI commands to:

- create a simple helloworld function written in java
- push the image to the Docker registry that's configured for Oracle Functions
- deploy the function to an application in Oracle Functions
- invoke the function

To get started with Oracle Functions:

1. Confirm that you have completed the prerequisite steps for using Oracle Functions, as described in Preparing for Oracle Functions. Specifically, that you have:
    - set up your tenancy (see Configuring Your Tenancy for Function Development)
    - set up your development environment (see Configuring Your Client Environment for Function Development)
2. Log in to the Console as a functions developer. If you're using the Limited Availability Release, go to https://console.us-phoenix-1.oraclecloud.com.
3. Use the Console to create a new application in Oracle Functions:

a. In the Console, open the navigation menu. Under **Solutions, Platform and Edge**, go to **Developer Services** and click **Functions**. If you're using the Limited Availability Release, go to the Oracle Functions landing page at https://console.us-phoenix-1.oraclecloud.com/functions.

b. Select the region you intend to use for Oracle Functions. Oracle recommends that you use the same region as the Docker registry specified in the Fn Project CLI context (see Configuring Your Client Environment for Function Development).

c. Select the compartment specified in the Fn Project CLI context (see Configuring Your Client Environment for Function Development).
The **Applications** page shows the applications already defined in the compartment.

d. Click **Create Application** and specify:

- The name for the new application as helloworld-app.
- The VCN and public subnet (or subnets) in which to run the function. For example, a VCN called acme-vcn-01 and a public subnet called Public Subnet IHsY:US-PHOENIX-AD-1). If a regional subnet has been defined, best practice is to select that subnet to make failover across availability domains simpler to implement. If a regional subnet has not been defined and you need to meet high availability requirements, select multiple subnets. Oracle recommends that public subnets are in the same region as the Docker registry specified in the Fn Project CLI context (see Configuring Your Client Environment for Function Development).

e. Click **Create**.

4. Log in to your development environment as a functions developer.
5. In a terminal window, create a helloworld java function by entering:

```
$ fn init --runtime java helloworld-func
```

A directory called helloworld-func is created, containing:

- A function definition file called func.yaml, containing the minimum amount of information required to build and run the function. See the Fn Project

documentation to find out about the additional parameters you can include in a func.yaml file.

- A /src directory containing source files and directories for the helloworld function (including /src/main/java/com/example/fn/HelloFunction.java).
- A Maven configuration file called pom.xml that specifies the project artifacts and dependencies required to compile the function from the source files.

6. Change directory to the newly created helloworld-func directory.
7. Enter the following single Fn Project command to build the function and its dependencies as a Docker image called helloworld-func, push the image to the specified Docker registry, and deploy the function to Oracle Functions:

```
$ fn deploy --app helloworld-app
```

8. (Optional) Assuming the specified Docker registry is Oracle Cloud Infrastructure Registry, use the Console to confirm that the helloworld-func image has been pushed to Oracle Cloud Infrastructure Registry successfully:
   a. In the Console, open the navigation menu. Under **Solutions, Platform and Edge**, go to **Developer Services** and click **Registry**.
   b. Choose the registry's region.
      You see all the repositories in the registry to which you have access. The image you pushed is in a new repository with a name constructed from:
      - the repository name in the address of the Docker registry in the Fn Project CLI context (see 6. Create an Fn Project CLI Context to Connect to Oracle Cloud Infrastructure)
      - the name of the helloworld-func image

      For example, the new repository might be called `acme-repo/helloworld-func`.
   c. Click the name of the new repository. You see details of the helloworld-func image that's been pushed to Oracle Cloud Infrastructure Registry.
9. (Optional) Use the Console to confirm that the function has been deployed to Oracle Functions successfully:

a. In the Console, open the navigation menu. Under **Solutions, Platform and Edge**, go to **Developer Services** and click **Functions**. If you're using the Limited Availability Release, go to the Oracle Functions landing page at https://console.us-phoenix-1.oraclecloud.com/functions.

b. Select the compartment specified in the Fn Project CLI context (see 6. Create an Fn Project CLI Context to Connect to Oracle Cloud Infrastructure).
The **Applications** page shows that an application called helloworld-app has been created.

c. Click the helloworld-app application to see the functions within it.
The **Functions** page shows that the helloworld-func function has been deployed to Oracle Functions.

10. In a terminal window, invoke the helloworld-func function by entering:

```
$ fn invoke helloworld-app helloworld-func
```

The 'Hello World !' message is displayed.

Congratulations! You've successfully created and deployed your first function to Oracle Functions!

Deploying a Function to Oracle Functions

You use Fn Project CLI commands to deploy functions to Oracle Functions.

Using Fn Project CLI Commands

To deploy a function to Oracle Functions using Fn Project CLI commands:

1. Confirm that you have completed the prerequisite steps for using Oracle Functions, as described in Preparing for Oracle Functions. Specifically, that you have:
   - set up your tenancy (see Configuring Your Tenancy for Function Development)
   - set up your development environment (see Configuring Your Client Environment for Function Development)

2. If the application to which you want to add the function doesn't yet exist in Oracle Functions, create it now using the Fn Project CLI or the Console. For example, you might create a new application called acmeapp. See Creating Applications.
3. Log in to your development environment as a functions developer.
4. In a terminal window, change directory to the directory containing the function code.
5. Initialize the function by entering:

   ```
   $ fn init --runtime <runtime-language> <function-name>
   ```

   where:

   - `<runtime-language>` is one of the supported runtime languages (currently go, java, node, and python are supported)
   - `<function-name>` is the name to use as the function name. If you don't specify a function name, the name of the current directory (in lower case) is used. Avoid entering confidential information.

   For example:

   ```
   $ fn init --runtime java acme-func
   ```

   A directory is created with the function name you specified, containing:

   - A function definition file called func.yaml, containing the minimum amount of information required to build and run the function. See the Fn Project documentation to find out about the additional parameters you can include in a func.yaml file.
   - A /src directory containing source files and directories.
   - A Maven configuration file called pom.xml that specifies the project artifacts and dependencies required to compile the function from the source files.

   Note that depending on the runtime language you specify, the `fn init` command might create an /example directory containing code for a helloworld application. As a matter of good practice, you'll probably want to delete the /example directory.
6. Change directory to the newly created directory.
7. Enter the following single Fn Project command to build the function and its dependencies

as a Docker image, push the image to the specified Docker registry, and deploy the function to Oracle Functions:

```
$ fn deploy --app <app-name>
```

where `<app-name>` is the name of the application in Oracle Functions to which you want to add the function. For example:

```
$ fn deploy --app acmeapp
```

Note that you can build, push, and deploy the function using separate Fn Project commands, instead of the single `fn deploy` command.

8. (Optional) Assuming the specified Docker registry is Oracle Cloud Infrastructure Registry, use the Console to confirm that the image has been pushed to Oracle Cloud Infrastructure Registry successfully:
    a. In the Console, open the navigation menu. Under **Solutions, Platform and Edge**, go to **Developer Services** and click **Registry**.
    b. Choose the registry's region.
    You see all the repositories in the registry to which you have access. The image you pushed is in a new private repository with a name constructed from:
        - the repository name in the address of the Docker registry in the Fn Project CLI context (see 6. Create an Fn Project CLI Context to Connect to Oracle Cloud Infrastructure)
        - the name of the image you pushed

    For example, the new repository might be called `acme-repo/acme-func`.
    c. Click the name of the new repository. You see details of the image that's been pushed to Oracle Cloud Infrastructure Registry
9. (Optional) Use the Console to confirm that the function has been deployed to Oracle Functions successfully:
    a. In the Console, open the navigation menu. Under **Solutions, Platform and Edge**, go to **Developer Services** and click **Functions**. If you're using the Limited Availability Release, go to the Oracle Functions landing page at https://console.us-phoenix-1.oraclecloud.com/functions.

b. Select the compartment specified in the Fn Project CLI context (see 6. Create an Fn Project CLI Context to Connect to Oracle Cloud Infrastructure).
The **Applications** page shows the applications in the compartment, including the one you specified in the `fn deploy` command.

c. Click the name of the application you specified in the `fn deploy` command to see the functions within it.
The **Functions** page shows that the function has been deployed to Oracle Functions.

Creating Applications

You can create applications in Oracle Functions in readiness for deploying functions. An application need not contain any functions.

You can create applications using the Console, the Fn Project CLI, and the API.

Using the Console

To create a new application in Oracle Functions using the Console:

1. Confirm that you have completed the prerequisite steps for using Oracle Functions, as described in Preparing for Oracle Functions. Specifically, that you have:
   - set up your tenancy (see Configuring Your Tenancy for Function Development)
   - set up your development environment (see Configuring Your Client Environment for Function Development)
2. Log in to the Console as a functions developer. If you're using the Limited Availability Release, go to https://console.us-phoenix-1.oraclecloud.com.
3. In the Console, open the navigation menu. Under **Solutions, Platform and Edge**, go to **Developer Services** and click **Functions**. If you're using the Limited Availability Release, go to the Oracle Functions landing page at https://console.us-phoenix-1.oraclecloud.com/functions.

4. Select the region you are using with Oracle Functions. Oracle recommends that you use the same region as the Docker registry that's specified in the Fn Project CLI context (see 6. Create an Fn Project CLI Context to Connect to Oracle Cloud Infrastructure).
5. Select the compartment specified in the Fn Project CLI context (see 6. Create an Fn Project CLI Context to Connect to Oracle Cloud Infrastructure).
   The **Applications** page shows the applications already defined in the compartment.
6. Click **Create Application** and specify:
   - A name for the new application (for example, acmeapp). Avoid entering confidential information.
   - The VCN and public subnet (or subnets) in which to run the function. For example, a VCN called acme-vcn-01 and a public subnet called Public Subnet IHsY:US-PHOENIX-AD-1). If a regional subnet has been defined, best practice is to select that subnet to make failover across availability domains simpler to implement. If a regional subnet has not been defined and you need to meet high availability requirements, select multiple subnets. Oracle recommends that the public subnets are in the same region as the Docker registry that's specified in the Fn Project CLI context (see 6. Create an Fn Project CLI Context to Connect to Oracle Cloud Infrastructure).
7. Click **Create**.
   The new application appears in the list of applications.

Using Fn Project CLI Commands

To create a new application in Oracle Functions using the Fn Project CLI:

1. Log in to your development environment as a functions developer.
2. In a terminal window, create a new application by entering:

   ```
   $ fn create app <app-name> --annotation oracle.com/oci/subnetIds='["<subnet-ocid>"]'
   ```

   where:

- `<app-name>` is the name of the new application. Avoid entering confidential information.
- `<subnet-ocid>` is the OCID of the public subnet (or subnets) in which to run functions. If a regional subnet has been defined, best practice is to select that subnet to make failover across availability domains simpler to implement. If a regional subnet has not been defined and you need to meet high availability requirements, specify multiple subnets (enclose each OCID in double quotes separated by commas, in the format `'["<subnet-ocid>","<subnet-ocid>"]'`). Oracle recommends that the public subnets are in the same region as the Docker registry that's specified in the Fn Project CLI context (see 6. Create an Fn Project CLI Context to Connect to Oracle Cloud Infrastructure).

For example:

```
 $ fn create app acmeapp --annotation oracle.com/oci/subnetIds='
["ocid1.subnet.oc1.phx.aaaaaaaacnh..."]'
```

An application is created in Oracle Functions, in the tenancy and region implied by the subnet OCID and belonging to the compartment specified in the Fn Project CLI context file.

3. Verify that the new application has been created by entering:

```
$ fn list apps
```

For example:

```
$ fn list apps

acmeapp
```

Using the API

Use these API operations to manage applications:

- CreateApp
- DeleteApp

- GetApp
- UpdateApp

Limited Availability Release: If you're using the Oracle Functions Limited Availability Release, click here to:

- obtain zip files containing the Oracle Functions Preview SDKs
- see code examples using the Oracle Functions Preview SDKs

Java, Python, Ruby, and Go SDKs are available.

When prompted, enter the password you were sent in your Limited Availability Release introductory email.

Creating Functions

You can create a new function definition in the Oracle Functions server in different ways:

- Using the Console or the Fn Project CLI command `fn create function` to create a new function based on an existing Docker image that has already been pushed to the Docker registry (as described in this topic).
- Using the single Fn Project CLI command `fn deploy` to build a new Docker image, push the image to the Docker registry, and create a new function based on the image in one step (as described in Deploying a Function to Oracle Functions).
- Using the API (see CreateFunction).

When using the Console or the `fn create function` command to create a new function based on an existing Docker image, you specify function metadata to store in the Oracle Functions server. For example, the maximum length of time the function is allowed to execute for.

The existing image on which you base a new function must be suitable for use with Oracle Functions. Typically, to build and push a suitable image, you or somebody else will use Fn Project CLI commands and/or Docker CLI commands. For example, having written your function code and a func.yaml file containing function metadata (perhaps based on a template helloworld function and func.yaml created using `fn init`), you can:

- Use `fn build` to build a new Docker image from the function.
- Use `docker push` to push the image to the Docker registry.

With the image in the Docker registry, you can then use the Console to create a function based on the image, as described in this topic.

Using the Console to create a new function from an existing Docker image

To use the Console to create a new function in the Oracle Functions server from an existing Docker image that has already been pushed to the Docker registry:

1. Log in to the Console as a functions developer. If you're using the Limited Availability Release, go to https://console.us-phoenix-1.oraclecloud.com.
2. In the Console, open the navigation menu. Under **Solutions, Platform and Edge**, go to **Developer Services** and click **Functions**. If you're using the Limited Availability Release, go to the Oracle Functions landing page at https://console.us-phoenix-1.oraclecloud.com/functions.
3. Select the region you are using with Oracle Functions. Oracle recommends that you use the same region as the Docker registry that's specified in the Fn Project CLI context (see 6. Create an Fn Project CLI Context to Connect to Oracle Cloud Infrastructure).
4. Select the compartment specified in the Fn Project CLI context (see 6. Create an Fn Project CLI Context to Connect to Oracle Cloud Infrastructure).
   The **Applications** page shows the applications defined in the compartment.
5. Click the name of the application in which you want to create the new function.
6. Click **Create Function** and specify:
   - **Name:** A name for the new function.
   - **Image:** The existing image in the Oracle Cloud Infrastructure Registry in your currently selected region. You first select the image repository, and then the image version.

- **Memory:** The maximum amount of memory the function can use during execution.
- **Timeout:** The maximum amount of time the function will be allowed to run for.
- **Tags:** Optionally, you can apply tags. If you have permissions to create a resource, you also have permissions to apply free-form tags to that resource. To apply a defined tag, you must have permissions to use the tag namespace. For more information about tagging, see Resource Tags. If you are not sure if you should apply tags, skip this option (you can apply tags later) or ask your administrator.

7. Click **Create** to create the new function in the Oracle Functions server.

The new function is shown in the Console, in the list of functions in the application you selected.

Using Fn Project CLI Commands

To use the Fn Project CLI to create a new function in the Oracle Functions server from an existing Docker image that has already been pushed to the Docker registry:

1. Log in to your development environment as a functions developer.
2. In a terminal window, create a new function by entering:

   ```
   $ fn create function <app-name> <function-name> <image-name>
   ```

   where:

   - `<app-name>` is the name of an existing application in which to create the new function.
   - `<function-name>` is the name of the new function you want to create. Avoid entering confidential information.
   - `<image-name>` is the name of the existing image in the Docker registry on which to base the new function.

   For example:

   ```
   $ fn create function acmeapp acme-func phx.ocir.io/acme-dev/acme-repo/acme-func:0.0.3
   ```

A new function is created in Oracle Functions, based on the existing image and with the name you specified

3. Verify that the new function has been created by entering:

```
$ fn list functions <app-name>
```

For example:

```
$ fn list functions acme-app

NAME            IMAGE
acme-func       phx.ocir.io/acme-dev/acme-repo/acme-func:0.0.3
```

Using the API

Use these API operations to create functions:

- CreateFunction

Limited Availability Release: If you're using the Oracle Functions Limited Availability Release, click here to:

- obtain zip files containing the Oracle Functions Preview SDKs
- see code examples using the Oracle Functions Preview SDKs

Java, Python, Ruby, and Go SDKs are available.

When prompted, enter the password you were sent in your Limited Availability Release introductory email.

Viewing Functions and Applications

Having deployed functions to Oracle Functions, you'll typically want to view the functions you've deployed, along with other functions in the same application and different applications. For example, you might want to see:

- all the applications in a compartment
- details of the image for a given function

You can view applications and functions using the Console, the Fn Project CLI, and the API.

Using the Console

To view details of applications and functions deployed to Oracle Functions using the Console:

1. Log in to the Console as a functions developer. If you're using the Limited Availability Release, go to https://console.us-phoenix-1.oraclecloud.com.
2. In the Console, open the navigation menu. Under **Solutions, Platform and Edge**, go to **Developer Services** and click **Functions**. If you're using the Limited Availability Release, go to the Oracle Functions landing page at https://console.us-phoenix-1.oraclecloud.com/functions.
3. Select the region you are using with Oracle Functions. Oracle recommends that you use the same region as the Docker registry that's specified in the Fn Project CLI context (see 6. Create an Fn Project CLI Context to Connect to Oracle Cloud Infrastructure).
4. Select the compartment containing the applications and functions that you want to see information about.
   The **Applications** page shows all the applications in the compartment you selected.
5. Click the name of an application to see the functions within it.
   The **Functions** page shows details for all the functions within the application you selected, including:
   - the Docker image created for each function
   - the names of triggers defined for each function (if any)
   - when the function was last updated
6. Click the name of a function on the **Functions** page to see additional information about that function including:
   - the values of timeout and memory configuration parameters
   - details of triggers defined for the function (if any)

Using Fn Project CLI Commands

To view details of applications and functions deployed to Oracle Functions using the Fn Project CLI:

1. Log in to your development environment as a functions developer.
2. If you want to see details about applications, in a terminal window:
    - Enter the following command to see a simple list of applications:

      ```
      $ fn list apps
      ```

      For example:

      ```
      $ fn list apps

      acme-app
      ```

    - Enter the following command to see more detail about a particular application:

      ```
      $ fn inspect app <app-name>
      ```

      For example:

      ```
      $ fn inspect app acme-app

      {
          "annotations": {
              "oracle.com/oci/appCode": "ff7jc5adw2a",
              "oracle.com/oci/compartmentId": "ocid1.compartment.oc1..aaaaaaaaw______nyq",
              "oracle.com/oci/subnetIds": [
                      "ocid1.subnet.oc1.phx.aaaaaaaao..."
              ],
              "oracle.com/oci/tenantId": "ocid1.tenancy.oc1..aaaaaaaap...keq"
          },
          "created_at": "2018-07-13T17:54:34.000Z",
          "id": "ocid1.fnapp.oc1.us-phoenix-1..3nigsx...",
          "name": "acme-app",
          "updated_at": "2018-07-13T17:54:34.000Z"
      }
      ```

3. If you want to see details about functions, in a terminal window:

- Enter the following command to see a simple list of functions in a particular application:

```
$ fn list functions <app-name>
```

For example:

```
$ fn list functions acme-app

NAME            IMAGE
acme-func       phx.ocir.io/acme-dev/acme-repo/acme-func:0.0.3
acme-func-dev   phx.ocir.io/acme-dev/acme-repo/acme-func-dev:0.0.7
acme-func-test  phx.ocir.io/acme-dev/acme-repo/acme-func-test:0.0.6
```

- Enter the following command to see more detail about a particular function:

```
$ fn inspect function <app-name> <function-name>
```

For example:

```
$ fn inspect function acme-app acme-func

{
    "annotations": {
        "fnproject.io/fn/invokeEndpoint": "https://fht7ns4mn2q.us-phoenix-
1.functions.oci.oraclecloud.com/invoke/ocid1.fnfunc.oc1..aaaa____uxoa",
        "oracle.com/oci/compartmentId": "ocid1.compartment.oc1..aaaaaaaaw______nyq",
    },
    "app_id": "ocid1.fnapp.oc1..3nigsx...",
    "created_at": "2018-07-26T12:50:53.000Z",
    "format": "default",
    "id": "ocid1.fnfunc.oc1..hkjsc...",
    "image": "phx.ocir.io/acme-dev/acme-repo/acme-func:0.0.3",
    "memory": 128,
    "name": "acme-func",
    "timeout": 30,
    "updated_at": "2018-07-26T13:59:18.000Z"
}
```

Using the API

Use these API operations to see details about applications and functions:

- ListApp
- ListFunctions

Limited Availability Release: If you're using the Oracle Functions Limited Availability Release, click here to:

- obtain zip files containing the Oracle Functions Preview SDKs
- see code examples using the Oracle Functions Preview SDKs

Java, Python, Ruby, and Go SDKs are available.

When prompted, enter the password you were sent in your Limited Availability Release introductory email.

Invoking Functions

You can invoke a function that you've deployed to Oracle Functions in different ways:

- Using the Fn Project CLI.
- Using the Oracle Cloud Infrastructure SDKs.
- Making a signed HTTP request to the function's invoke endpoint. Every function has an invoke endpoint.

Each of the above invokes the function via requests to the API. Any request to the API must be authenticated by including a signature and the OCID of the compartment to which the function belongs in the request header. Such a request is referred to as a 'signed' request. The signature includes Oracle Cloud Infrastructure credentials in an encrypted form.

If you use Fn Project CLI commands to invoke a function, authentication is handled for you. See Using Fn Project CLI Commands to Invoke Functions.

If you use an Oracle Cloud Infrastructure SDK to invoke a function, you can use the SDK to handle authentication. See Using SDKs to Invoke Functions.

If you make a signed HTTP request to a function's invoke endpoint, you'll have to handle authentication yourself by including a signature and the OCID of the compartment to which the function belongs in the request header. You can do this in different ways:

- Using a bash script provided by Oracle (commonly referred to as oci-curl) to send the signed request (see Sending a Signed Request to a Function's Invoke Endpoint (using oci-curl)).
- Writing code to programmatically sign requests (for information about the required credentials and how to sign the requests, see Request Signatures).

Using Fn Project CLI Commands to Invoke Functions

To invoke a function deployed to Oracle Functions using the Fn Project CLI:

1. Log in to your development environment as a functions developer.
2. In a terminal window, enter:

   ```
   $ fn invoke <app-name> <function-name>
   ```

   where:

   - `<app-name>` is the name of the application containing the function you want to invoke
   - `<function-name>` is the name of the function you want to invoke

   For example:

   ```
   $ fn invoke helloworld-app helloworld-func

   Hello World !
   ```

Tip

If you want to pass arguments and values to a function, prefix the `fn invoke` command with `echo -n '<argument>=<value>' |`

If the function is expecting the argument and value in JSON format, you can use the `fn invoke` command's `--content-type` parameter to specify `application/json`.

For example:

```
$ echo -n '{"name":"Bob"}' | fn invoke helloworld-app
helloworld-func --content-type application/json

Hello Bob !
```

Using SDKs to Invoke Functions

If you're writing a program to invoke a function in a language for which an Oracle Cloud Infrastructure SDK exists, Oracle recommends you use that SDK to send API requests to invoke the function. Among other things, the SDK will facilitate Oracle Cloud Infrastructure authentication.

Limited Availability Release: If you're using the Oracle Functions Limited Availability Release, click here to:

- obtain zip files containing the Oracle Functions Preview SDKs
- see code examples using the Oracle Functions Preview SDKs

Java, Python, Ruby, and Go SDKs are available.

When prompted, enter the password you were sent in your Limited Availability Release introductory email.

Sending a Signed Request to a Function's Invoke Endpoint (using oci-curl)

These instructions assume:

- you want to invoke a function using oci-curl as the functions developer that's configured for your development environment.
- you have already configured oci-curl appropriately (see 7. Set the Context for the Fn Project CLI Using the oracle.profile Parameter)

To invoke a function deployed to Oracle Functions by sending a signed request to the function's invoke endpoint using oci-curl:

1. Log in to your development environment as a functions developer.
2. Obtain the function's invoke endpoint:
    a. In a terminal window, enter:

    ```
    $ fn inspect function <app-name> <function-name>
    ```

    where:

    - `<app-name>` is the name of the application containing the function you want to invoke
    - `<function-name>` is the name of the function you want to invoke

    For example:

    ```
    $ fn inspect function helloworld-app helloworld-func

    {
       "annotations": {
            "fnproject.io/fn/invokeEndpoint": "https://fht7ns4mn2q.us-phoenix-
    1.functions.oci.oraclecloud.com/invoke/ocid1.fnfunc.oc1..aaaa____uxoa",

    ...
    }
    ```

    The function's invoke endpoint is the value of `"fnproject.io/fn/invokeEndpoint"` . For example,

`https://fht7ns4mn2q.us-phoenix-1.functions.oci.oraclecloud.com/invoke/ocid1.fnfunc.oc1..aaaa____uxoa` (abbreviated for readabiilty).

b. Copy the function's invoke endpoint.

3. In a terminal window, use the `source` command to set up the current shell environment for oci-curl by entering:

```
$ source <path-to-script>/oci-curl.sh
```

where `<path-to-script>` is the path to the location of the oci-curl.sh script (see 3. Create and Configure a Copy of oci-curl). For example:

```
$ source ~/oci-curl/oci-curl.sh
```

4. In the same terminal window in which you entered the `source` command, use oci-curl to invoke the function by sending a signed POST request to the function's invoke endpoint by entering:

```
oci-curl "<invoke-endpoint-host>" post <filename> "<invoke-endpoint-path>"
```

where:

- `<invoke-endpoint-host>` is the first half of the endpoint you obtained in the earlier step, excluding `https://` and up to (but not including) `/invoke`
- `<filename>` is the name of a file containing data to pass to the function (you must specify a file, even if it's empty)
- `<invoke-endpoint-path>` is the second half of the endpoint you obtained in the earlier step, from (and including)`/invoke` onwards

For example:

```
$ oci-curl "fht7ns4mn2q.us-phoenix-1.functions.oci.oraclecloud.com" post payload.json
"/invoke/ocid1.fnfunc.oc1..aaaa____uxoa"
```

5. Assuming a passphrase was provided to encrypt the API signing key (as recommended by Oracle), enter the passphrase when prompted.

Exporting Function Logs

When a function you've deployed to Oracle Functions is invoked, you'll often want to see the messages output by the function. For example, you might want a function to export message logs to an external logging service like Papertrail.

You can specify that functions in an application are to export their message logs to an external logging service using the Fn Project CLI and the API.

Using Fn Project CLI Commands

To export the logs of functions deployed to Oracle Functions to an external logging service using the Fn Project CLI:

1. Log in to your development environment as a functions developer.
2. If you want to create a new application and specify that all functions in the application export their logs to an external logging service:
    a. Enter:

    ```
    fn create app <app-name> --syslog-url <logging-service-url> --annotation
    oracle.com/oci/subnetIds='["<subnet-ocid>"]'
    ```

    where:

    - `<app-name>` is the name of the new application. Avoid entering confidential information.
    - `<logging-service-url>` is the syslog URL to which to export logs.
    - `<subnet-ocid>` is the OCID of the public subnet (or subnets) in which to run functions. If a regional subnet has been defined, best practice is to select that subnet to make failover across availability domains simpler to implement. If a regional subnet has not been defined and you need to meet high availability requirements, select multiple subnets (enclose each OCID in double quotes separated by commas, in the format `'["<subnet-ocid>","<subnet-ocid>"]'`. Oracle recommends that the public subnets are in the same region as the Docker registry that's specified in the Fn

Project CLI context (see 6. Create an Fn Project CLI Context to Connect to Oracle Cloud Infrastructure).

For example, `fn create app acmeapp --syslog-url tcp://my.papertrail.com:4242 --annotation oracle.com/oci/subnetIds='["ocid1.subnet.oc1.phx.aaaaaaaacnh..."]'`

b. Deploy one or more functions to the application you have just created (see Deploying a Function to Oracle Functions).

3. If you want to update an existing application and specify that all functions in the application export their logs to an external logging service, enter:

```
fn update app <app-name> --syslog-url <logging-service-url>
```

where:

- `<app-name>` is the name of the application to update
- `<logging-service-url>` is the syslog URL to which to export logs

For example, `fn update app acmeapp --syslog-url tcp://my.papertrail.com:4242`

Whenever a function is invoked in the application you just created or updated, its messages are exported to the syslog URL that you specified.

Using the API

Use these API operations to specify a syslog URL for an application:

- CreateApp
- UpdateApp

Limited Availability Release: If you're using the Oracle Functions Limited Availability Release, click here to:

- obtain zip files containing the Oracle Functions Preview SDKs
- see code examples using the Oracle Functions Preview SDKs

Java, Python, Ruby, and Go SDKs are available.

When prompted, enter the password you were sent in your Limited Availability Release introductory email.

Deleting Applications, Functions, and Triggers

You can delete applications, functions, and triggers in Oracle Functions that you or other functions developers have created, provided you have been granted the necessary permission (FN_APP_DELETE, FN_FUNCTION_DELETE, or FN_TRIGGER_DELETE as appropriate).

Note that application, function, and trigger deletion is permanent. You cannot undelete an application, function, or trigger that you've deleted.

You can delete applications, functions, and triggers using the Console, the Fn Project CLI, and the API.

Using the Console

When using the Console to delete applications, functions, and triggers, note that:

- when you delete a function, any triggers associated with the function are also deleted
- when you delete an application, all of its functions and any associated triggers are also deleted
- you're always prompted to confirm deletion because you cannot undelete an application, function, or trigger later

To delete applications, functions, and triggers in Oracle Functions using the Console:

1. Log in to the Console as a functions developer. If you're using the Limited Availability Release, go to https://console.us-phoenix-1.oraclecloud.com.
2. In the Console, open the navigation menu. Under **Solutions, Platform and Edge**, go to **Developer Services** and click **Functions**. If you're using the Limited Availability Release, go to the Oracle Functions landing page at https://console.us-phoenix-1.oraclecloud.com/functions.

3. Select the region you are using with Oracle Functions. Oracle recommends that you use the same region as the Docker registry that's specified in the Fn Project CLI context (see 6. Create an Fn Project CLI Context to Connect to Oracle Cloud Infrastructure).
4. Select the compartment specified in the Fn Project CLI context (see 6. Create an Fn Project CLI Context to Connect to Oracle Cloud Infrastructure).
   The **Applications** page shows the applications defined in the compartment.
5. To delete an application, and all of its functions and associated triggers:
   a. Click the name of the application you want to delete.
   b. On the **Application Detail** page, click **Delete** and confirm you want to delete the application as follows:
      - If the application does not have functions within it, click **Delete** to confirm that you want to delete the application.
      - If the application does have functions within it, you are shown a list of the functions in the application, and the number of triggers associated with each function. To delete the application, enter `DELETE <APPLICATION-NAME>` in the text box, and click **Delete**.
6. To delete a function and all its associated triggers:
   a. Click the name of the application containing the function you want to delete.
   b. On the **Application Detail** page, click the name of the function you want to delete
   c. On the **Function Detail** page, click **Delete** and confirm you want to delete the function as follows:
      - If the function does not have any triggers, click **Delete** to confirm that you want to delete the function.
      - If the function does have triggers, you are shown a list of the triggers. To delete the function, select the **Delete All Triggers For This Function** option, and click **Delete**.

Using Fn Project CLI Commands

When using the Fn Project CLI to delete applications, functions, and triggers, note that:

- you cannot delete an application if it contains functions (you must delete the functions first)
- you cannot delete a function that has associated triggers (you must delete the triggers first)

To delete applications, functions, and triggers in Oracle Functions using the Fn Project CLI:

1. Log in to your development environment as a functions developer.
2. To delete an application:
    a. In a terminal window, enter:

    ```
    $ fn delete app <app-name>
    ```

    where <app-name> is the name of the application to delete.
    For example:

    ```
    $ fn delete app acmeapp
    ```

    b. Verify that the application has been deleted by entering:

    ```
    $ fn list apps
    ```

3. To delete a function:
    a. In a terminal window, enter:

    ```
    $ fn delete function <app-name> <function-name>
    ```

    where:

    - <app-name> is the name of the application containing the function you want to delete.
    - <function-name> is the name of the function you want to delete.

    For example:

    ```
    $ fn delete function acmeapp acme-func
    ```

    b. Verify that the function has been deleted by entering:

    ```
    $ fn list functions <app-name>
    ```

For example:

```
$ fn list functions acmeapp
```

4. To delete a trigger:
    a. In a terminal window, enter:

    ```
    $ fn delete trigger <app-name> <function-name> <trigger-name>
    ```

    where:

    - <app-name> is the name of the application containing the function that has an associated trigger that you want to delete.
    - <function-name> is the name of the function that has the associated trigger that you want to delete.
    - <trigger-name> is the name of the trigger you want to delete.

    For example:

    ```
    $ fn delete trigger acmeapp acme-func acme-func-trigger
    ```

    b. Verify that the trigger has been deleted by entering:

    ```
    $ fn list triggers <app-name> <function-name>
    ```

    For example:

    ```
    $ fn list triggers acmeapp acme-func
    ```

Using the API

Use these API operations to delete applications, functions, and triggers:

- DeleteApp
- DeleteFunction
- DeleteTrigger

Limited Availability Release: If you're using the Oracle Functions Limited Availability Release, click here to:

- obtain zip files containing the Oracle Functions Preview SDKs
- see code examples using the Oracle Functions Preview SDKs

Java, Python, Ruby, and Go SDKs are available.

When prompted, enter the password you were sent in your Limited Availability Release introductory email.

Accessing File Systems from Running Functions

A function you've deployed to Oracle Functions can access the file system of the container in which it's running as follows:

- the function can read files from all directories
- the function can write files to the /tmp directory

For example, you might want a function to download an Excel file and then read its contents. To meet this requirement, you might create a function that writes the file to the /tmp directory in the container's filesystem, and then subsequently reads the file.

When writing files to the /tmp directory, the /tmp directory is generally always writable. However, the maximum allowable size of the /tmp directory depends on the maximum memory threshold specified for the function:

Maximum memory threshold for the function (MB)	Maximum allowed size of /tmp (MB)	Maximum allowed number of files (inodes) in /tmp
128MB	32MB	1024
256MB	64MB	2048
512MB	128MB	4096
1024MB	256MB	8192

Note that the /tmp directory might be shared by multiple invocations of the function. A file written by an earlier invocation of a function could still exist when the function is invoked a second time. It is your responsibility to delete any files to avoid unexpected behavior.

Using the Fn Project CLI with Oracle Functions

Oracle Functions is powered by the Fn Project open source engine. As a result, you can use the Fn Project CLI to perform create, read, update, and delete operations on Oracle Functions.

To enable you to use the Fn Project CLI with Oracle Functions, you perform a number of preparatory tasks. See Configuring Your Client Environment for Function Development.

Most Fn Project CLI commands have a similar syntax:

```
fn [global options] <command> [command options] [subcommands] [arguments]
```

For example, to:

- list all the available applications, use the command `fn list apps`
- create an application, use a command like `fn create app acmeapp --annotation oracle.com/oci/subnetIds='["ocid1.subnet.oc1.phx.aaaaaaaacnh..."]'`
- invoke a function, use a command like `fn invoke helloworld-app helloworld-func`
- change the profile that the Fn Project CLI uses for its context, use a command like `fn update context oracle.profile john-oci-profile`

To see a complete list of Fn Project CLI commands:

- Log in to your development environment as a functions developer and enter `fn --help` or `fn -h` in a terminal window.
- In a web browser, go to the Fn Project CLI documentation.

To see detailed information about individual Fn Project CLI commands:

- Log in to your development environment as a functions developer and enter `fn <command> [subcommand] --help` or `fn <command> [subcommand] -h` in a terminal window. For example:

    - `fn create --help`
    - `fn update app -h`
- In a web browser, go to the Fn Project CLI documentation and select the command from the list.

From time to time, new versions of the Fn Project CLI are released:

- To see which version of the Fn Project CLI is currently installed and whether it is the most recent version, log in to your development environment as a functions developer and enter `fn version` in a terminal window. The Fn Project CLI version number is displayed. If a more recent version of the Fn Project CLI is available, the number of the latest available version is also displayed.
- To upgrade the Fn Project CLI to the most recent version, reinstall the Fn Project CLI by following the instructions in 5. Install the Fn Project CLI.

Integrating Oracle Functions with Other Oracle Cloud Infrastructure Services

You can invoke functions in Oracle Functions from other Oracle Cloud Infrastructure services. Typically, you'll want an event in another service to initiate a request to a trigger defined in Oracle Functions. The request to the trigger invokes the associated function.

This functionality is not available in the Limited Availability Release.

Changing Oracle Functions Default Behavior

You can change several aspects of Oracle Functions default behavior using configuration parameters and environment variables.

Depending on the parameter, you can override a default value by specifying an alternative value in the following ways (note the order of precedence):

- by adding an entry to the func.yaml file (which overrides default values)
- by explicitly setting an environment variable (which overrides values set in the func.yaml file)
- by including a command option when you invoke the function using the Fn Project CLI (which overrides values set in environment variables or in the func.yaml file)

The following table indicates the parameters you can set, the default value, and where the default value can be overridden.

Parameter Description	Default Value	Units	func.yaml Parameter	Environment Variable	Fn CLI option	Notes
Maximum time a function will be allowed to run.	30	Seconds	timeout	n/a	n/a	Maximum value: 120
Maximum memory threshold for a function.	128	MB	memory	FN_MEMORY	--memory	One of: • 128 • 256 • 512 • 1024 If this limit is exceeded during execution, the function is stopped and an error message is logged.

For more information about the above parameters, and other configuration parameters, see Func files in the Fn Project documentation.

Differences between Oracle Functions and Fn Project

In general, Oracle Functions and Fn Project are very similar. However there are some differences, as detailed below.

Differences in Authentication When Making API Calls

When you use the Oracle Cloud Infrastructure API with Oracle Functions, in the request header you have to provide:

- the OCID of the compartment to which the function belongs
- Oracle Cloud Infrastructure authentication details

Differences When Invoking Functions

To invoke a function deployed to Oracle Functions, you have to explicitly specify an Oracle Cloud Infrastructure endpoint (unless you're using the Fn Project CLI).

For example, when you use oci-curl to invoke a function, you have to send a request to the function's invoke endpoint (for example `https://fht7ns4mn2q.us-phoenix-1.functions.oci.oraclecloud.com/invoke/ocid1.fnfunc.oc1..aaaa____uxoa`).

You can obtain the appropriate endpoint by making a call to the API, either directly or by using the Fn Project CLI command:

```
$ fn inspect function <app-name> <function-name>
```

Additional Context Configuration Parameters in Oracle Functions

As well as supporting Fn Project context configuration parameters, Oracle Functions also has some additional parameters, as shown in the following table.

Additional Parameter	Set in	Value	Notes
`provider`	A context configuration .yaml file in ~/.fn/contexts	`oracle`	Enables Oracle Functions rather than Fn Project functionality. When `provider` is set to `oracle`, the following parameters are valid: • `oracle.compartment-id` • `oracle.profile` See 6. Create an Fn Project CLI Context to Connect to Oracle Cloud Infrastructure.
`oracle.compartment-id`	A context configuration .yaml file in ~/.fn/contexts	`<compartment-ocid>`	Specifies the OCID of the Oracle Cloud Infrastructure compartment that owns Oracle Functions resources. See 6. Create an Fn Project CLI Context to Connect to Oracle Cloud Infrastructure.
`oracle.profile`	A context configuration .yaml file in ~/.fn/contexts	`<profile-name>`	Specifies which profile to use from the ~/.oci/config file. If not set, the profile named default is used. See 7. Set the Context for the Fn Project CLI Using the oracle.profile Parameter

Use of Annotations

When you're creating and viewing Oracle Functions resources using the Fn Project CLI, annotations enable you to identify and specify associated Oracle Cloud Infrastructure resources.

For example:

- When you're using the Fn Project CLI to create a new application, you use the `--annotation` parameter to specify the OCID of the public subnet in which to run the function.
- When you're using the Fn Project CLI to view the properties of a function, the `annotations` element shows the OCID of the compartment that owns the function.

Note that unlike other configuration parameters and environment variables, annotation values cannot be passed as arguments to running Docker containers.

Troubleshooting Oracle Functions

This topic covers common issues related to Oracle Functions and how you can address them.

Using DEBUG=1 to see more details about an error

If you encounter an unexpected error when using an Fn Project CLI command, you can find out more about the problem by starting the command with the string `DEBUG=1` and running the command again. For example:

```
$ DEBUG=1 fn invoke helloworld-app helloworld-func
```

Note that `DEBUG=1` must appear before the command, and that `DEBUG` must be in upper case.

Using --display-call-id when invoking functions to aid issue resolution

If you encounter an issue when invoking a function, you can engage with Oracle Support. Oracle Support can investigate the issue more efficiently if you provide the call id of the function invocation. You can obtain the call id using the `--display-call-id` command option. For example:

```
$ fn invoke helloworld-app helloworld-func --display-call-id



Call ID: 01CS23SDG71BT2N9GZJ002DQM5

Hello World !
```

Invoking any function in an application always returns an error

If you see a message similar to the following when invoking any function in an application, double-check that at least one internet gateway has been defined for the VCN specified for the application:

```
{"message":"Failed to pull image 'iad.ocir.io/acme-tenancy/acme-app/acme-func:0.0.3': Get
https://iad.ocir.io/v2/: net/http: request canceled while waiting for connection (Client.Timeout
exceeded while awaiting headers)"}

Fn: Error calling function: status 500
```

If an internet gateway has not been defined for the VCN already, define one now.

Creating a new application displays an error message in the New Application dialog

If you've already reached the limit for the number of applications in your tenancy, you might see a message similar to the following in the **New Application** dialog when trying to create a new application:

Unable to create your app, please try again.

Double-check how many applications already exist in your tenancy. Compare that with the number of applications you're allowed to create. See Oracle Functions Capabilities and Limits.

If you've exceeded the number of applications allowed in your tenancy, consider:

- deleting unwanted applications (see Deleting Applications, Functions, and Triggers)
- requesting an increase to the application limit (see Service Limits for instructions)

Running Fn Project CLI commands returns a 401 error

If you see a message similar to the following when running an Fn Project CLI command, double-check that the credentials specified for your current profile in the ~/.oci/config file are authenticating you correctly:

```
$ fn list apps

Fn: [GET /apps][401] ListApps default  &{Fields: Message:}
```

For example:

- Does `user` specify the OCID of your Oracle Cloud Infrastructure user account?
- Does `fingerprint` specify the fingerprint of the public API key value uploaded to the Console?
- Does `key_file` specify the full path to the private key file?

See 2. Create a Profile in the Oracle Cloud Infrastructure CLI Configuration File. Also see API Errors .

Running Fn Project CLI commands returns a 404 error

If you see a message similar to the following when running an Fn Project CLI command, double-check that you are authorized to access function-related and network resources:

```
$ fn list apps

Fn: [GET /apps][404] ListApps default  &{Fields: Message:Resource is not authorized or not found}
```

For example:

- Does `oracle.compartment-id` in your current context correctly specify the OCID of the compartment that owns deployed functions?
- Have policies been set up correctly to give group access to function-related and network resources?
- Does your user account belong to the group to which access to function-related and network resources has been granted?

See 6. Create an Fn Project CLI Context to Connect to Oracle Cloud Infrastructure and Create Policies to Control Access to Function-Related Resources. Also see API Errors .

Oracle Functions attempts to interact with docker.io

If you see a message similar to the following when deploying a function, double-check that your development environment doesn't have the FN_REGISTRY environment variable set to your Docker username:

```
The push refers to repository [docker.io. ...
.
.
.
denied: requested access to the resource is denied
Fn: error running docker push, are you logged into docker?: exit status 1
See fn <command> --help' for more information.
```

If you have used the open source Fn Project platform, you might have followed instructions in the Fn Project documentation to set the FN_REGISTRY environment variable to your Docker username to enable interaction with the official Docker registry.

The FN_REGISTRY environment variable overrides the value of the registry option in your Fn Project CLI context.

To use the Fn Project CLI with Oracle Functions, do one of the following:

- Unset the FN_REGISTRY variable.

- Override the FN_REGISTRY variable using the `--registry` global option whenever you enter an Fn Project CLI command that interacts with Oracle Cloud Infrastructure Registry.

Running `fn version` shows that a more recent version of the Fn Project CLI is available

If you see a message similar to the following when you enter the `fn version` command, a more recent version of the Fn Project CLI is available:

```
$ fn version

Client version: 0.5.33 is not latest: 0.5.34
Server version: ?
```

To upgrade the Fn Project CLI to the most recent version, reinstall the Fn Project CLI by following the instructions in 5. Install the Fn Project CLI.

Deploying a function to Oracle Functions returns "Fn: Missing subnets annotation" message

When you deploy a function to Oracle Functions, you might see the following message:

```
$ fn deploy --app joes-helloworld-app
Deploying helloworld-func to app: joes-helloworld-app
.
.
.
Fn: Missing subnets annotation
```

If you see the `Fn: Missing subnets annotation` message, confirm that you entered the correct application name. For example, you might have misspelled the application name, or the application might not be in the compartment currently specified by the Fn Project CLI context.

Invoking a function returns a timeout message and a 504 error

When you invoke a function that you've deployed to Oracle Functions, the function execution is subject to a maximum memory threshold. If this limit is exceeded, function execution stops and the following error message is returned:

```
{"code":"StatusGatewayTimeout","message":"Container initialization timed out, please ensure you are
using the latest fdk / format and check the logs"}
Fn: Error calling function: status 504
```

If you see this error, increase the maximum memory threshold when you invoke the function. Valid values for the maximum memory threshold are 128MB, 256MB, 512MB, and 1024MB (see Changing Oracle Functions Default Behavior).

For example, to set a function's maximum memory threshold to 256MB:

- Use the following syntax when invoking the function using the Fn Project CLI. This will set the maximum memory threshold to 256MB for the current function invocation:

  ```
  $ fn invoke <app-name> <function-name> --memory 256
  ```

- Add the following line to the function's func.yaml file. This will set the maximum memory threshold to 256MB whenever the function is invoked:

  ```
  memory: 256
  ```

  Note that if you edit the func.yaml file, you must re-deploy the function to Oracle Functions before invoking it again.

It's a good idea to use Fn Project CLI version 0.5.44 (or later) when creating a helloworld Python function. When you enter the `fn init --runtime python <function-name>` command to create the helloworld function, the line `memory: 256` is added to the func.yaml file automatically.

Invoking a function returns a StatusServiceUnavailable message and a 503 error

When you invoke a function that you've deployed to Oracle Functions, you might see the following error message:

```
{"code":"StatusServiceUnavailable","message":""subnet ocid1.subnet.oc1.phx.aaaaaaaac... does not exist
or FaaS is not authorized to use it"}
Fn: Error calling function: status 503
```

If you see this error, it's likely that a policy has not been created to give Oracle Functions access to network resources. See Create a Policy to Give the Oracle Functions Service Access to Virtual Network Resources.

Invoking a function returns an InternalServerError message and a 500 error

Oracle Functions enables you to export a function's logs to an external logging service (like Papertrail) by setting a syslog URL for the application. See Exporting Function Logs.

If the syslog URL is invalid or unreachable, you will see the following error when you invoke the function:

```
{"code":"InternalServerError","message":"Internal server error"}
Fn: Error calling function: status 500
```

To confirm that external logging service's URL is the cause of the error:

1. Update the application to unset `--syslog-url`. For example, by entering `fn update app helloworld-app --syslog-url ""`
2. Deploy the function you want to run. See Deploying a Function to Oracle Functions.
3. Invoke the function. See Invoking Functions.

If the function runs successfully, the external logging service's URL is not reachable from the subnet in which the function is running. Double-check that:

- the external logging service's URL is valid
- the external logging service's URL is publicly accessible
- the subnet in which the function is running has outbound access to the public internet

Details for Functions

This topic covers details for writing policies to control access to Oracle Functions.

Resource-Types

Aggregate Resource-Type

- `functions-family`

Individual Resource-Types

- `fn-app`
- `fn-function`
- `fn-trigger`
- `fn-invocation`

Comments

A policy that uses `<verb> functions-family` is equivalent to writing one with a separate `<verb> <individual resource-type>` statement for each of the individual resource-types.

See the table in Details for Verb + Resource-Type Combinations for a detailed breakout of the API operations covered by each verb, for each individual resource-type included in `functions-family`.

Supported Variables

Oracle Functions supports all the general variables (see General Variables for All Requests).

Details for Verb + Resource-Type Combinations

The following tables show the permissions and API operations covered by each verb. The level of access is cumulative as you go from `inspect` > `read` > `use` > `manage`. A plus sign (+) in a table cell indicates incremental access compared to the cell directly above it, whereas "no extra" indicates no incremental access..

For example, the `read` verb for the `fn-app` resource-type includes the same permissions and API operations as the `inspect` verb, plus the FN_APP_READ permission and the `GetApp` API operation. In the case of the `fn-app` resource-type, the `use` verb covers no additional permissions or API operations compared to `read`. Lastly, `manage` covers more permissions and operations compared to `use`.

fn-app

INSPECT

Permissions	APIs Fully Covered	APIs Partially Covered
FN_APP_LIST	`ListApp`	*none*

READ

Permissions	APIs Fully Covered	APIs Partially Covered
INSPECT +	*INSPECT* +	*none*
FN_APP_READ	`GetApp`	

USE

Permissions	APIs Fully Covered	APIs Partially Covered
no extra	*no extra*	*none*

MANAGE

Permissions	APIs Fully Covered	APIs Partially Covered
USE +	*USE* +	*none*
FN_APP_CREATE	`CreateApp`	
FN_APP_DELETE	`DeleteApp`	
FN_APP_UPDATE	`UpdateApp`	

fn-function

INSPECT

Permissions	APIs Fully Covered	APIs Partially Covered
FN_FUNCTION_LIST	`ListFunctions`	*none*

READ

Permissions	APIs Fully Covered	APIs Partially Covered
INSPECT +	*INSPECT +*	*none*
FN_FUNCTION_READ	`GetFunction`	

USE

Permissions	APIs Fully Covered	APIs Partially Covered
no extra	*no extra*	*none*

MANAGE

Permissions	APIs Fully Covered	APIs Partially Covered
USE +	*USE +*	*none*
FN_FUNCTION_CREATE	`CreateFunction`	
FN_FUNCTION_DELETE	`DeleteFunction`	
FN_FUNCTION_UPDATE	`UpdateFunction`	

fn-trigger

INSPECT

Permissions	APIs Fully Covered	APIs Partially Covered
FN_TRIGGER_LIST	`ListTriggers`	*none*

READ

Permissions	APIs Fully Covered	APIs Partially Covered
INSPECT +	*INSPECT +*	*none*
FN_TRIGGER_READ	`GetTrigger`	

USE

Permissions	APIs Fully Covered	APIs Partially Covered
no extra	*no extra*	*none*

MANAGE

Permissions	APIs Fully Covered	APIs Partially Covered
USE +	*USE +*	*none*
FN_TRIGGER_CREATE	`CreateTrigger`	
FN_TRIGGER_DELETE	`DeleteTrigger`	
FN_TRIGGER_UPDATE	`UpdateTrigger`	

fn-invocation

INSPECT

Permissions	APIs Fully Covered	APIs Partially Covered
none	*none*	*none*

READ

Permissions	APIs Fully Covered	APIs Partially Covered
none	*none*	*none*

USE

Permissions	APIs Fully Covered	APIs Partially Covered
FN_INVOCATION	`InvokeFunction`	*none*

MANAGE

Permissions	APIs Fully Covered	APIs Partially Covered
no extra	*no extra*	*none*

Permissions Required for Each API Operation

The following table lists the API operations in a logical order, grouped by resource type. For information about permissions, see Permissions.

API Operation	Permissions Required to Use the Operation
`CreateApp`	FN_APP_CREATE
`DeleteApp`	FN_APP_DELETE
`ListApp`	FN_APP_LIST
`GetApp`	FN_APP_READ
`UpdateApp`	FN_APP_UPDATE
`CreateFunction`	FN_FUNCTION_CREATE
`DeleteFunction`	FN_FUNCTION_DELETE
`ListFunctions`	FN_FUNCTION_LIST
`GetFunction`	FN_FUNCTION_READ
`UpdateFunction`	FN_FUNCTION_UPDATE
`CreateTrigger`	FN_TRIGGER_CREATE
`DeleteTrigger`	FN_TRIGGER_DELETE
`ListTriggers`	FN_TRIGGER_LIST
`GetTrigger`	FN_TRIGGER_READ
`UpdateTrigger`	FN_TRIGGER_UPDATE
`InvokeFunction`	FN_INVOCATION