***Chapter 2*: Understanding Identity and Access Management**

The **Oracle Cloud Infrastructure** (**OCI**) **Identity and Access Management** (**IAM**) service gives you full control of your cloud environment, allowing you to decide what type of access you want to give to a group of users. This is done through the unique approach of having an **Oracle Cloud ID** (**OCID**) assigned to each resource in your cloud environment.

OCI IAM includes its own IAM service, which can be integrated with an existing Microsoft Active Directory configuration using **Active Directory Federation Services** (**AD FS**), along with any SAML 2.0-compliant **Identity Provider** (**IdP**). OCI also offers integration with the Oracle Identity Cloud Service for those customers who have previously created IAM entities in that service. This integration enables customers to manage their IAM entities within the OCI console, regardless of whether they have been created within OCI or inside the broader Oracle Cloud Identity Service. The concepts and guidance that follow are the same regardless of the approach taken.

IAM uses traditional identity concepts, such as principals, users, groups, and policies, and introduces a new feature, called compartments. IAM Principal is the foundation of OCI IAM; therefore, we will be discussing this first before moving on to other constructs.

In this chapter, we will cover the following topics:

* Principals
* Organizing resources using compartments
* Accessing resources from compartments using policies
* Using instance principals to make a call to the OCI API
* Federating OCI access using a third-party IdP

**Principals**

Like any other IAM, OCI IAM also has principals. OCI IAM Principal is a method that allows you to interact with OCI resources. There are three types of principals, and we will define each of them next.

**The root user**

This is the very first user on the OCI account. A root user is persistent in nature and has full administrator access to all of the OCI resources on the account.

**IAM users/groups**

Users are persistent and can be individual people or applications, whereas a group is a collection of users. You can put the same users into multiple groups. These users enforce the policy of least privilege. Users have no permissions until they are placed in one, or more, groups.

**Instance principals**

**Instance Principals** are used when you want to call the OCI API from an instance deployed on top of OCI itself. The main benefit of using an instance principal is that you don't have to store any credentials within the instance to make these API calls. Oracle uses **Dynamic Groups** to implement instance principals and control them by using a policy definition. You can create these dynamic groups based on matching rules that decide who will be part of this group. We will go through this in more detail in the *Using instance principals to make a call to the OCI API* section of this chapter.

A dynamic group is accompanied by the required policies to access OCI resources. You can view a logical diagram of the OCI IAM in the following diagram:

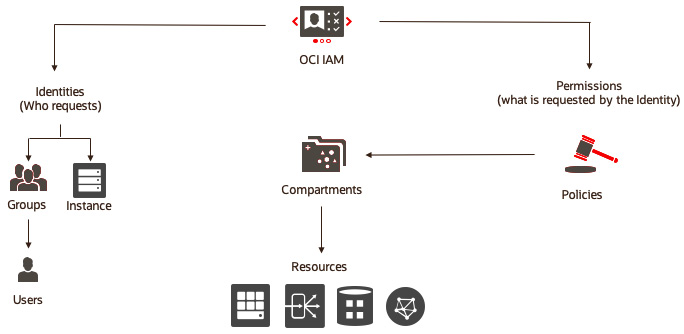


Figure 2.1 – OCI IAM login diagram

We will discuss how instance principals work in greater detail toward the end of this chapter. However, before that, let's look at how IAM works in conjunction with principals.

An IAM service authenticates a principal using three things: the username, the password, and the API signing key.

**The username and password**

You need to have a password to sign in to the OCI web console. The administrator of a tenancy sets a one-time password for your account. When you first log in, you will be asked to change this password.

**The API signing key**

You need to have an OCI API signing key when you want to send authenticated calls to the OCI API from the **Software Development Kit/Command Line Interface** (**SDK/CLI**). In order to have an API signing key, you need to input an RSA key in PEM format (which has a minimum of 2,048 bits).

Let's go through the process of generating an API signing key.

**Generating an API signing key**

Using an API key is the standard method for OCI authentication. Every standard authentication mode (the non-standard being Swift/SigV4) uses an API key to make a signed request API call.

You can generate a key pair using **openssl**. The recommended way is to use an encrypted password while creating the key pair. Let's use the following command to generate a password-protected private key:

$ openssl genrsa -out private\_key.pem -aes128 2048

Generating RSA private key, 2048 bit long modulus

.............................................+++

...........................+++e is 65537 (0x10001)

Enter pass phrase for private\_key.pem:

Verifying - Enter pass phrase for private\_key.pem:

$ cat private\_key.pem

-----BEGIN RSA PRIVATE KEY-----

Proc-Type: 4,ENCRYPTED

DEK-Info: AES-128-CBC,5EE51D737AC6938FC830BF54E86ACB23

mg7euVlAV05QSdAxkjEyE6CubUjpoGSAALYOs5vIvTIamN3W7SjGtKy WXEcv3p11

…………………

cI93H0riOTlEh2cmxp97So9/0/+c3atA05ld6btsqzo9nofiR49cDoR HSTi2nfDC

0WGOWTiNSuNhzJdE0nc9MzscTQ7fEfNCUuN+56bsjmT45wIooZrpOCq CGUem8j/q

-----END RSA PRIVATE KEY-----

Change the permission of the PEM file so that only you can read the key. To do this, you can use the following command:

$ chmod go-rwx private\_key.pem

Now, let's generate the public key using the following command:

$ openssl rsa -pubout -in private\_key.pem -out public\_key.pem

Enter pass phrase for private\_key.pem:

writing RSA key

$ cat public\_key.pem

-----BEGIN PUBLIC KEY-----

MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAwAR9a/LRwoU 1UoyFdA+5

fIczdM4se6Yvp5dFUz5yJJu757P+3Ro8HA9qxw2UIOwQ6ADqmjQXp0t RU27SSpJd

…………

T/f99DU5pMnBR/QB1jmwN82Rym1Mx2Qx8qPqHl+isXUuiAfrHO9N/Ue iqASU1JIc

7QIDAQAB

-----END PUBLIC KEY-----

The preceding command generates the key. Next, we will verify whether that key matches with the public key.

**Verifying whether a private key matches a public key**

There may be occasions where you might not be sure if you are using the right private key. The way in which to ensure that a private key matches a certain public key is to check the modulus and see whether it matches. The following is a set of commands that you can use to verify this:

$ openssl rsa -noout -modulus -in private\_key.pem

Enter pass phrase for private\_key.pem:

Modulus

$ openssl rsa -pubin -in public\_key.pem -text -noout

Public-Key: (2048 bit)

Modulus:

    00:bc:b8:f8:2b:4c:c2:f8:95:a3:43:54:fe:b9:5e:

    27:56:05:75:3e:fb:42:ec:22:e9:7b:fd:26:52:2f:

    80:d0:9e:e4:c8:df:93:2f:9c:7a:01:4b:4e:7e:2b:

    ……..

    c9:69

Exponent: 65537 (0x10001)

From the preceding output of both commands, you can see that the modulus of both the private and public certificate matches. However, for security reasons, we have omitted the whole output of the public key's modulus.

So, you can see how we use **openssl** to generate the API signing key. Now, let's generate a fingerprint and upload this to the OCI portal.

**Generating a fingerprint**

When you upload your public key to the identity control plane, you get a key ID in return:

* The key format is **tenantId/userId/fingerprint**.
* **fingerprint** is the fingerprint of your public key.

If you accidentally lose your key ID, then you can generate the fingerprint again using the following command:

$ openssl rsa -in public\_key.pem -pubout -outform DER | openssl md5 -c

d4:1d:8c:d9:8f:00:b2:04:e9:80:09:98:ec:f8:42:7e

**Uploading the public key**

To upload the public PEM key, you need to first log in to the OCI console. This is located at <https://console.us-phoenix-1.oraclecloud.com/>. Follow these steps:

1. Log in to the OCI console.
2. Open the **Profile** menu and click on **User Settings**.
3. Click on **API Keys**.
4. Click on **Add Public Key**.
5. Choose the **PASTE PUBLIC KEYS** option.
6. Copy the public PEM key contents.
7. Paste the copied content of the PEM file into the dialogue box and click on **Add**.

You will notice the fingerprint of the public key is displayed, as shown in the following screenshot:

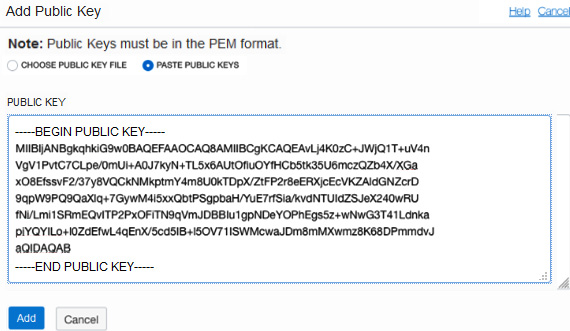


Figure 2.2 – Uploading the public key

Now you are all set to use a private key and fingerprint to get yourself authenticated using the OCI CLI or other tools that require you to use an OCI API key.

In the next section, let's discuss authorization, which is a type of privilege that you need to operate on OCI resources.

**Authorization**

You can define what privileges you need in policies and associate them with principals. OCI authorization works on the least privilege-first approach, so you are not allowed to perform any actions.

You can specify one or more policy statements in a human-readable format. These policy statements allow you to gain access to OCI resources and outline what you can do with them.

Let's take a look at a few human-readable policy statements:

Allow group <group\_name> to <verb> <resource-type> in tenancy

Allow group <group\_name> to <verb> <resource-type> in compartment <compartment\_name> [where <conditions>]

You can either attach these policies to a compartment or at the tenancy level.

**Organizing resources using compartments**

Compartments are a hierarchical construct, that is, they are a logical boundary that groups resources, each of which can exist in only one compartment. This construct can be used for a number of purposes, including the following:

* **IAM**: This is where you can group resources in a compartment for the purpose of restricting access to those resources.
* **Metering**/**Billing**: You can set limits on resource usage within a compartment; alternatively, you can bill the usage within a compartment to a specific contract.
* **Visibility**/**Compliance**/**Audits**: People in a particular department should only know about the resources in their compartment. You can distinguish between resources/usage in this compartment in order to apply specific governance/compliance rules.
* **Mergers**/**Acquisitions**/**Changes**: You can move an acquired company's tenancy inside of its new parent company's tenancy; alternatively, you can remove a subsidiary's compartment from the parent tenancy.

When you sign up for OCI, Oracle will create an organization tenancy for you, by default. From an end user perspective, this is your top-level resource container. Because this is something that gets created by OCI, you should not place or create resources at this level. You have full control of all compartments (and the resources within those compartments) within the tenancy.

Just as folders are hierarchical groupings of resources that can have restrictions set upon them, but also be easily amended to meet the changing needs of the organization, so too are compartments—a hierarchical grouping of cloud resources.

Compartments allow customers to set up a logical boundary that groups resources. Each resource in OCI today must exist in only one compartment. Compartments are a hierarchical construct; they allow customers to manage their resources vertically.

**Design considerations**

There are certain design considerations that you have to keep in your mind while designing the compartment model. We have listed them as follows:

* Every resource resides in a single compartment, but you can share these resources across different compartments as well.
* Aside from OCI Object Storage buckets, you can't reassign a resource to a different compartment after creation.
* You can delete or rename a compartment after creation.
* A compartment can have sub-compartments. This can go down to *six levels deep*.
* If you are the administrator of the account, then you can access this compartment without a policy assigned to it. Otherwise, you need at least one policy assigned to it.
* A sub-compartment inherits access permissions from its parent compartment.
* A policy is always attached to a compartment. So, when you write an IAM policy, you have to specify which compartment you want it to act on.
* We recommend that you use a separate compartment for the network resources of differing security levels and each team/project.

When you sign up for OCI, Oracle creates a default administrators account for your tenancy. It also creates a default group of administrators as well. You cannot delete this group, and there must always be at least one member in it. If you put any other users into this group, then those users will have full access to all of the resources. By default, there will be a policy that is generated and allows administrator groups access to all resources. You cannot delete this policy, nor it can be changed. This is depicted in the following diagram:

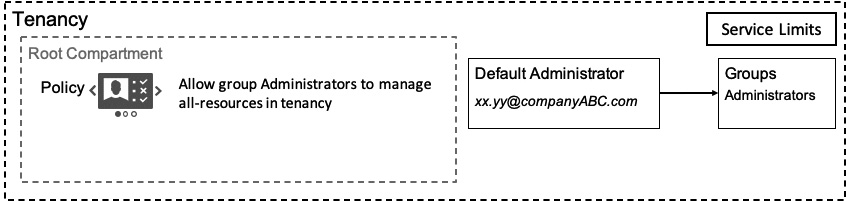


Figure 2.3 – Compartment constructs

So far, you have learned about the design considerations of compartments and have also learned how to organize resources using compartments. In the next section, we will discuss a couple of reference models that will allow you to organize your compartments based on their usage function.

**Reference model of compartments**

Let's take a look at an ideal reference architecture for model compartments, as depicted in the following screenshot. Here, we have created a **NetworkInfra** compartment, which has all the critical resources, and then we have divided this into another compartment layer:

* **NetworkInfra**: This is a compartment for critical network infrastructure components that are centrally managed by network administrators.
* **Dev, Test, and Prod networks**: This compartment is modeled as a separate compartment and allows you to easily write a policy about who can use the network.
* **Project** (**A**/**B**/**C**): This compartment is used for the resources used by a particular team or project; it is separated for the purposes of distributed management.

The compartment design model is shown in the following diagram:

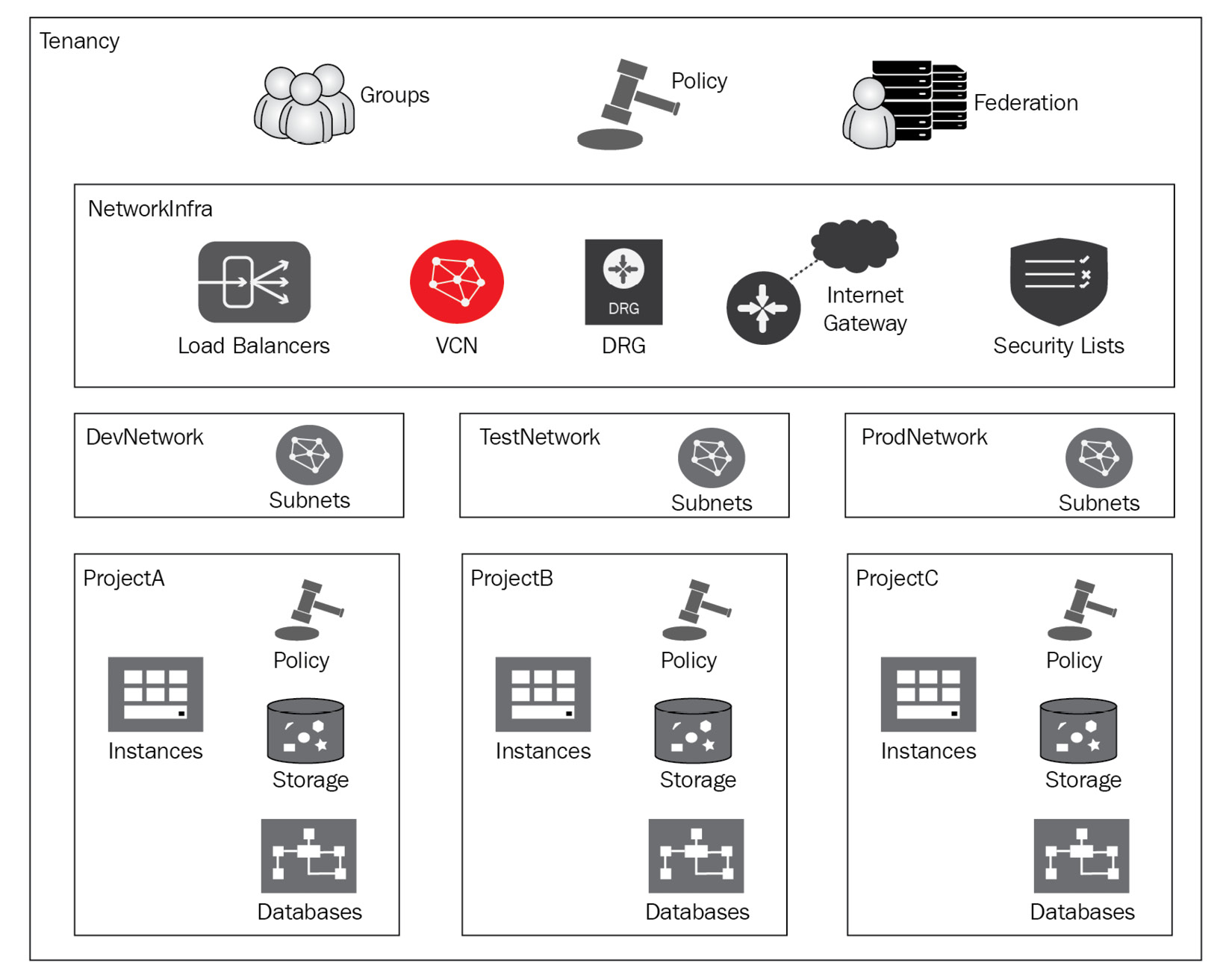


Figure 2.4 – The compartment design model

Ideally, you should use a sharable structure around the compartment for better manageability. Having said that, this will always give us the benefit of managing resources by functions than anything else, for example, having a compartment of shared IT resources and provisioning network and IT infrastructure resources onto that.

**Compartment Explorer**

Customers need the ability to see their entire enterprise organized by compartment so that they can easily view their allocated resources across their nested compartments. Compartment Explorer allows customers to view all the resources of a particular compartment in the context of a compartment tree. The benefit of this is that you can easily find and manipulate resources across the entire enterprise in a super convenient way. Additionally, you can view all the information of a particular resource, delete a resource, and move that resource between compartments.

The following steps demonstrate how to access the Compartment Explorer.

To check the **Compartment Explorer**, you need to first log in to the OCI console, which is located at<https://console.us-phoenix-1.oraclecloud.com/>:

1. Log in to the OCI console.
2. Open the **Navigation** menu.
3. Under **Governance and Administration**, expand **Governance** and select **Compartment Explorer**.

Here, you can see the Explorer, and from there, you can browse through your resources, as shown in the following screenshot:

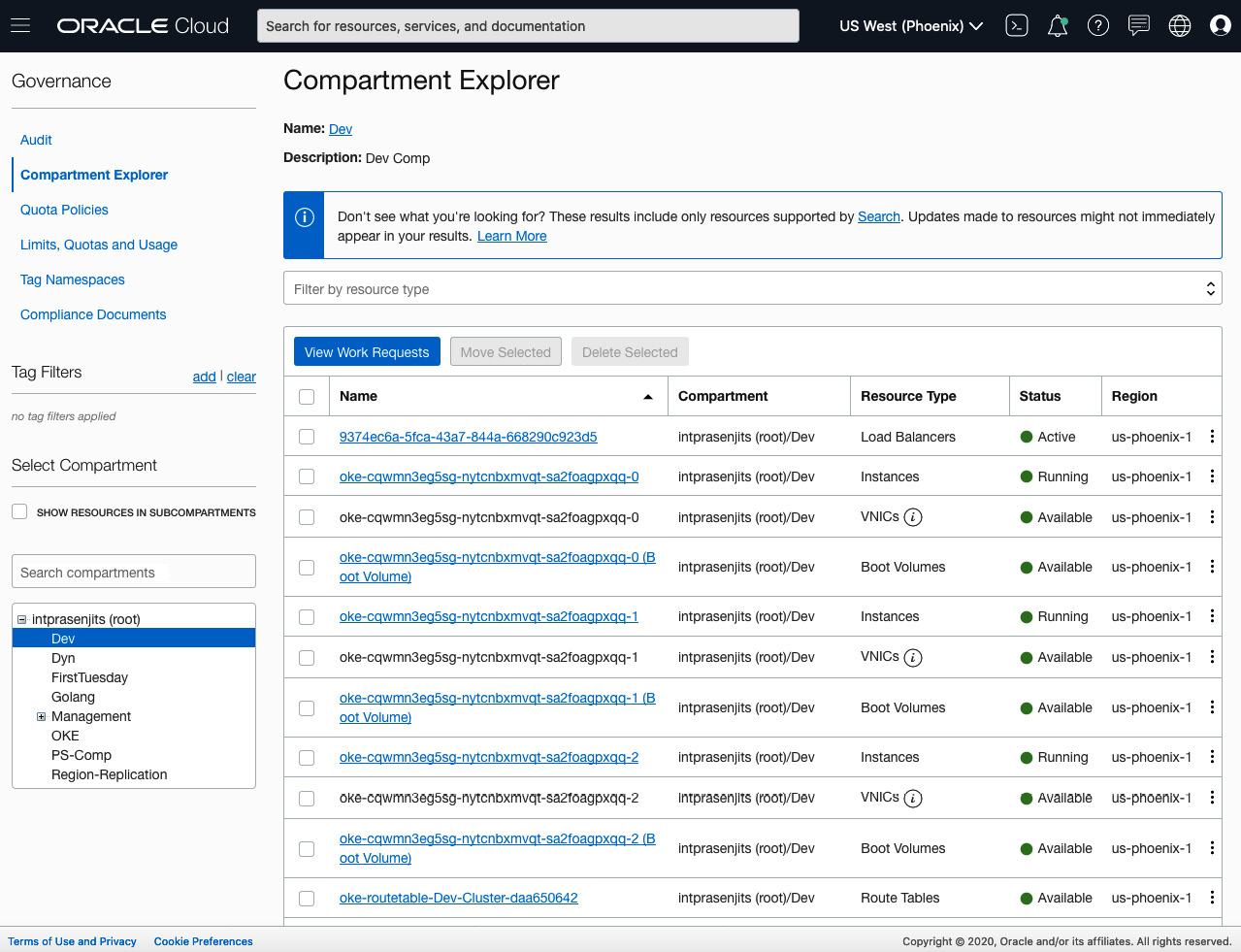


Figure 2.5 – Compartment Explorer

If you want to move a compartment to a different upper-level (that is, parent) compartment, then you need to use **Compartment Explorer**. However, keep in mind that with this movement across different compartments, you are also going to move all of the original compartment's contents to the target compartment.

However, there are restrictions when you move resources to another compartment. They are as follows:

* The name of the compartment cannot be the same when you move one compartment to another compartment. That means the parent and sub-compartment cannot have the same name.
* Additionally, you cannot use the same name in the sub-compartment when you have another sub-compartment with that name already in use. That means two sub-compartments cannot have the same name under a parent compartment.

Take a look at the following steps to learn how to move resources across compartments:

1. Log in to the OCI console.
2. Open the **Navigation** menu.
3. Under **Governance and Administration**, expand **Governance** and select **Compartment Explorer**.
4. Check which resource you want to move.
5. On the right-hand side, there is an **Action** icon, which is three dots. Click on it and select **Move Resource**, as shown in the following screenshot:

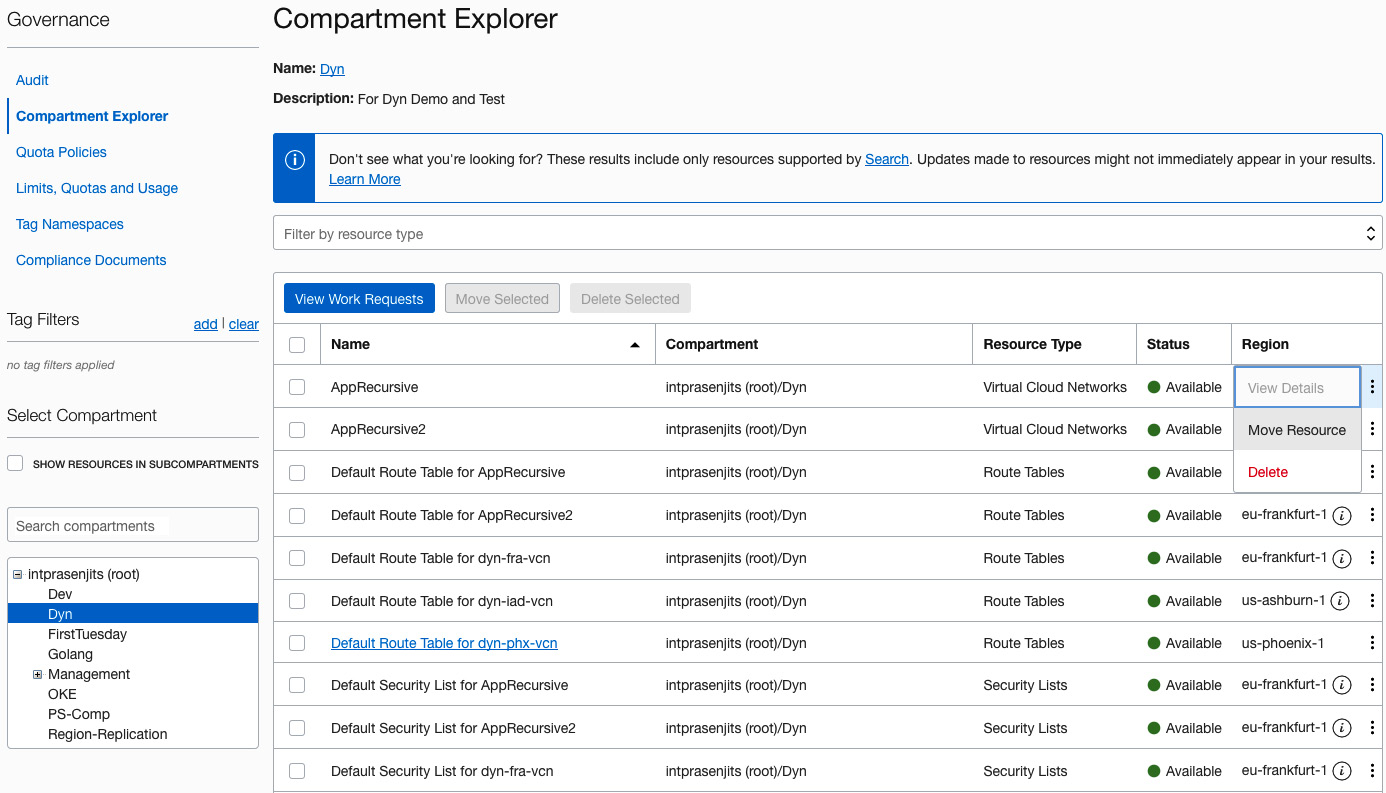


Figure 2.6 – Moving resources across compartments

1. Choose which compartment you want this resource to go to from the list. This is shown in the following screenshot:

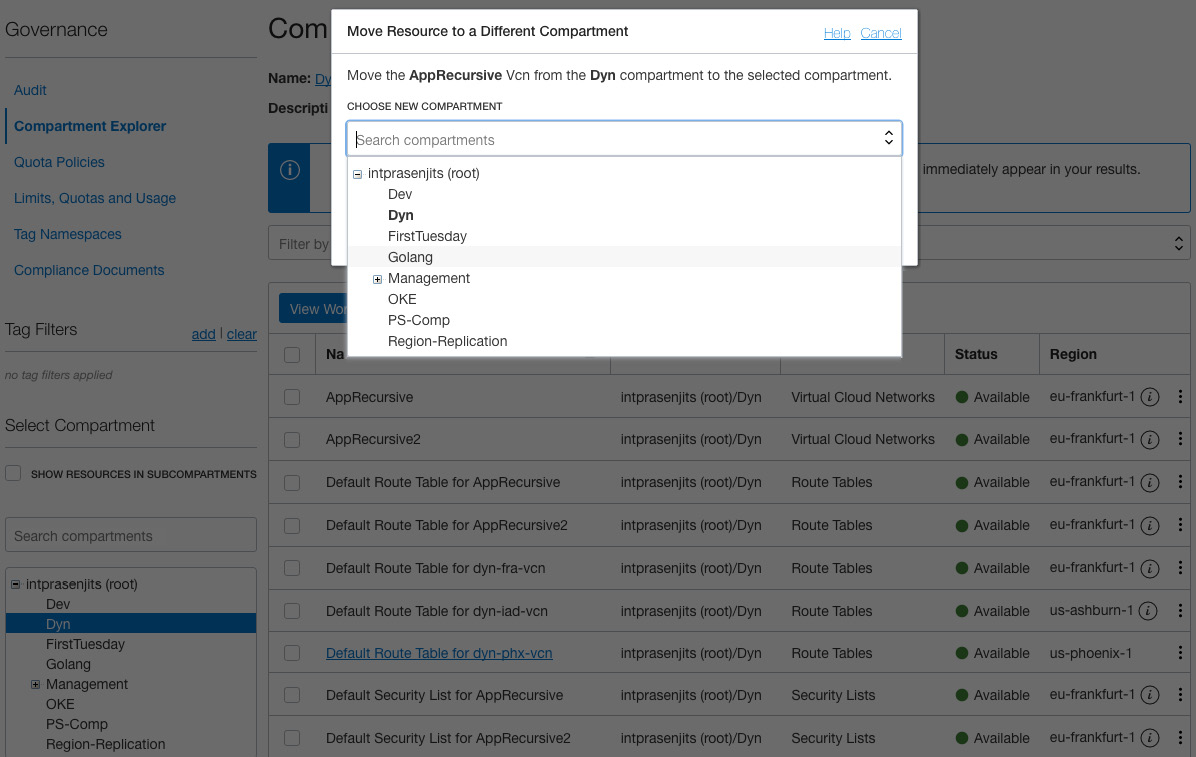


Figure 2.7 – Moving a resource to another compartment

1. Click on **Move Resource**.

In this section, you have learned how to use compartments to organize resources. In the next section, we will discuss how to secure access to the resources in these compartments using policies.

**Accessing resources from compartments using a policy**

A policy is an entity that specifies which groups can access specific resources, and in which ways. You tend to assign access at the compartment level, which indicates that all users in the group, to which the policy is assigned, can access all the resources within that compartment using the level of permission specified in the policy. Policies can also be applied at the tenancy level, and in such cases, the granted access is available to all compartments within the tenancy.

There are three requirements for a policy: an action or a verb, a resource type, and whether the policy is at the tenancy or compartment level. Furthermore, IAM allows granular policies, so they can be applied at either the aggregate level or the individual resource level. Polices can also include one more condition. Conditions such as *any* or *all* can be used. You can also use multiple conditions using logical *OR* and *AND* operators.

For conditions, you can use *any* or *all* with multiple conditions for a logical *OR* or *AND* operator, respectively.

Let's take a look at the structure of a policy statement:

Allow <subject> to <verb> <resource-type> in <location> where <conditions>

Now, let's look at each part of this policy statement in detail.

**Verbs**

Verbs can be one of the following:

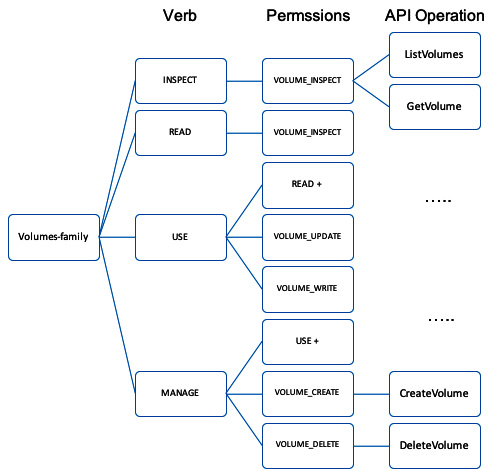


Figure 2.8 – Verbs and permissions mapping

Let's look at each of these verbs and explore what they do:

* **INSPECT**: This gives you the ability to list resources.
* **READ**: The **READ**verb also includes the **INSPECT** verb, but it also adds the ability to get user-specified metadata.
* **USE**: The **USE** verb also includes the **READ** verb, but it also adds the ability to work with existing OCI resources.
* **MANAGE**: This includes all permissions for the resource.

Permissions are the core of authorization. Permissions decide what a user can do on a given resource.

By using verbs, you can essentially simplify the process of granting access to resources. If you are sending an API request, then you need to make sure that you have access to one or more permissions.

All policy statements begin with **allow**. Therefore, by default, all access to resources within compartments is denied unless explicitly allowed by a policy statement. Here, **subject** can be a group, a comma-separated list of groups, or a special construct, such as **any-user**, which encapsulates all users within the tenancy. The **resource-type** element is the more common element of the complete list, and it currently includes approximately 12 different resource types. The **location** field of a policy statement is required, and a location can be a compartment name, a compartment ID, or tenancy, which applies to all of the compartments within the tenancy. Unlike **subject**, a comma-separated list of compartments cannot be used for the **location** field. If the same policy is to be applied to multiple compartments, multiple policy statements must be used.

The following is the list of the available resource types:

* All-resources
* Database-family
* Instance-family
* Cluster-family
* Compute-management-family
* Data-catalog-family
* DNS
* File-family
* Object-family
* Virtual-network-family
* Volume-family

Let's take a look at some of the most common policies using the verbs and resource types that we have just gone through.

In this example, we are allowing network administrators to manage a **Virtual Cloud Network** (**VCN**):

Allow group NetworkAdmins to manage virtual-network-family in tenancy

In this example, we are allowing the object writers group to write to the OCI Object Storage bucket:

Allow group ObjectWriters to manage objects in compartment ABC where any {request.permission='OBJECT\_CREATE', request.permission='OBJECT\_INSPECT'}

In this example, we are allowing OCI Block Storage and Object Storage to encrypt and decrypt volumes and buckets using customer/OCI managed keys:

Allow service blockstorage, objectstorage-<region\_name> to use keys in compartment ABC

**Policy inheritance**

Policy inheritance occurs between the parent compartment and the child compartment. For example, OCI has a built-in policy for administrators: **allow administrators to manage all-resources in tenancy**. Policy inheritance is responsible for the administrators' group being able to do anything in *any* of the compartments in the tenancy. This is depicted in the following diagram:

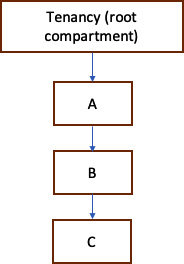


Figure 2.9 – Policy inheritance

Here, you can see that there are three levels of compartments: A, B, and C. Policies that apply to the resources in compartment A also apply to the resources in compartments B and C. If you write this policy, that is, **Allow group NewtworkAdmins to manage virtual-network-family in compartment A**, it will automatically allow the **NetworkAdmins** group to manage VCNs in compartments A, B, and C.

**Policy attachment**

You can assign a policy to a compartment or tenancy. If you have access to manage policies in the tenancy, and if the policy is attached to your tenancy, then you can change or delete it. Additionally, if you have access to a child compartment, and if anyone attaches a policy to the child compartment, then you can change or delete it.

In this section, we have learned about OCI IAM policies. In the next section, we will discuss the use of these policy statements in a more advanced IAM construct, called instance principals.

**Using instance principals to make a call to the OCI API**

Instance principals enable OCI instances to make API calls against other OCI services. Using instance principals, you can make OCI calls without the need to configure user credentials or a configuration file.

Even without instance principals, you can still achieve this by storing API credentials on each instance. However, then, you will be faced with a credential rotation problem. Additionally, auditing at the instance level is impossible since credentials are the same across hosts.

So, the ideal solution is to use instance principals that give instances their own identity. The instances that have instance principals configured become a new type of principal, and this is in addition to the existing OCI IAM user/group.

To implement an instance principal, you need to use dynamic groups, which allow policies to be defined on instances. An instance principal implements API authentication at the instance level, removing the need for any credential management.

Authorization is done via dynamic groups. You can create a principal actor by grouping OCI instances, and this is done by using dynamic groups. Here, you specify the permission to that dynamic group using an IAM-level policy. To manage the membership of this dynamic group, you need to define rules called **Matching Rules**. Resources that match the rule criteria become members of the dynamic group.

Take a look at the following diagram to understand how it actually works:

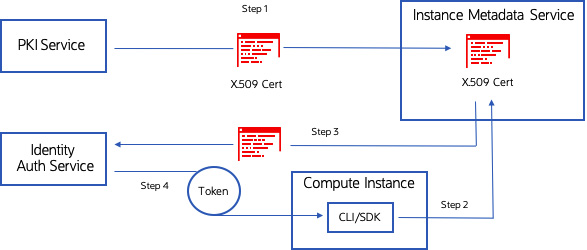


Figure 2.10 – Instance principal certificate assignment

OCI uses an internal **Public Key Infrastructure** (**PKI**) Service, which issues X.509 certificates for every compute instance. OCI's internal **Certificate Authority** (**CA**) assigns these certificates that hold information about the instance, such as the instance ID, compartment ID, and more. The OCI SDK/CLI calls the Instance Metadata Service to get this certificate, and then uses it to call the Identity Auth Service. The auth service then returns a token that this instance uses for calls to OCI APIs.

When you send OCI API calls using that token, the OCI auth service will check whether any matching policy exists. If a match is found, then it will be authorized (using the new *instances* subject).

**Creating an instance principal**

In this section, we will show you how to create an instance principal where an instance within OCI can access OCI Object Storage without any stored user credentials, private keys, or fingerprints. To create the instance principal, you need to first create a dynamic group. So, let's do that first.

To create a dynamic group, first, log in to the OCI console, which is located at<https://console.us-phoenix-1.oraclecloud.com/>:

1. Log in to the OCI console.
2. Open the **Navigation** menu.
3. Under **Identity**, select **Dynamic Groups**.
4. Click on **Create Dynamic Groups**.
5. Provide a suitable **Name** and **Description**.
6. In the **Matching Rules** section, type in the following:

ALL {instance.compartment.id = '<compartment-ocid>'}

Change **compartment-ocid** to your compartment OCID, as shown in the following screenshot:

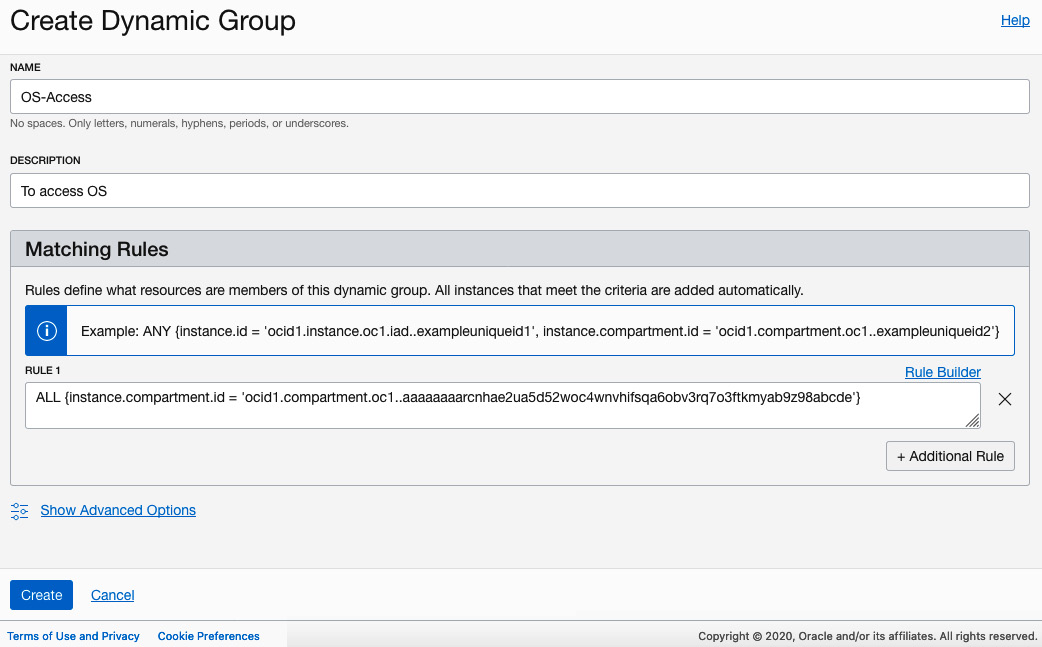


Figure 2.11 – Creating a dynamic group

1. Click on **Create**.
2. Within the **Identity** menu, click on **Policies**.
3. Click on **Create Policy**.
4. Provide a suitable **Name** and **Description**.
5. Choose the compartment that you want to attach this policy to.
6. In **Policy Statements**, write the following policies to allow the created dynamic group to manage buckets:

Allow dynamic-group OS-Access to manage buckets in tenancy

Allow dynamic-group OS-Access to manage objects in tenancy

The preceding details are shown in the following screenshot:

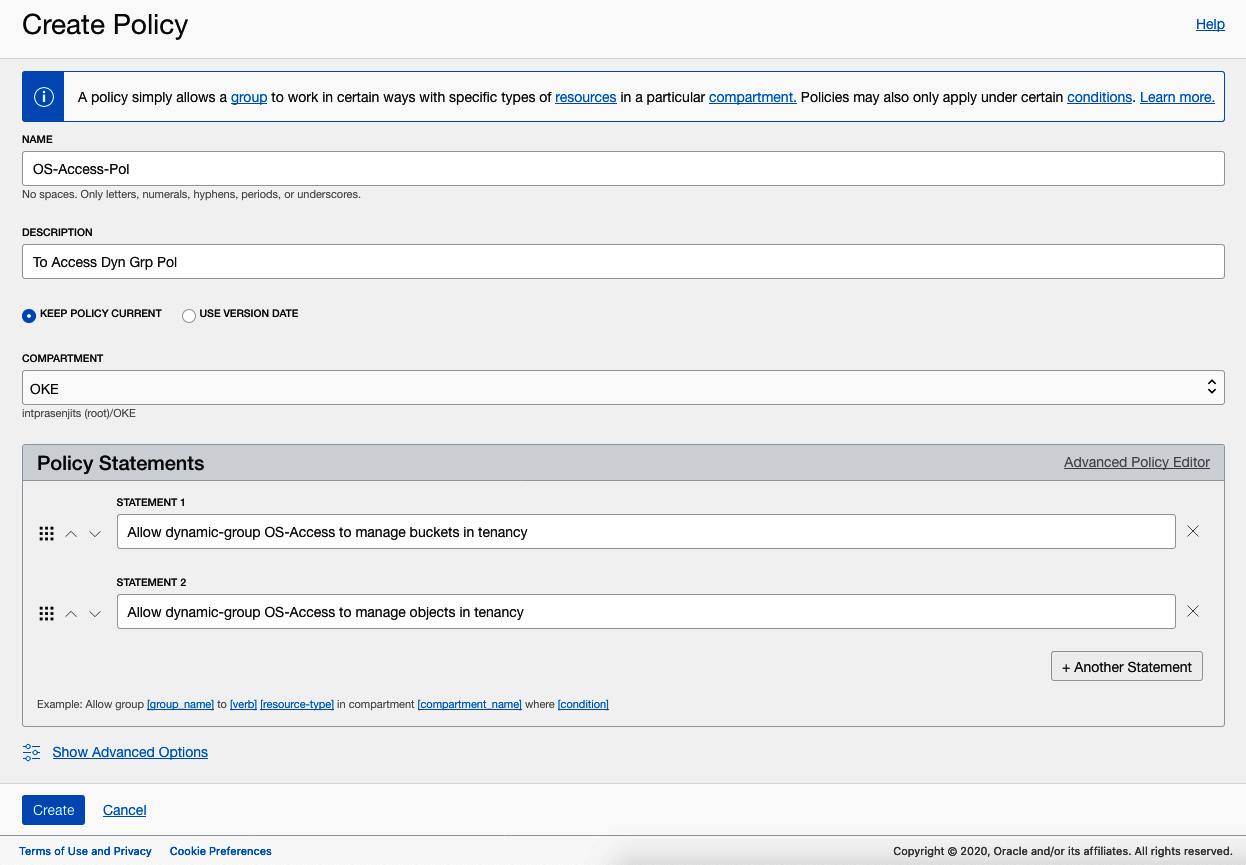


Figure 2.12 – Creating an IAM policy

1. Click on **Create**.

So, you can create a dynamic group and associate policy to give access to certain, or all, instances in a particular compartment. To test it out, you will need to create an instance and install the OCI CLI. We will go through that process in [*Chapter 4*](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9781800566460/B16798_04_Final_NM_ePub.xhtml#_idTextAnchor083), *Compute Choices on Oracle Cloud Infrastructure*, and [*Chapter 10*](https://learning.oreilly.com/library/view/oracle-cloud-infrastructure/9781800566460/B16798_10_Final_NM_ePub.xhtml#_idTextAnchor163), *Interacting with Oracle Cloud Infrastructure Using the CLI/API/SDK*.

Once you install the OCI CLI, run the following command to authenticate against the OCI Object Storage API:

$ oci os ns get --auth instance\_principal

{

  "data": "intprasenjits"

}

In this section, we have learned how to use instance principals to give access to an OCI instance to call the OCI API without carrying any credentials or configuration files. In the next section, we will discuss how you can take IAM further and bring in your own identity using federation concepts.

**Federating OCI access using a third-party IdP**

OCI's recommendation is to have a federation established between your existing IdP and OCI to manage the OCI console login. As an administrator, it's your responsibility to create the federated trust between your existing IdP and OCI IAM. Once this trust is established, you can create the mapping between on-premises groups and IAM groups. For enterprises that use custom policies for user authentication, a federation is super important.

OCI's best practice is to have a federation administrators' group. This should then be mapped to the federated IdP administrator group. The administrators' group from the federated IdP holds administrative privileges and can manage customer tenancy.

As a best practice, you should have access to the OCI-level tenancy administrator user. If a situation occurs where you break the federation, then you can always use this account to log in to the OCI console and fix the problem.

If you want to leverage **System for Cross-domain Identity Management** (**SCIM**), then you should federate your OCI tenancy with Oracle Identity Cloud Service. Federated users can use API keys and auth tokens, which we described earlier in this chapter, to authenticate against the OCI API and manage these settings from their **User Settings** page.

There are three ways to configure federations:

* Microsoft AD FS
* Any SAML 2.0-compliant IdP
* Oracle Identity Cloud Service

**Configuring a federation**

To configure a federation, you need to choose which federation you want to integrate your IAM with. In this case, we will assume that it's going to be Microsoft Active Directory. Follow these steps:

1. Log in to the OCI console.
2. Open the **Navigation** menu.
3. Under **Identity**, select **Federation**.
4. Click on **Add Identity Provider**.
5. Provide a suitable **Name** and **Description**.
6. Choose **Microsoft Active Directory Federation Service** (**AD FS**).
7. Upload the **FederationMetadata.xml** document from your AD FS server, as shown in the following screenshot:

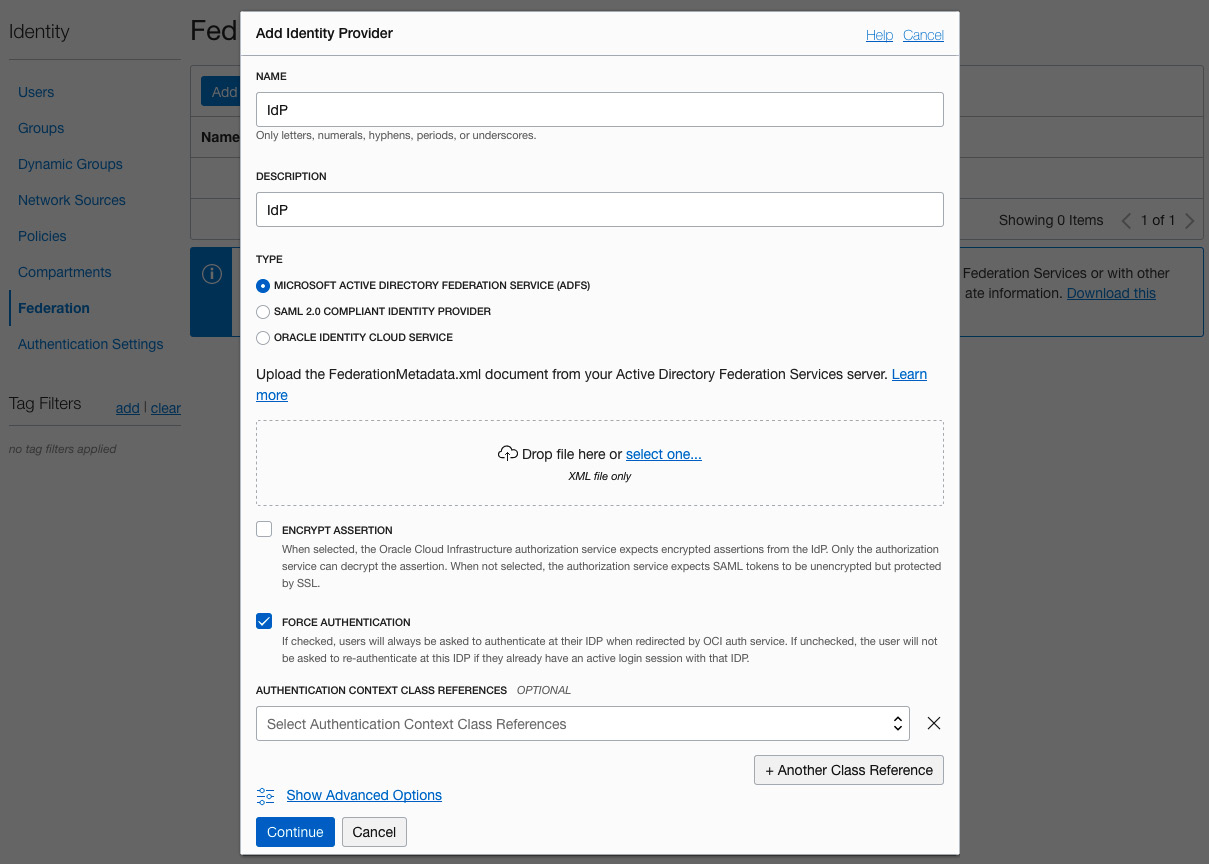


Figure 2.13 – Adding an IdP

1. Click on **Continue**.
2. At this stage, you will need to create a mapping of Active Directory resources with an IAM resource.
3. Under **Identity Provider Group**, provide the Active Directory group name. This is case-sensitive as well.
4. Choose the IAM group from the list under **OCI Group**.
5. Once your mapping is done, click on **Create**.

So, you have learned how to not only rely on OCI IAM constructs but bring in your own IdP as well, by integrating your IdP with OCI IAM using a federation.

**Summary**

In this chapter, you learned about the security fundamentals of OCI. We explained principals, compartments, policies, instance principals, and federation concepts. You have learned how to design a logical separation of resources within compartments, and then give access to certain users to perform operations on certain resources within a compartment. You have also looked at how to implement IAM to give native security access to OCI instances, so it can call the OCI API without the need to store local user credentials.

In the next chapter, we will learn how OCI implements virtual networking and

In the next chapter, we will learn how OCI implements virtual networking and some of the advanced scenarios that depict how you can connect your on-premises data center to a regional OCI data center.