ORACLE
University

Oracle Cloud Infrastructure Operations Professional: Hands-on Workshop

Activity Guide

D1102591GC10

Learn more from Oracle University at ***education.oracle.com***

1002082023

Table of Contents

Common Operational Activities: Host a Web Application

Lab 01-1 Practices

Estimated Time: 120 minutes

Get Started

Overview

In this lab, you will deploy and manage a scalable, secure, DNS-mapped front-end web application that can be accessed by the internal users of your organization—either within your virtual cloud network or on-premises infrastructure.

This lab builds up a web application in several parts. First, you'll host an Apache app on a single virtual machine (VM). Then, you'll configure the host app on a pool of autoscaling compute instances behind a load balancer. And finally, you'll add front-end SSL certification to the load balancer.

In this lab, you'll:

a. Create a VCN with gateways, route tables, security lists, and subnets.

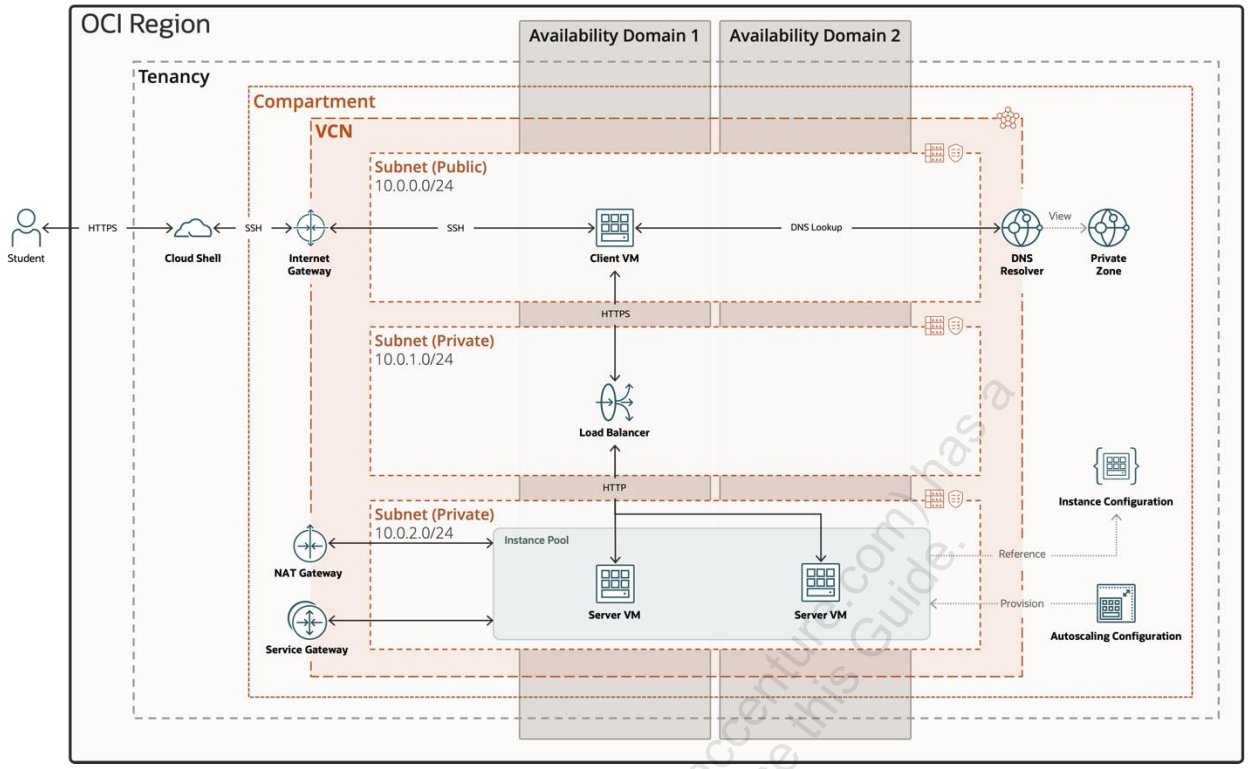
b. Create a VM that hosts an Apache application in two different ways:

   1) SSH into the machine and use Bash to configure Apache

   2) Provide a cloud-init script to execute when the VM launches

c. Create and auto-scale compute pools and attach to the load balancer.

d. Create a private DNS zone and configure front-end SSL for the load balancer.



Prerequisites

- Required policies have been set up for you.
- Set up lab environment.

Create a Virtual Cloud Network

In this first practice, you will set up a VCN and its common elements - gateways, security lists, and route tables, and subnets. While you could do this with the Cloud Console's VCN Wizard, this lab will guide you through the manual way to better understand the console and process.

Tasks

1. Sign in to the Oracle Cloud Infrastructure Cloud Console using your cloud tenant name, username, and password.

Create a VCN

2. To start, you'll create an empty VCN.

    a. From the navigation menu, under Networking, click **Virtual Cloud Networks**. Choose your compartment under **List Scope** from the left navigation pane and click **Create VCN**. A menu will pop up.

        1) For the **Name**, enter `<REGION>-OP-LAB01-1-VCN-01` (replace `<REGION>` with the appropriate region code). See the Region Codes and Availability Domains topic in the Oracle Cloud Infrastructure Registry documentation.

        2) For **Compartment,** ensure that your compartment is selected.

        3) Under **IPv4 CIDR Blocks**, add `10.0.0.0/16`
        Press **Enter** after typing the CIDR block.

        4) Leave the defaults for the other options.

        5) Click **Create VCN**. The console will create the VCN and take you to its details page.

Add Gateways to the VCN

3. To allow bi-directional traffic between the Internet and the VCN, you will add an Internet gateway. Later, you'll attach this to a public subnet.

    a. In the left navigation pane, under **Resources**, click **Internet Gateways (0)** to display the (currently empty) Internet Gateways list.

    b. Click **Create Internet Gateway**.

1) For the **Name**, enter `<REGION>-OP-LAB01-1-IG`
See the Region Codes and Availability Domains topic in the Oracle Cloud Infrastructure Registry documentation.

2) Under **Create In Compartment**, ensure that your compartment is selected.

3) At the bottom, click **Create Internet Gateway**. The Internet gateway should now be displayed in the list.

4. To allow private (egress-initiated only) traffic from the VCN into the Internet, you'll create a NAT gateway. Later, you'll attach this to a private subnet.

   a. In the left navigation pane, under **Resources**, click **NAT Gateways (0)** to display the list of NAT gateways.

   b. Click **Create NAT Gateway**.

      1) For **Name**, enter `<REGION>-OP-LAB01-1-NG`

      2) Under **Create In Compartment**, ensure that your compartment is selected.

      3) Leave **Ephemeral Public IP Address** selected.

      4) Click **Create NAT Gateway**. The NAT gateway should now be displayed in the list.

5. To allow traffic between the VCN and Oracle services, you will add a service gateway. Later, you'll attach this to a private subnet. Strictly speaking, this gateway may not be needed for this lab. However, it will come in handy if you need to use the Bastion service to connect to private instances and debug.

   a. In the left navigation pane, under **Resources**, click **Service Gateways (0)** to show the list of service gateways.

   b. Click **Create Service Gateway.**

      1) For **Name**, enter `<REGION>-OP-LAB01-1-SG`

      2) Under **Create In Compartment**, ensure that your compartment is selected.

      3) Under **Services**, select **ALL *<REGION>* Services in Oracle Services Network**, where ***<REGION>*** will be replaced by the appropriate region code.

      4) Click **Create Service Gateway**. It should now be displayed in the list.

Add Route Tables to the VCN

6. To direct traffic to these gateways, you need to create route tables to point to them.

   a. In the left navigation pane, under **Resources**, click **Route Tables (1)** to display a list of route tables. There should currently be a default route table with no rules.

   b. First, you'll create one for public Internet access. Click **Create Route Table**.

      1) For **Name**, `<REGION>-OP-LAB01-1-RT-PUBLIC`

      2) Under **Route Rules (Optional)**, click **+ Another Route Rule**.

         a) For **Target Type**, select **Internet Gateway.**

         b) For **Destination CIDR Block**, type `0.0.0.0/0`

         c) Under **Compartment**, ensure that your compartment is selected.

         d) Under **Target Internet Gateway**, select the internet gateway you created earlier. This should be named `<REGION>-OP-LAB01-1-IG`

      3) Click **Create**. The route table should now show in the list and have one route rule.

   c. Next, you'll create one for private Internet and Oracle services access. Click **Create Route Table.**

      1) For **Name**, `<REGION>-OP-LAB01-1-RT-PRIVATE`

      2) To add a rule for the NAT gateway, under **Route Rules (Optional)**, click **+ Another Route Rule**.

         a) For **Target Type**, select **NAT Gateway.**

         b) For **Destination CIDR Block**, type `0.0.0.0/0`

         c) Under **Compartment**, ensure that your compartment is selected.

         d) Under **Target NAT Gateway**, select the NAT gateway you created earlier. This should be named `<REGION>-OP-LAB01-1-NG`

      3) To add a second rule, for the service gateway, click **+ Another Route Rule**

         a) For **Target Type**, select **Service Gateway.**

b) For **Destination Services**, select **ALL *<REGION>* Services in Oracle Services Network**, where ***<REGION>*** will be replaced by the appropriate region code.

c) Under **Compartment**, ensure that your compartment is selected.

d) Under **Target Service Gateway**, select the service gateway you created earlier. This should be named `<REGION>-OP-LAB01-1-SG`

4) Click **Create**. The route table should now show in the list and have two route rules.

Add Security Lists to the VCN

7. In order for subnets to accept traffic from these gateways, you'll need to create security lists that specify allowed traffic.

   a. In the left navigation pane, under **Resources**, click **Security Lists (1)** to display a list of security lists. There should currently be a default security list that allows SSH and certain ICMP traffic.

   b. First, you'll create a security list for public Internet traffic. Click **Create Security List**.

      1) For **Name**, enter `<REGION>-OP-LAB01-1-SL-PUBLIC`

      2) Under **Compartment**, ensure that your compartment is selected.

      3) Under **Allow Rules for Ingress**, click **+ Another Ingress Rule**. A form for **Ingress Rule 1** should appear.

         a) Leave **Stateless** deselected.

         b) For **Source Type**, leave CIDR selected.

         c) For **Source CIDR**, enter `0.0.0.0/0` to allow traffic from the Internet.

         d) For **IP Protocol**, leave TCP selected.

         e) Leave **Source Port Range** empty.

         f) For **Destination Port Range** enter `80,443` to allow HTTP and HTTPS traffic.

      4) Click **Create Security List**. It should now appear in the list.

c. Next, you'll create another security list for private traffic. Click **Create Security List**.

1) For **Name**, enter `<REGION>-OP-LAB01-1-SL-PRIVATE`

2) Under **Allow Rules for Ingress**, click **+ Another Ingress Rule**. A form for **Ingress Rule 1** should appear.

a) Leave **Stateless** deselected.

b) For **Source Type**, leave CIDR selected.

c) For **Source CIDR**, enter `10.0.0.0/16` to limit traffic to that originating in the VCN.

d) For **IP Protocol**, leave TCP selected.

e) Leave **Source Port Range** empty.

f) For **Destination Port Range** enter `80,443` to allow HTTP and HTTPS traffic.

3) Click **Create Security List**. It should now appear in the list.

Add Subnets to the VCN

8. Next, you'll add three subnets to the VCN: one public and two private. In the left navigation pane, click **Subnets (0)** to display the list of subnets. First, you'll create the public subnet. Click **Create Subnet**.

1) For **Name**, enter `<REGION>-OP-LAB01-1-SNT-PUBLIC`

2) Under **Compartment**, ensure that your compartment is selected.

3) For **Subnet Type**, select **Regional (Recommended).**

4) For **CIDR Block**, enter `10.0.0.0/24`

5) For **Route Table**, select the route table you created earlier for public access. This should be named `<REGION>-OP-LAB01-1-RT-PUBLIC`

6) For **Subnet Access**, select **Public Subnet.**

7) Leave **Use DNS hostnames in this SUBNET** selected.

8) For **DNS Label**, enter `SNTPUBLIC`

9) For the DHCP options, select the default.

10) Under the box **Security Lists**, you will attach two security lists.

    a) First, select the default security list in the drop-down to allow SSH and ICMP traffic.

    b) Second, click **+ Another Security List** to add another drop-down list.

    c) Under the new drop-down list, select the security list you created earlier for public HTTP(S) access. This should be named `<REGION>-OP-LAB01-1-SL-PUBLIC`

11) Click **Create Subnet**. The subnet should appear in the list.

9. Click **Create Subnet** again to create a private subnet. This subnet will be used for a load balancer you will create later.

    1) For **Name**, enter `<REGION>-OP-LAB01-1-SNT-PRIVATE1`

    2) Under **Compartment**, ensure that your compartment is selected.

    3) For **Subnet Type**, select **Regional (Recommended).**

    4) For **CIDR Block**, enter `10.0.1.0/24`

    5) For **Route Table**, select the route table you created earlier for private access. This should be named `<REGION>-OP-LAB01-1-RT-PRIVATE`

    6) For **Subnet Access**, select **Private Subnet.**

    7) Leave **Use DNS hostnames in this SUBNET** selected.

    8) For **DNS Label**, enter `SNTPRIVATE1`

    9) For the DHCP options, select the default.

    10) Under the box **Security Lists**, you will attach two security lists.

        a) First, select the default security list in the drop-down list to allow SSH and ICMP traffic.

        b) Second, click **+ Another Security List** to add another drop-down list.

c) Under the new drop-down list, select the security list you created earlier for intra-VCN HTTP(S) access. This should be named `<REGION>-OP-LAB01-1-SL-PRIVATE`

11) Click **Create Subnet**. The subnet should appear in the list.

10. Click **Create Subnet** again to create a second private subnet. This will be used for an instance pool you will create later.

1) For **Name**, enter `<REGION>-OP-LAB01-1-SNT-PRIVATE2`

2) Under **Compartment**, ensure that your compartment is selected.

3) For **Subnet Type**, select **Regional (Recommended).**

4) For **CIDR Block**, enter `10.0.2.0/24`

5) For **Route Table**, select the route table you created earlier for private access. This should be named `<REGION>-OP-LAB01-1-RT-PRIVATE`

6) For **Subnet Access**, select **Private Subnet.**

7) Leave **Use DNS hostnames in this SUBNET** selected.

8) For **DNS Label**, enter `SNTPRIVATE2`

9) For the DHCP options, select the default.

10) Under the box **Security Lists**, you will attach two security lists.

a) First, select the default security list in the drop-down list to allow SSH and ICMP traffic.

b) Second, click **+ Another Security List** to add another drop-down list.

c) Under the new drop-down list, select the security list you created earlier for intra-VCN HTTP(S) access. This should be named `<REGION>-OP-LAB01-1-SL-PRIVATE`

11) Click **Create Subnet**. The subnet should appear in the list.

You should now have a VCN with one public subnet and two private subnets. The public subnet is attached to an Internet gateway; the private subnets are attached to a NAT gateway and a service gateway. All the subnets allow SSH and ICMP traffic due to the default security list. The public subnet also allows HTTP(S) traffic from anywhere. The private subnets allow HTTP(S) traffic originating from within the VCN.

Create SSH Keys for Cloud Shell

Note: If your Cloud Shell VM already has an SSH key pair and you have downloaded the public key, you can skip to creating the compute instances.

Tasks

1. Open Cloud Shell using the icon on the top-right corner.
2. Use the following command in Cloud Shell to generate an SSH key pair (remember not to include the `$`):

   ```
   $ ssh-keygen -t rsa -b 4096
   ```

3. It will prompt you for the location to save the files. Leave it empty and press **Enter** to use the default.
4. It will prompt you for a password. For this lab, leave it empty and press **Enter** to skip. Press **Enter** again to confirm.
5. Add your keys to the ssh-agent with the following two commands:

   ```
   $ eval "$(ssh-agent -s)"
   $ ssh-add ~/.ssh/id_rsa
   ```

6. Download the public key:

   a. Click the gear on the top right of Cloud Shell.

   b. Click **Download**.

   c. Enter `.ssh/id_rsa.pub` to specify the public key. Click **Download** to confirm. This is the key you will upload to compute instances when provisioning them.

7. Minimize the Cloud Shell.

Set Up a Webserver with SSH

In this practice, you will add a VM instance to the public subnet in the VCN, then use SSH in Cloud Shell to install and configure an Apache webserver.

Tasks

Create a VM

1. First, you'll create a VM in your public subnet.

    a. In the navigation menu, click **Compute**, then **Instances** to navigate the list of compute instances.

    b. Under **Compartment** on the left, ensure that your compartment is selected.

    c. Click **Create instance** to open the instance creation form.

        1) For **Name**, enter `<REGION>-OP-LAB01-1-VM-01`. (Remember to replace `<REGION>` with the appropriate region code. See the Region Codes and Availability Domains topic in the Oracle Cloud Infrastructure Registry documentation.)

        2) In the **Placement** box, select **AD 1** for the first availability domain.

        3) In the **Image and Shape** box:

            a) Leave the **Image** as **Oracle Linux 8**.

            b) Click **Change shape** to open the shape configuration dialog box.

                i) For **Instance type**, leave **Virtual Machine** selected.

                ii) Change the **Shape series** to **Ampere.**

                iii) Select **VM.Standard.A1.Flex.**

                iv) Leave **Number of OCPUs** at one.

                v) Leave **Amount of memory (GB)** at six.

                vi) Click **Select shape** to confirm the shape configuration.

        4) In the **Networking** box:

a) Under **Primary Network**, choose **Select existing virtual cloud network.**

b) Under **Virtual cloud network**, select the VCN you created earlier. It should be named `<REGION>-OP-LAB01-1-VCN-01`

c) Under **Subnet**, choose **Select existing subnet** and choose the public subnet you created earlier. It should be named `<REGION>-OP-LAB01-1-SNT-PUBLIC`

d) Leave **Assign a public IPv4 address** selected.

5) Under the **Add SSH keys** box, upload the SSH keys generated earlier or paste your Cloud Shell machine's public key.

6) Leave the **Boot volume** options as the default.

d. Click **Create**. The console will take you to the instance's details page.

2. Wait for the instance to finish provisioning to view its details.

a. Under **Instance Access**, copy the **Public IP address**.

Install and Configure Apache HTTP Server

3. Next, you will use Cloud Shell to `SSH` into the VM and install Apache.

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

```
$ ssh opc@<public_ip_address>
```

If you get a `Connection refused` error, wait a minute before trying again. It's possible for the VM to show it is *RUNNING* but `sshd` hasn't finished booting.

If you get the error that your key has permissions that are too open, use this to update permissions:

```
$ chmod 600 <private_key>
```

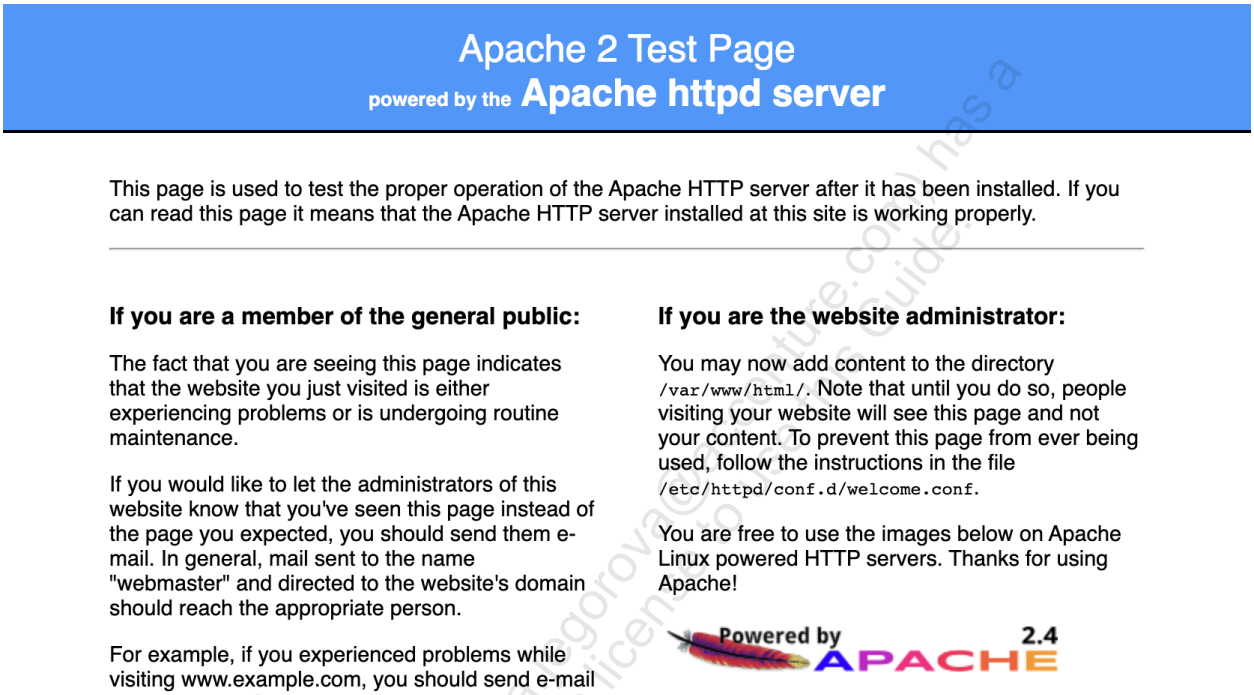
b. Install Apache webserver (`httpd`), configure it to start whenever the VM boots, and activate it.

```
$ sudo dnf install httpd
$ sudo systemctl enable httpd
$ sudo systemctl start httpd
```

c. Open the OS firewall to HTTP.

```
$ sudo firewall-cmd --permanent --add-service=http
$ sudo firewall-cmd --reload
```

4. To verify that the webserver is running, enter its public IP address into your browser's address bar. It might take a few minutes after the instance says it is running. Apache's default page should display.



5. Now that you have seen the process of manually starting an Apache webserver, you can do it through cloud-init automation instead. Terminate the above instance.

   a. You should still be on the instance's details page. If not, then navigate to it:

      1) From the navigation menu, click **Instances** under Compute.

      2) Ensure that your compartment is selected on the left panel.

      3) In the table, click your compute instance's name.

   b. Click **Terminate**. Select **Permanently delete the attached boot volume**. Click **Terminate instance** to confirm.

Set Up Webserver with Cloud-init

For this practice, you'll create a second VM instance. This time, you'll use a cloud-init script instead of using SSH to install and activate an Apache webserver with the same firewall settings.

Tasks

Create a VM

1. Navigate to the compute instances list:
    a. If you are still in the instance details page from the terminated instance, click **Instances** in the breadcrumbs.
    b. Otherwise, in the navigation menu, click **Compute**, then **Instances** to navigate the list of compute instances. Under **Compartment** on the left, ensure that your compartment is selected.
2. Click **Create instance** to open the instance creation form.
    a. For **Name**, enter `<REGION>-OP-LAB01-1-VM-02`. (Remember to replace `<REGION>` with the appropriate region code. See the Region Codes and Availability Domains topic in the Oracle Cloud Infrastructure Registry documentation.)
    b. In the **Placement** box, select **AD 1** for the first availability domain.
    c. In the **Image and Shape** box:
        i. Leave the **Image** as **Oracle Linux 8.**
        ii. Click **Change shape** to open the shape configuration dialog box.
            1. For **Instance type**, leave **Virtual Machine** selected.
            2. Change the **Shape series** to **Ampere.**
            3. Select **VM.Standard.A1.Flex.**
            4. Leave **Number of OCPUs** at one.
            5. Leave **Amount of memory (GB)** at six.
            6. Click **Select shape** to confirm the shape configuration.

d. In the **Networking** box:

   i. Under **Primary Network**, choose **Select existing virtual cloud network.**

   ii. Under **Virtual cloud network**, select the VCN you created earlier. It should be named `<REGION>-OP-LAB01-1-VCN-01`

   iii. Under **Subnet**, choose **Select existing subnet** and choose the public subnet you created earlier. It should be named `<REGION>-OP-LAB01-1-SNT-PUBLIC`

   iv. Leave **Assign a public IPv4 address** selected.

e. Under the **Add SSH keys** box, upload or paste your Cloud Shell machine's public key. For more information on this, see the Appendix at the end of this lab.

f. Leave the **Boot volume** options as the default.

g. Click **Show advanced options.**

   i. Under the **Management** tab, choose **Paste cloud-init script**.

   ii. Type the following commands:

```
#!/bin/sh
sudo dnf install httpd --assumeyes --quiet
sudo systemctl enable httpd
sudo systemctl start httpd
sudo firewall-offline-cmd --add-service=http
systemctl restart firewalld
echo '<!doctype html><html><body><h1>This is
myoracle.com!</h1></body></html>' | sudo tee /var/www/html/
index.html
```

   Note: There are a couple differences between these commands and the ones we used through `SSH`:

   - We added two flags for `dnf` to (1) assume we answer "yes" when it asks whether to install httpd and dependencies and (2) limit logging.
   - We used `firewall-offline-cmd` and `systemctl` instead of `firewall-cmd` because `firewalld` is not fully online during the boot process.
   - We also added a line at the end to pass a basic webpage to Apache. This is to suppress warnings when we add a load balancer later.

c. Click **Create instance**. This will take you to the instance's details page.

3. To verify that the webserver is running, enter its public IP address into your browser's address bar. It might take a few minutes after the instance says it is running. You should get the following reply from the webserver:

```
This is myoracle.com!
```

Now that you have manually provisioned a compute instance and automatically configured it through cloud-init, you can fully automate provisioning behind a load balancer.

4. Terminate the above instance.

   a. You should still be on the instance's details page. If not, then navigate to it:

      1) From the navigation menu, click **Instances** under Compute.

      2) Ensure that your compartment is selected on the left panel.

      3) In the table, click your compute instance's name.

   b. Click **Terminate**. Select **Permanently delete the attached boot volume**. Click **Terminate instance** to confirm.

Create Private Load Balancer and Autoscale Compute Pools

In this practice, you'll create a private load balancer and distribute the web traffic to an automatically scaling pool of VMs. You will create an instance configuration and pool based on an instance. You'll then create an autoscaling configuration that will add an instance to that pool when a CPU threshold is hit.

Tasks

Provision a Client Machine

For this part of the lab, you will place the load balancer and Apache hosts in private subnets. To access them, you will SSH into a machine in the public subnet that will then act as the HTTP client. It will use `curl` instead of a browser to retrieve the webpage.

1. In the navigation menu, click **Compute**, then **Instances** to navigate the list of compute instances.
2. Under **Compartment** on the left, ensure that your compartment is selected.
3. Click **Create instance** to open the instance creation form.
    a. For **Name**, enter `<REGION>-OP-LAB01-1-VM-CLIENT`. (Remember to replace `<REGION>` with the appropriate region code. See the Region Codes and Availability Domains topic in the Oracle Cloud Infrastructure Registry documentation.)
    b. In the **Placement** box, select **AD 1** for the first availability domain.
    c. In the Image and Shape box:
        1) Leave the Image as **Oracle Linux 8**.
        2) Click **Change shape** to open the shape configuration dialog box.
        3) For Instance type, leave **Virtual Machine** selected.
        4) Change the Shape series to **Ampere**.
        5) Select **VM.Standard.A1.Flex**
        6) Leave **Number of OCPUs** at one.
        7) Leave **Amount of memory (GB)** at six.

8) Click **Select shape** to confirm the shape configuration.

d. In the **Networking** box:

1) Under Primary Network, choose **Select existing virtual cloud network**.

2) Under **Virtual cloud network**, select the VCN you created earlier. It should be named `<REGION>-OP-LAB01-1-VCN-01`.

3) Under **Subnet**, choose **Select existing subnet** and choose the public subnet you created earlier. It should be named `<REGION>-OP-LAB01-1-SNT-PUBLIC`.

4) Leave **Assign a public IPv4 address** selected.

e. Under the **Add SSH keys** box, upload or paste your Cloud Shell machine's public key.

f. Leave the **Boot volume** options as the default.

g. Click **Create**. The console will take you to the instance's details page.

4. Wait for the instance to finish provisioning to view its details.

a. Under **Instance Access**, copy the **Public IP address**.

Provision a Load Balancer

5. Navigate to the load balancers list:

a. In the navigation menu, navigate to **Networking**, then **Load Balancers**.

b. Under **Compartment** on the left, ensure that your compartment is selected.

6. Click **Create Load Balancer** at the top of the table. It will open a dialog box and ask you to select a load balancer type. Leave **Load Balancer** selected. Click **Create Load Balancer** at the bottom of the dialog box. This will take you to the load balancer creation form.

a. The first page of the form will be for configuring the load balancer itself.

1) For the **Name**, enter `<REGION>-OP-LAB01-1-LB-01`.

2) For **Visibility Type**, select **Private**.

3) Leave the **Bandwidth** section as the default.

4) Leave **Enable IPv6 Address Assignment** deselected.

5) In the **Choose Networking** box:

   a) Select the VCN you created earlier. This should be named `<REGION>-OP-LAB01-1-VCN-01`

   b) Select the first private subnet you created earlier. This should be named `<REGION>-OP-LAB01-1-SNT-PRIVATE1`

6) Click **Next** to go to the next page.

b. The second page of the form will be for configuring the backends of the load balancer.

1) Specify **Weighted Round Robin** for the **Load Balancing Policy.**

2) Skip **Add Backends** since you'll add a backend later.

3) Leave the box for **Specify Health Check Policy** as the default.

4) Leave **Use SSL** deselected.

5) Click **Show Advanced Options.** For the **Backend Set Name**, enter `<REGION>-OP-LAB01-1-LBBS-01`

6) Click **Next** to go to the next page.

c. The third page of the form will be for configuring the listener.

1) For **Listener Name**, enter `<REGION>-OP-LAB01-1-LSN-01`

2) Change **Listener traffic type** to **HTTP.**

3) Leave the listener's port at 80.

4) Click **Next** to go to the next page.

d. The fourth page will be to configure logging.

1) Disable **Error Logs.**

2) Leave **Access Logs** disabled.

3) Click **Submit**. This will take you to the load balancer's details page. Ignore the high-risk warning from Smart Check.

Create an Instance Configuration

7. Next, you will create an instance configuration to create the instance pool that you will attach to the load balancer. Navigate to the instance configuration list:

    a. In the navigation menu, click **Compute**, then **Instance Configurations.**

    b. Under **Compartment** on the left, ensure that your compartment is selected.

8. Click **Create instance configuration**.

    a. For **Name**, enter `<REGION>-OP-LAB01-1-INST-CF`

    b. For **Create in compartment**, leave your compartment selected.

    c. For **Compartment to create instances in**, leave your compartment selected.

    d. In the **Placement** box, select **AD 2** for the second availability domain.

    e. In the **Image and Shape** box:

        1) Leave the **Image** as **Oracle Linux 8.**

        2) Click **Change shape** to open the shape configuration dialog box.

            a) For **Instance type**, leave **Virtual Machine** selected.

            b) Change the **Shape series** to **Ampere.**

            c) Select **VM.Standard.A1.Flex.**

            d) Leave **Number of OCPUs** at one.

            e) Leave **Amount of memory (GB)** at six.

            f) Click **Select shape** to confirm the shape configuration.

    f. In the **Networking** box:

        1) Under Primary Network, choose **Select existing virtual cloud network**.

        2) Under **Virtual cloud network**, select the VCN you created earlier. It should be named `<REGION>-OP-LAB01-1-VCN-01`

3) Under **Subnet**, choose **Select existing subnet** and choose the second private subnet you created earlier. It should be named `<REGION>-OP-LAB01-1-SNT-PRIVATE2`

4) **Do not assign a public IPv4 address** should now be selected.

g. Under the **Add SSH keys** box, upload or paste your Cloud Shell machine's public key.

h. Leave the **Boot volume** options as the default.

i. Skip **Block volumes**.

j. Click **Show advanced options.**

1) Under the **Management** tab, choose **Paste cloud-init script**.

2) Copy the following commands:

```
#!/bin/sh
sudo dnf install httpd --assumeyes --quiet
sudo systemctl enable httpd
sudo systemctl start httpd
sudo firewall-offline-cmd --add-service=http
systemctl restart firewalld
echo '<!doctype html><html><body><h1>This is myoracle.com!
</h1></body></html>' | sudo tee /var/www/html/index.html
```

k. Click **Create**. This will take you to the instance configuration's details page.

Create an Instance Pool

9. Now, you can create an instance pool based on this instance configuration. Navigate to instance pool list. In the breadcrumbs menu, click **Compute**. From the left navigation pane, click **Instance Pools**. Click **Create instance pool**. This will take you to the instance pool creation form.

a. The first page will be for basic details.

1) For **Name**, enter `<REGION>-OP-LAB01-1-INST-PL`

2) Ensure that your compartment is selected.

3) For **Instance configuration**, select the instance configuration you just created (`<REGION>-OP-LAB01-1-INST-CF`)

4) Set the number of instances to **1**.

5) Click **Next**.

b. The second page will be for instance placement.

1) Under **Availability domain selection 1**:

a) Select **AD 1** for the **Availability domain**.

b) Leave **Fault domains** as is.

c) Select your VCN (*<REGION>*-OP-LAB01-1-VCN-01).

d) Select the second private subnet (*<REGION>*-OP-LAB01-1-SNT-PRIVATE2).

2) Click **+ Another availability domain**.

a) Select **AD 2** for the **Availability domain**.

b) Leave **Fault domains** as is.

c) Select your VCN (*<REGION>*-OP-LAB01-1-VCN-01).

d) Select the second private subnet (*<REGION>*-OP-LAB01-1-SNT-PRIVATE2).

3) Select **Attach a load balancer**.

4) Under the **Load balancer 1** box:

a) Leave **Load balancer** as the **Load balancer type.**

b) Select the load balancer you created earlier (*<REGION>*-OP-LAB01-1-LB-01).

c) Select the load balancer's backend set (*<REGION>*-OP-LAB01-1-LBBS-01).

d) For **Port**, enter 80.

e) For **VNIC**, select **Primary VNIC.**

5) Click **Next**.

c. Review your configuration and click **Create**. This will take you to the instance pool's details page.

Create an Autoscaling Configuration

10. You now have a load balancer with an instance pool in its backend. To set the instance pool to automatically scale, you'll need to create an autoscaling configuration. Navigate to the autoscaling configuration list:

    a. In the breadcrumbs menu, click **Compute**.

    b. From the left navigation pane, click **Autoscaling Configurations**.

11. Click **Create autoscaling configuration**. This will take you to the autoscaling configuration creation form.

    a. The first page will be for basic details.

        1) For **Name**, enter `<REGION>-OP-LAB01-1-AS-CONFIG`

        2) Choose your compartment.

        3) Select your instance pool (`<REGION>-OP-LAB01-1-INST-PL`). It may take a few minutes for it to appear.

        4) Click **Next**.

    b. On the **Configure autoscaling policy** page:

        1) Ensure that **Metric-based autoscaling** is selected.

        2) In the **Configure autoscaling policy** box:

            a) Name the policy `<REGION>-OP-LAB01-1-AS-POL`

            b) Leave the **Cooldown in Seconds** as 300. This is the minimum period of time between scaling actions.

            c) Select **CPU utilization** for **Performance Metric**.

            d) Enter the following for the **Scale-out rule**:

                i) **Scale-out Operator:** Greater than (>)

                ii) **Threshold percentage**: 75

iii) **Number of instances to add**: 1

e) Enter the following for the **Scale-in rule**:

i) **Scale-in operator**: Less than (<)

ii) **Threshold percentage**: 25

iii) **Number of instances to remove**: 1

f) Enter the following for the **Scaling limits**:

i) **Minimum number of instances**: 1

ii) **Maximum number of instances**: 2

iii) **Initial number of instances**: 1

3) Click **Next**.

c. Review your configuration and click **Create**. This will take you to the autoscaling configuration's details page.

12. Navigate back to your instance pool's details page.

a. Click **Compute** in the breadcrumbs menu and select **Instance Pools** in the left navigation pane.

b. Wait for your instance pool to change state from Scaling to Running.

c. Click the pool's name (`<REGION>-OP-LAB01-1-INST-PL`) to go to its details page.

13. Here, you will be able to view the instance pool's instances. On the left navigation pane under **Resources**, select **Attached instances**. You should see one instance listed in the table.

Test the Autoscaling Configuration

14. Next, you will simulate high load on the instance pool to test the autoscaling configuration. To simulate load, you will `SSH` into the machine and run a common Linux stress-testing tool, `stress-ng`.

Note: Recall that the instance pool is in a private subnet. This means that we cannot access it directly. Instead, we will use the VM in the public subnet as both a bastion to SSH

into the private instances *and* an HTTP client to test Apache on the private instances. For the rest of the document, we will refer to the instance in the public subnet as the *client* and the instance in the private subnet as the *server*.

a. Open Cloud Shell by clicking its icon at the top-right corner.

b. `SSH` into the server through the client with the following command. The inner proxy command connects to the public VM, then the outer command connects to the private VM. Replace the angle bracket labels with the appropriate content. (Remember not to include the `$` when you paste the commands).

```
$ ssh -o ProxyCommand="ssh -W %h:%p -p 22 opc@<client-public-ip>" -p 22 opc@<server-private-ip>
```

c. Enter `yes` twice to confirm that you want to SSH into both machines.

d. Run the following commands in succession to make the CPU perform matrix multiplication on all cores (only one in this instance) for 15 minutes.

```
$ sudo dnf install stress-ng --assumeyes
$ stress-ng --matrix 0 -t 15m > /dev/null 2>&1 &
$ top
```

Note: The code `> /dev/null 2>&1 &` suppresses the output and makes stress-ng run in the background. This way, you can view the running process with `top`.

15. Click **Compute** in the breadcrumbs menu at the top-left corner to also view the CPU utilization and Memory utilization graphs.

16. Wait 5-10 minutes for the total instance count to increase. If you go to the **Instance Pools** area in **Compute**, you'll see a second instance has been created and your instance pool is in a Scaling state.

17. If you are still viewing top in Cloud Shell, press `q` to quit, then type `exit` to end the `SSH` session.

18. To test the webservers behind the load balancer. Log in to your client VM using `SSH` in Cloud Shell and use `curl` to retrieve the home page:

```
$ ssh opc@<client-public-ip-address>
$ curl http://<private-load-balancer-ip-address>
```

You should get the webpage in response:

```
<!doctype html><html><body><h1>This is myoracle.com!</h1></body></html>
```

Create a Private Zone Using DNS Management

As demonstrated in the previous sections of the lab, using IP addresses can be rather cumbersome. In this practice, you'll add a private Domain Name System (DNS) zone to your VCN to use hostnames instead of IP addresses.

Create a Private Zone

1. Open the navigation menu and click **Networking**. Under **DNS Management**, click **Zones**.
2. Click the **Private Zones** tab. You should see zones for your existing subnets.
3. Click **Create Zone** to open the zone creation dialog box.
    a. For **Zone Name**, enter `myoracle.com`
    b. Ensure that your compartment is selected.
    c. Leave the **Zone Type** as the default
    d. Choose **Selecting existing DNS Private View** and select your VCN's private view (`<REGION>-OP-LAB01-1-VCN-01`). This will allow machines in your VCN to resolve DNS names in this private zone.
    e. Skip the advanced options.
    f. Click **Create**. This will take you to the zone's details page.

Add a Zone Record

4. You should still be on the zone's details page. If not, navigate to the zones in the main menu under **Networking** and **Zones**. Click the **Private Zones** tab. Click your zone's name.
5. Scroll down and click **Add Record**.
6. In the **Add Record** dialog box,
    a. Select a record type: **A - IPv4 Address** from the drop-down list.
    b. Name: *Leave blank*
    c. Click the lock icon at the right to unlock the TTL fields.

d. TTL: 30

e. TTL Unit: Seconds

f. Rdata Mode: Basic

g. Address: *<private-load-balancer-ip-address>*

7. Click **Submit**.

8. Repeat Step 4 with following data:

a. Select a record type: **CNAME** from the drop-down list.

b. Name: `web`

c. TTL: 30

d. TTL Unit: Seconds

e. Rdata Mode: Basic

f. Target: `myoracle.com`

9. Click **Submit**.

10. Once your records have been added, click **Publish Changes**.

11. In the confirmation dialog box, click **Publish Changes**.

Test the Server

1. To test whether Private DNS is working, log in to your client VM using SSH in Cloud Shell and enter following commands in succession.

```
$ ssh opc@<client-machine-public-ip>
$ curl -k http://myoracle.com
$ curl -k http://web.myoracle.com
```

2. Both `curl` commands should return this:

```
<!doctype html><html><body><h1>This is myoracle.com!
</h1></body></html>
```

3. Exit the SSH session:

```
$ exit
```

Configure Front-end SSL with Load Balancing

In the previous sections, you connected to the webserver(s) over HTTP. To use HTTPS, you'll need to pass an RSA key pair to the load balancer and create a certificate to bind the public key to the hostname.

In this practice, you will generate an RSA key pair and a self-signed certificate. Note that no major browser (nor `curl`) will trust actually a self-signed certificate, but you will use one for the purposes of this lab.

Create an SSL/TLS Certificate

1. Open Cloud Shell if it isn't already open. Make sure that you are not still in an SSH session (the prompt on the left should be `<your_username>@cloudshell`).

2. Use the following command to generate an RSA key pair and associated certificate:

```
$ openssl req -x509 -newkey rsa:4096 -keyout key.pem -out
cert.pem -sha256 -days 365 -nodes -subj '/CN=myoracle.com'
```

3. Enter `ls` and you should see both `cert.pem` and `key.pem`.

4. Download both `cert.pem` and `key.pem`:

   a. Click the gear at the top-right corner of Cloud Shell.

   b. Click **Download**.

   c. Type the name of the file you want to download (`cert.pem`)

   d. Repeat *Step a* to *Step c* for `key.pem`

Add the Certificate to the Load Balancer

1. In the navigation menu, select **Networking**, and then click **Load Balancers**.

2. Make sure your compartment is selected on the left under **List Scope**.

3. Click the name of your load balancer to go to its details page (`<REGION>-OP-LAB01-1-LB-01`).

4. Under **Resources** on the left, click **Certificates (0)**.

5. Above the certificates table, there should be a field titled **Certificate Resource**. Use the drop-down to select **Load Balancer Managed Certificate**.

6. Click **Add Certificate** to open a new dialog box.

    a. For Certificate Name, enter `<REGION>-OP-LAB01-1-CERT`. (Remember to replace `<REGION>` with the appropriate region code. See the Region Codes and Availability Domains topic in the Oracle Cloud Infrastructure Registry documentation.)

    b. Select **Choose SSL Certificate File** and upload `cert.pem` (you should have downloaded this earlier).

    c. Select the box for **Specify Private Key** and upload `key.pem`

    d. Click **Add Certificate** and close the work request notification that will pop up.

7. Wait for the certificate to appear in the table.

Create an HTTPS Listener

1. You should still be on your load balancer's details page. If not, navigate to it.

2. Under **Resources** on the left, click **Listeners**. The table should list the listener you created earlier to listen for HTTP (`<REGION>-OP-LAB01-1-LSN-01`).

3. Next, you will create a listener for HTTPS traffic. Click **Create the listener**.

    a. **Name**: `<REGION>-OP-LAB01-1-LSN-02`

    b. **Protocol**: HTTPS

    c. **Port**: 443

    d. **Certificate Resource**: Load Balancer Managed Certificate

    e. **Select the certificate**: `<REGION>-OP-LAB01-1-CERT`

    f. **Backend set:** Choose your backend set, `<REGION>-OP-LAB01-1-LBBS-01`

    g. **Idle Timeout in Seconds** (optional): 60

    h. **Routing Policy** (optional): Keep default

4. Click **Create Listener** and then click **Close** on the work request notification. Wait for the listener to appear in the table.

5. To test if the certificate is working, log in to the client VM (`<REGION>-OP-LAB01-1-VM-CLIENT`) using SSH in Cloud Shell. Return to the Instance Details page in Compute if you need the IP address.

```
$ ssh opc@<client-public-IP-address>
```

6. Use curl to send an HTTPS request for the webpage. The `-k` flag is needed to prevent curl from requiring a verified (non-self-signed) certificate.

```
$ curl -k https://myoracle.com
```

You should get the following webpage as a response:

```
<!doctype html><html><body><h1>This is
myoracle.com!</h1></body></html>
```

7. Repeat the curl command for the CNAME.

```
$ curl -k https://web.myoracle.com
```

You should get the same webpage as a response.

Common Operational Activities: Compare Storage Types

Lab 2-1 Practices

Estimated time: 45 minutes

Get Started

Overview

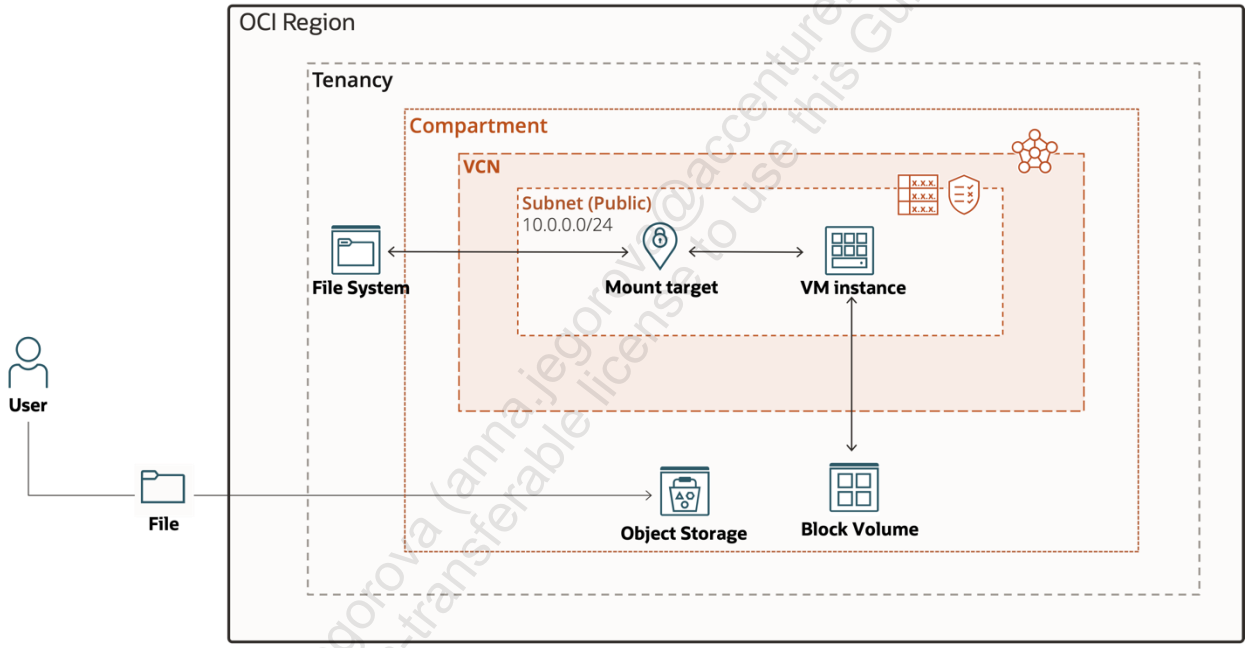
In this lab, you'll create and compare three different types of storage: block volumes, file systems, and Object Storage buckets.

In this lab, you'll:

a. Create a block volume and attach it to a compute instance.

b. Create a file system, export it, and mount in on a host.

c. Create a bucket and upload a file to it.



Prerequisites

- Requited policies (already set up for you)

Set Up Lab Environment

Before you begin the lab, you'll need to create a VCN and compute instance.

Create a Virtual Cloud Network

Use the VCN Wizard to set up a VCN with Internet connectivity.

1. In the navigation menu, navigate to **Networking**, and click **Virtual Cloud Networks**.
2. Confirm that you are in the proper compartment at the left.
3. Click **Start VCN Wizard**. Select **Create a VCN with Internet Connectivity**.
4. Under **Basic Information**, name the VCN `IAD-OP-LAB02-1-VCN-01`, replacing `IAD` with your appropriate region code.
5. Under **Configure VCN and Subnets**, leave the CIDR blocks and DNS resolution option as the defaults.
6. Click **Next** and confirm the information on the next screen.
7. Click **Create**. Wait for all of the resources to finish provisioning.
8. Click **View Virtual Cloud Network**.

Create SSH Keys for Cloud Shell

Note: If your Cloud Shell VM already has an SSH key pair and you have downloaded the public key, you can skip to creating the compute instances. You are likely to have done this in previous labs.

9. Open Cloud Shell using the icon at the top-right corner.
10. Use the following command in Cloud Shell to generate an SSH key pair (remember not to include the `$`):

```
$ ssh-keygen -t rsa -b 4096
```

11. It will prompt you for the location to save the files. Leave it empty and press **Enter** to use the default.
12. It will prompt you for a password. For this lab, leave it empty and press **Enter** to skip. Press **Enter** again to confirm.

13. Add your keys to the ssh-agent with the following two commands:

```
$ eval "$(ssh-agent -s)"
$ ssh-add ~/.ssh/id_rsa
```

14. Download the public key:

    a. Click the gear at the top-right corner of Cloud Shell.

    b. Click **Download**.

    c. Enter `.ssh/id_rsa.pub` to specify the public key. Click **Download** to confirm. This is the key you will upload to compute instances when provisioning them.

15. Close Cloud Shell.

Create the Compute Instance

16. In the navigation menu, click **Compute**, then **Instances** to go to the compute instances table.

17. Click **Create instance**, and fill in the fields:

    a. Name the instance `IAD-OP-LAB02-1-VM-01`, replacing `IAD` with your appropriate region code.

    b. Under **Create in compartment**, ensure that your compartment is selected.

    c. In the **Placement** box, select **AD 1** for the availability domain.

    d. In the **Image and Shape** box:

        1) For **Image**, select **Oracle Linux 8.x** (latest version).

        2) For **Shape**, select **VM.Standard.A1.Flex** (in the Ampere shape series) and configure it with **1 OCPU** and **6GB memory**.

    e. In the **Networking** box:

        1) Select your existing VCN, `IAD-OP-LAB02-1-VCN-01`.

        2) Select your existing public subnet, `Public Subnet-IAD-OP-LAB02-1-VCN-01`.

3) Assign a public IP address.

f. Upload the SSH key generated earlier.

g. In the **Boot volume** box, leave all boxes as default.

h. Click **Create**.

Create, Attach, and Mount a Block Volume to a Compute Instance

Block volumes are often used to add storage capacity to an Oracle Cloud Infrastructure instance. You can create a block volume through the Cloud Console or an API after you have created a VCN and instance. Once created, the volume can be attached to an instance using a volume attachment. You can then connect to the volume from your instance's guest OS using iSCSI (which you will use in this lab) or paravirtualized mode. The volume can then be mounted and used by your instance.

Provision the Block Volume

1. In the navigation menu, navigate to **Storage** and click **Block Storage**.

2. Click **Create Block Volume** and provide these details:
   - **Name**: `IAD-OP-LAB02-1-BV-01`
   - **Compartment**: Your compartment
   - **Availability Domain**: It must be the same AD as your instance (AD 1).
   - **Volume size**: Choose a custom size of 50 GB.
   - **Target Volume Performance**: Leave everything in this box as the default.
   - **Backup Policy**: Select the **No Backup Policy**. This would configure automatic backups.

   Note: There are three predefined backup policies: Bronze, Silver, and Gold. Each backup policy has a set backup frequency and retention periods. Bronze includes monthly incremental backups (12-month retention). Silver includes weekly (four-week retention) and monthly incremental backups. Gold includes daily (seven-day retention), weekly, and monthly incremental backups. All full backups are retained for five years.

3. Leave all other options (including cross-region replication as **OFF**) as their default values and click **Create Block Volume**. The volume will be ready to attach once its icon no longer lists it as **Provisioning** in the volume list.

Attach the Block Volume

Once the block volume is created, you can attach it to the VM instance you launched. When you attach a block volume to a VM instance, you have two options for attachment type, iSCSI or paravirtualized.

4. You can attach block volumes either from the block volume's page or the compute instance's page. You'll do it from the compute instance's page for this lab.

    a. Go to **Compute**, then **Instances** in the navigation menu.

    b. Click your VM instance to go to its details page.

    c. Click **Attached block volumes** to open the block volumes list.

5. Click **Attach block volume**.

    a. Click **Select volume** and choose your volume (`IAD-OP-LAB02-1-BV-01`).

    b. **Device Path:** Select `/dev/oracleoci/oraclevdb`

    c. For **Attachment mode**, choose **Paravirtualized**. Relative to ISCSI, this requires more virtualization, but is simpler to work with.

    d. For **Access**, choose **Read/write**. This will allow only one instance to attach to the block volume.

    e. Click **Attach**.

Format and Mount the Device in the Operating System

Now that the hardware is virtually attached, you need to configure the operating system of the VM to work with the volume.

6. Open Cloud Shell using the icon at the top-right corner.

7. Connect to the instance through SSH. The public IP address of the instance is listed on its details page. In Cloud Shell, SSH into the instance.

    ```
    ssh opc@<public_ip_address>
    ```

8. Enter `yes` to confirm.

9. Once the disk is attached, run these commands to format the disk and mount it.

    a. List the devices. You should see a device attached to oraclevdb.

    ```
    $ ls -l /dev/oracleoci/oraclevd*
    ```

    b. Make an `ext4` filesystem on the drive.

    ```
    $ sudo mkfs -t ext4 /dev/oracleoci/oraclevdb
    # Press y if prompted
    ```

c. Make a directory to mount the device on.

```
$ sudo mkdir /mnt/IAD-OP-LAB02-1-BV
```

d. Mount the device to the directory.

```
$ sudo mount /dev/oracleoci/oraclevdb /mnt/IAD-OP-LAB02-1-BV
```

e. Change directory and list the contents.

```
$ cd /mnt/IAD-OP-LAB02-1-BV
$ ls -l
```

f. List the disk space.

```
$ df -h
```

You should see a ~50G drive on `/mnt/IAD-OP-LAB02-1-BV`

10. Create a file on the disk.

```
$ sudo touch /mnt/IAD-OP-LAB02-1-BV/example_file.txt
```

Create a File System, Export It, and Mount in on a Host

In this practice, you will create a file system in File Storage. When you create this file system, the Cloud Console will automatically create an associated mount target and export. A mount target is an NFS endpoint in a subnet of your choice and allows you to access your file system. It provides the IP address or DNS name used in the mount command when connecting NFS clients to a file system. An export controls how an endpoint accesses a file system and includes an export path that uniquely identifies that file system, so that many file systems can exist on a single mount target. A file system must have at least one export in one mount target for instances to mount the file system.

Open the Subnet to NFS

First, you need to create a security list that will allow the file storage system to accept connections to and from the subnet. For the purposes of this lab, the mount target and the compute instance must be in the same subnet for these rules to work.

1. In the main menu, navigate to **Networking**, then **Virtual Cloud Network**. Click into your VCN, `IAD-OP-LAB02-1-VCN-01`.
2. In the left navigation pane, click **Security Lists**, then click your default security list.
3. Create the following ingress rules. All egress should already be allowed.
    a. Stateful ingress from all ports in the source CIDR block (10.0.0.0/24) to TCP port 111 and ports 2048-2050
    b. Stateful ingress from all ports in the source CIDR block (10.0.0.0/24) to UDP ports 111 and port 2048

Provision a File System

4. Open the navigation menu and click **Storage**. Under **File Storage**, click **File Systems**.
5. Make sure your correct compartment is listed at the left.
6. Click **Create File System**.

    Note: File systems are encrypted by default. You cannot turn off encryption.

    a. You can choose to accept the system defaults or change them by clicking **Edit Details**. Edit the file system name to be `IAD-OP-LAB02-1-FS`. Leave the rest of this area as default (AD, compartment, encryption).

Note on Encryption: File systems use Oracle-managed keys by default, which leaves all encryption-related matters to Oracle. Optionally, you can encrypt the data in this file system using your own Vault encryption key. Currently, only symmetric Advanced Encryption Standard (AES) keys are supported for file system encryption.

b. Note the export path, which is the file system name `/IAD-OP-LAB02-1-FS`.

c. Create a new mount target and edit the name to be `IAD-OP-LAB02-1-MNT-01` by clicking **Edit Details** at the right. Your pre-created VCN (`IAD-OP-LAB02-1-VCN-01`) and public subnet should be listed by default.

d. Click **Create**. Your file system, export and export path, and mount target will all be created.

7. Click your newly created file system, and then click **Exports** in the left navigation pane.

8. Find the export you just created, click the Actions menu (three vertical dots on the right), and then click **Mount Commands**.

9. In **Image**, ensure that **Oracle Linux** is selected.

10. Keep these commands open or note them in a text editor.

Mount the File System to the Instance

11. Open Cloud Shell if it is not already open. SSH into the instance if you are not already connected.

```
$ ssh opc@<public_ip_address>
```

12. Install the NFS client by copying and pasting the **Command to install NFS client** from the Console or typing the following:

```
$ sudo yum install nfs-utils
```

13. Create a mount point by copying and pasting the **Command to create the mount point directory** from the Console or type the following.

```
$ sudo mkdir -p /mnt/IAD-OP-LAB02-1-FS
```

14. Mount the file system by copying and pasting the **Command to mount the file system** from the Console or type the following. Replace `<private_ip>` with the private IP of the instance.

```
$ sudo mount <private_ip>:/IAD-OP-LAB02-1-FS /mnt/IAD-OP-LAB02-1-FS
```

15. Create a file on the file system.

```
$ sudo touch /mnt/IAD-OP-LAB02-1-FS/example_file.txt
```

16. You can see both the example file on the block volume and the example file on the file system:

```
$ sudo ls -R /mnt
```

Create a Bucket and Upload a File to It

Object Storage can store an unlimited amount of unstructured data of any content type, including analytic data and rich content such as images and videos. A regional service, Object Storage isn't tied to any specific compute instance. You can access an Object Storage endpoint (for example, https://objectstorage.us-ashburn-1.oraclecloud.com in the Ashburn/IAD region) through the service gateway you created in your VCN. Object Storage buckets are logical containers for storing objects. Users or systems create buckets as needed within a region. In this practice, you'll create an Object Storage bucket and upload a file to it.

Create a Bucket

1. In the navigation menu, under **Storage** and **Object Storage**, click **Buckets**.
2. Ensure your compartment is selected on the left.
3. Click **Create Bucket**.
4. In the **Create Bucket** dialog box, specify the attributes of the bucket:
   a. **Bucket Name:** `<your_username>-IAD-OP-LAB02-1-BKT-01`

      Note: Your bucket name has to be unique within the entire tenancy. If you run into a conflict, add numbers to the end.

   b. **Default Storage Tier:** Standard

   c. **Encryption:** Encrypt using Oracle-managed keys

      A few notes on bucket options:

      Storage tier types:
      - **Standard** is the default storage tier used for Object Storage service data and allows for immediate access.
      - **Archive** is the default storage tier used for Archive Storage service data.

      Do not select the other options, except for **Uncommitted Multipart Uploads Clean-up**, which will delete uncommitted multipart uploads.

   d. **Encryption:** Buckets are encrypted with keys managed by Oracle by default. You can optionally encrypt the data in this bucket using your own Vault encryption key, though that option does not exist for the purpose of this lab.

5. Click **Create**. The bucket is created immediately, and you can start uploading objects.

Create a Pre-Authenticated Request

To upload or download objects from your local machine, you could simply click upload or download on the bucket's details page, similar to any other website that involves uploading or downloading content. For this lab, you will upload and download from the compute instance that you provisioned earlier. There are two main methods to do this: pre-authenticated requests (PARs) and the OCI CLI. This lab will use PARs for simplicity, but the OCI CLI would be more secure.

6. In the left navigation panel, click **Pre-Authenticated Requests**.
7. Click **Create Pre-Authenticated Request** above the table.
    a. Name the PAR: `IAD-OP-LAB02-1-PAR-01`
    b. For **Pre-Authenticated Request Target**, select **Bucket**.
    c. For **Access Type**, select **Permit object reads and writes**.
    d. Keep **Enable Object Listing** deselected.
    e. Leave the expiration date as the default.
    f. Click **Create Pre-Authenticated Request** at the bottom to confirm.
8. Save the PAR URL to a text file.

Upload and Download to a Bucket from a VM Using a PAR

9. Open Cloud Shell if it is not already open. SSH into the instance if you are not already connected.
```
$ ssh opc@<public_ip_address>
```
10. Change to your home directory.
```
$ cd ~
```
11. Make a file to upload and download.
```
$ echo "Some example data" > example_file.txt
$ ls
```

12. Upload the file to Object Storage using an HTTP call. Substitute in the URL for the PAR that you recorded earlier.

```
$ curl -X PUT --data-binary '@/home/opc/example_file.txt'
<PAR_URL>example_file.txt
```

13. Look in the Console on your bucket's details page for the example file.

14. Delete the file from the compute instance.

```
$ rm example_file.txt
```

15. Download the file from Object Storage using an HTTP call. Substitute in the URL for the PAR that you recorded earlier.

```
$ curl -X GET <PAR-URL>example_file.txt
```

You should see the sample data you stored.

Common Operational Activities: Set Up Block Storage Disaster Recovery

Lab 3-1 Practices

Estimated Time: 45 minutes

Get Started

Overview

The block volume service offers a high level of durability—all volumes replicated across multiple storage servers with built-in repair mechanisms. However, block volume backups and replicas are still recommended for disaster recovery and business continuity for certain scenarios, such as application failure, availability domain failure, or operator error.

In this lab, you'll:

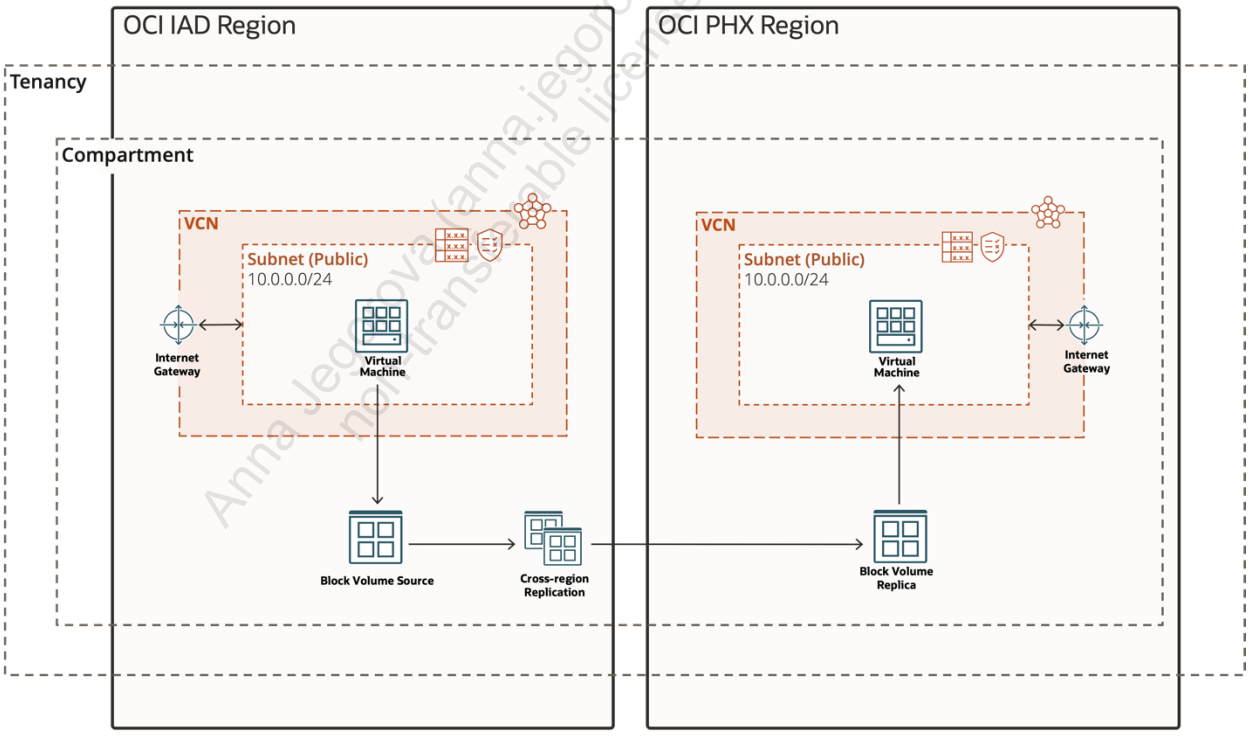
a. Create and attach a block volume to a compute instance.

b. Enable regular backups for the block volume.

c. Enable cross-region replication of the block volume and add data.

d. Activate the volume replica in the destination region.

e. Attach the volume replica to a compute instance in the destination region and view the data.



Prerequisites

- The required policies have already been set up for you.

Set Up Lab Environment

For this lab, you will need to create both a VCN and compute instance in two regions—one for the source block volume and one for the cross-region block volume replica.

Create a VCN in the US East (Ashburn) Region

Use the VCN Wizard to set up a VCN with Internet connectivity. This will house the compute instances you provision later.

1. In the navigation menu, navigate to **Networking**, and click **Virtual Cloud Networks**.
2. Confirm that you are in the proper compartment at the left.
3. Click **Start VCN Wizard**. Select **Create a VCN with Internet Connectivity**.
4. Under **Basic Information**, name the VCN `IAD-OP-LAB03-1-VCN-01`, replacing `IAD` with your appropriate region code.
5. Under **Configure VCN and Subnets**, leave the CIDR blocks and DNS resolution option as the defaults.
6. Click **Next** and confirm the information on the next screen.
7. Click **Create**. Wait for all of the resources to finish provisioning.
8. Click **View Virtual Cloud Network**.

Create a VCN in the US West (Phoenix) Region

Use the VCN Wizard to set up a VCN with Internet connectivity. This will house the compute instances you provision later.

1. Change to the US West (Phoenix) / PHX region in the console header.
2. In the navigation menu, navigate to **Networking**, and click **Virtual Cloud Networks**.
3. Confirm that you are in the proper compartment at the left.
4. Click **Start VCN Wizard**. Select **Create a VCN with Internet Connectivity**.
5. Under **Basic Information**, name the VCN `PHX-OP-LAB03-1-VCN-01`, replacing `PHX` with your appropriate region code.

6. Under **Configure VCN and Subnets**, leave the CIDR blocks and DNS resolution option as the defaults.

7. Click **Next** and confirm the information on the next screen.

8. Click **Create**. Wait for all of the resources to finish provisioning.

9. Click **View Virtual Cloud Network**.

Create SSH Keys for Cloud Shell

Note: If your Cloud Shell VM already has an SSH key pair and you have downloaded the public key, you can skip to creating the compute instance.

1. Open Cloud Shell using the icon at the top-right corner.

2. Use the following command in Cloud Shell to generate an SSH key pair (remember not to include the `$`):

```
$ ssh-keygen -t rsa -b 4096
```

3. It will prompt you for the location to save the files. Leave it empty and press **Enter** to use the default.

4. It will prompt you for a password. For this lab, leave it empty and press **Enter** to skip. Press **Enter** again to confirm.

5. Add your keys to the ssh-agent with the following two commands:

```
$ eval "$(ssh-agent -s)"
$ ssh-add ~/.ssh/id_rsa
```

6. Download the public key:

   a. Click the gear at the top-right corner of Cloud Shell.

   b. Click **Download**.

   c. Enter `.ssh/id_rsa.pub` to specify the public key. Click **Download** to confirm. This is the key you will upload to compute instances when provisioning them.

7. Close Cloud Shell.

Create Compute Instance in IAD Region

You'll now create a compute instance for your block volume source.

1. Return to the US East (Ashburn)/IAD region in the console header.
2. In the navigation menu, navigate to Compute, then click **Instances**. Click **Create instance**, and fill in the fields:
    a. **Name**: `IAD-OP-LAB03-1-VM-01`
    b. Create in your compartment.
    c. **Availability Domain**: `AD 1`
    d. **Image and Shape**: Oracle Linux 8.X (pick the latest) and VM.Standard.A1.Flex with 1 OCPU and 6 GB memory (select Ampere shape series).
    e. Select your existing IAD region VCN, `IAD-OP-LAB03-1-VCN-01`.
    f. Select your existing IAD region subnet, `Public Subnet-IAD-OP-LAB03-1-VCN-01`.
    g. Upload the public SSH key that you downloaded from Cloud Shell (it should be named `id_rsa.pub`).
3. Click **Create Instance**.

Create Compute Instance in PHX Region

You'll now create a compute instance for your block volume replica destination.

1. Return to the US West (Phoenix)/PHX region in the console header.
2. If you are not on the compute instances table, then in the navigation menu, navigate to **Compute**, then click **Instances**. Click **Create instance**, and fill in the fields:
    a. **Name**: `PHX-OP-LAB03-1-VM-01`
    b. Create in your compartment.
    c. **Availability Domain**: `AD 1`

d. **Image and Shape**: Oracle Linux 8.X (pick the latest) and VM.Standard.A1.Flex with 1 OCPU and 6GB memory (select Ampere shape series).

e. Select your existing PHX region VCN, `PHX-OP-LAB03-1-VCN-01`.

f. Select your existing PHX region subnet, `Public Subnet-PHX-OP-LAB03-1-VCN-01`.

g. Upload the public SSH key that you downloaded from Cloud Shell (it should be named `id_rsa.pub`).

3. Click **Create Instance**.

Create and Attach Block Volume to a Compute Instance

In this practice, you'll create, attach, and mount the block volume that you will later use as the source for cross-region replication. During cross-region replication, a replica of the source is created, and when that replica is activated, it creates a clone of the source volume.

Provision the Block Volume

1. Return to the US East (Ashburn) / IAD region in the console header.
2. In the navigation menu, navigate to **Storage** and click **Block Storage**.
3. Click **Create Block Volume** and provide these details:
   - **Name**: `IAD-OP-LAB03-1-BV-SRC`
   - **Compartment**: Your compartment
   - **Availability Domain**: It must be the same AD as your instance (AD 1).
   - **Volume size**: Choose a custom size of 50 GB.
   - **Target Volume Performance**: Leave everything in this box as the default.
   - **Backup Policy**: Select the **No Backup Policy**. This would configure automatic backups.
4. Leave all other options (including cross-region replication as **OFF**) as their default values and click **Create Block Volume**. The volume will be ready to attach once its icon no longer lists it as Provisioning in the volume list.

Create a Block Volume Backup

5. Aside from automatic backups, you can create manual backups at any time. In the left navigation pane, click **Block Volume Backups**, then click **Create Block Volume Backup** to add a manual backup.
   a. Name the backup: `Backup01`
   b. Select **Full Backup**.
   c. Click **Create Block Volume Backup**. Your backup will now be listed under Block Volume Backups.

Attach the Block Volume

6. You can attach block volumes either from the block volume's page or the compute instance's page. You'll do it from the compute instance's page for this lab.

    a. Go to the **Compute**, then **Instances** in the navigation menu.

    b. Click your VM instance to go to its details page.

    c. Click **Attached block volumes** to open the block volumes list.

7. Click **Attach block volume**.

    a. Click **Select volume** and choose your volume (`IAD-OP-LAB03-1-BV-SRC`)

    b. **Device Path:** Select `/dev/oracleoci/oraclevdb`

    c. For **Attachment mode**, choose **Paravirtualized**. Relative to ISCSI, this requires more virtualization, but is simpler to work with.

    d. For **Access**, choose **Read/write**. This will allow only one instance to attach to the block volume.

    e. Click **Attach**.

Format and Mount the Device in the Operating System

Now that the hardware is virtually attached, you need to configure the operating system of the VM to work with the volume.

8. Open Cloud Shell using the icon at the top-right corner.

9. Connect to the instance through SSH. The public IP address of the instance is listed on its details page. In Cloud Shell, SSH into the instance.

    ```
    ssh opc@<ashburn_vm_public_ip_address>
    ```

10. Enter `yes` to confirm.

11. Once the disk is attached, run these commands to format the disk and mount it.

    a. List the devices. You should see a device attached to oraclevdb.

    ```
    $ ls -l /dev/oracleoci/oraclevd*
    ```

b. Make an ext4 filesystem on the drive.

```
$ sudo mkfs -t ext4 /dev/oracleoci/oraclevdb
# Press y when prompted
```

c. Make a directory to mount the device on.

```
$ sudo mkdir /mnt/datadisk1
```

d. Mount the device to the directory.

```
$ sudo mount /dev/oracleoci/oraclevdb /mnt/datadisk1
```

e. Change directory and list the contents

```
$ cd /mnt/datadisk1
$ ls -l
```

Note: When mounting a storage volume for the first time, you can format the storage volume and create a single, primary partition that occupies the entire volume by using the `fdisk` command (Caution: Using `fdisk` to format the disk deletes any data on the disk).

12. Next, create a file named `before_xrr.txt`. This file will be the file replicated after you enable cross-region replication and activate and attach the volume replica in the next practices. Enter the code:

```
$ sudo bash -c 'echo "Created before XRR was enabled" >
before_xrr.txt'
```

13. Type `cat before_xrr.txt` to see the contents of the new file.

Enable Cross-Region Replication

Enable Cross-Region Replication

1. In the navigation menu, click **Storage**. Under Block Storage, click **Block Volumes**.
2. Click the block volume (`IAD-OP-LAB03-1-BV-SRC`) to open its details page.
3. Click **Edit**. This will open an editing form.
    a. Scroll until you see **Backup Policies**. This is where you could enable automatic backups after you have already created a volume.
    b. In the Cross Region Replication section, select **ON**. This will add options.
        1) For **Region**, select **PHX / US West (Phoenix)**. Each region has specific regions that it can replicate to.
        2) For **Availability Domain** select the AD of the other instance (**AD 1**).
        3) Change the name for the volume replica: `PHX-OP-LAB03-1-BV-REPLICA`**.**
    c. Select **Confirm** to acknowledge the cost warning.
    d. Click **Save Changes**.
4. Select the **US West (Phoenix)** region in the console header.
5. If you're not already in the Block Volume area, in the navigation menu, click **Storage**. Under Block Storage, click **Block Volumes**.
6. Click **Block Volume Replicas** on the left navigation panel to verify that replica is being provisioned for you.

Add Data to the Block Volume

7. Change regions to **US East (Ashburn)**.
8. If Cloud Shell is not already open, then open it using the icon at the top-right corner.
9. If you are not already connected to the Ashburn instance, then connect to it through SSH. The public IP address of the instance is listed on its details page. In Cloud Shell, SSH into the instance.

```
ssh opc@<ashburn_vm_public_ip_address>
```

10. Change directory to the mounted volume.

```
$ cd /mnt/datadisk1
$ ls -l
```

11. Make a new file for cross-region replication to replicate.

```
$ sudo bash -c 'echo "Created after XRR was enabled" >
after_xrr.txt'
```

12. Type `cat after_xrr.txt` to see the contents of the new file.

Activate Volume Replica in Destination Region

Activate the Volume Replica

To create a new volume from a volume replica, you need to activate the replica. The activation process creates a new volume by cloning the replica.

1. Change to the **US West (Phoenix) region** and navigate to the **Block Volume Replicas** page.
2. Click the replica's name once it is done provisioning.
3. Click **Activate** to open the Activate Volume Replica form. On the form, specify the settings for the new volume, including:
   - **Name**: `PHX-OP-LAB03-1-BV-DST`
   - **Compartment**: Your compartment
   - **Size and performance settings**: Default (this will copy the source)
   - **Backup policy**: None
   - **Cross Region Replication**: Off
   - **Encryption:** Oracle-managed key
4. Click **Activate Replica** to confirm.
5. Return to the **Block Volumes** by clicking **Block Storage** in the breadcrumbs, then **Block Volumes** in the left navigation pane. The new volume will appear in the block volumes list, in the provisioning state.

Add More Data to the Source Block Volume

6. Change regions to **US East (Ashburn)**.
7. If Cloud Shell is not already open, then open it using the icon at the top-right corner.
8. If you are not already connected to the **Ashburn** instance, then connect to it through SSH. The public IP address of the instance is listed on its details page. In Cloud Shell, SSH into the instance.

   ```
   ssh opc@<ashburn_vm_public_ip_address>
   ```

9. Change directory to the mounted volume.

   ```
   $ cd /mnt/datadisk1
   $ ls -l
   ```

10. Make a new file for cross-region replication to replicate.

```
$ sudo bash -c 'echo "Created after the replica was activated" >
after_activation.txt'
```

11. Type `cat after_activation.txt` to see the contents of the new file.

Attach Volume Replica to Compute Instance in Destination Region

Attach the Replicated Block Volume

1. Ensure that you are in the **US West (Phoenix) region**.
2. In the navigation menu, under Compute, click **Instances**, and click into the Phoenix VM (`PHX-OP-LAB03-1-VM-1`). Click **Attached block volumes** in the left navigation pane.
3. Click **Attach block volume**.
    a. Click **Select volume** and choose your volume (`PHX-OP-LAB03-1-BV-DST`).
    b. **Device Path:** Select `/dev/oracleoci/oraclevdb`
    c. For **Attachment mode**, choose **Paravirtualized**. Relative to ISCSI, this requires more virtualization, but is simpler to work with.
    d. For **Access**, choose **Read/write**. This will allow only one instance to attach to the block volume.
    e. Click **Attach**.

Format and Mount the Device in the Operating System

Now that the hardware is virtually attached, you need to configure the operating system of the VM to work with the volume.

4. Open Cloud Shell using the icon at the top-right corner. (Close and reopen it if it is already open.)
5. Connect to the **Phoenix** instance through SSH. The public IP address of the instance is listed on its details page. In Cloud Shell, SSH into the instance.
    ```
    ssh opc@<phoenix_vm_public_ip_address>
    ```
6. Enter `yes` to confirm.
7. Once the disk is attached, run these commands to format the disk and mount it.
    a. List the devices. You should see a device attached to oraclevdb.
    ```
    $ ls -l /dev/oracleoci/oraclevd*
    ```

b. Make a directory to mount the device on.

```
$ sudo mkdir /mnt/datadisk1
```

c. Mount the device to the directory.

```
$ sudo mount /dev/oracleoci/oraclevdb /mnt/datadisk1
```

View the Data on the Replicated Volume

8. Now go to `datadisk1` and verify that the replicated files are available.

```
$ cd /mnt/datadisk1
$ ls
```

9. You should see the files `before_xrr.txt` and `after_xrr.txt` but **not** `after_activation.txt`. This is because the activated volume is a *point-in-time clone* of the replication target at the time of activation. If you were to activate the replica *again*, then you would see `after_activation.txt` on the new volume.

Common Operational Activities: Create a Compute Instance in Cloud Console, CLI, and SDK

Lab 4-1 Practices

Estimated Time: 45 minutes

Get Started

Overview

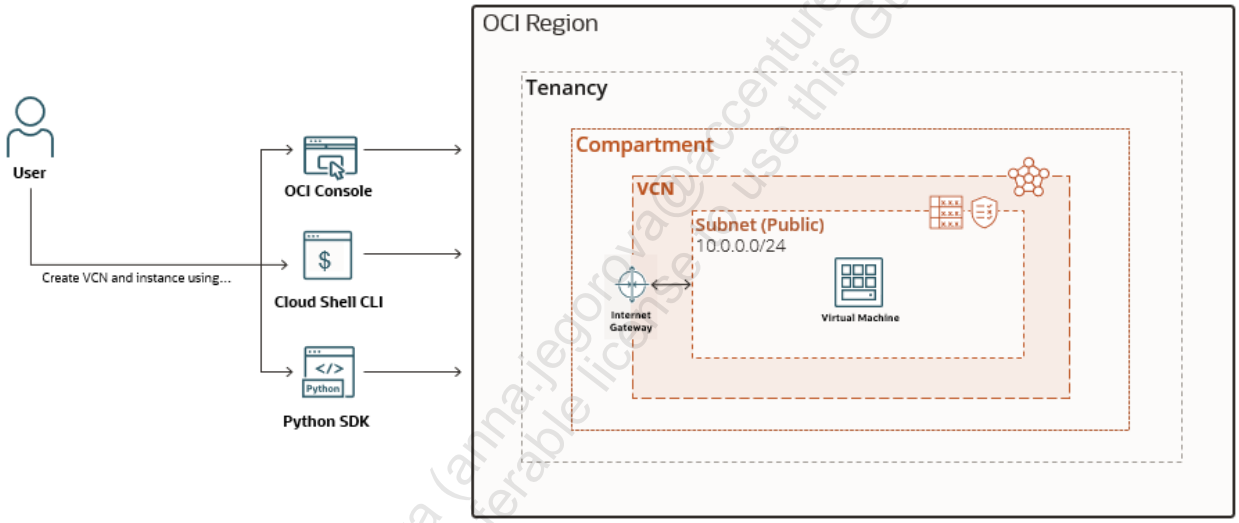
In this lab, you will provision a compute instance using three different methods to compare their functionality.

In this lab, you'll:

a. Create a VM instance in the Cloud Console.

b. Create a VM instance using Cloud Shell in the CLI on the Cloud Console.

c. Create a VM instance using a Python SDK in Cloud Editor and Cloud Shell.



Prerequisites

- The required IAM permissions have already been set up for you.

Assumptions

- Basic familiarity with Bash commands and Python

Set Up Lab Environment

Create a Virtual Cloud Network

Use the VCN Wizard to set up a VCN with Internet connectivity. This will house the compute instances you provision later.

1. In the navigation menu, navigate to **Networking**, and click **Virtual Cloud Networks**.
2. Confirm that you are in the proper compartment at the left.
3. Click **Start VCN Wizard**. Select **Create a VCN with Internet Connectivity**.
4. Under **Basic Information**, name the VCN `IAD-OP-LAB04-1-VCN-01`, replacing `IAD` with your appropriate region code.
5. Under **Configure VCN and Subnets**, leave the CIDR blocks and DNS resolution option as the defaults.
6. Click **Next** and confirm the information on the next screen.
7. Click **Create**. Wait for all of the resources to finish provisioning.
8. Click **View Virtual Cloud Network**.

Create SSH Keys for Cloud Shell

Note: If your Cloud Shell VM already has an SSH key pair and you have downloaded the public key, you can skip this.

9. Open Cloud Shell using the icon at the top-right corner.
10. Use the following command in Cloud Shell to generate an SSH key pair (remember not to include the `$`):

```
$ ssh-keygen -t rsa -b 4096
```

11. It will prompt you for the location to save the files. Leave it empty and press **Enter** to use the default.
12. It will prompt you for a password. For this lab, leave it empty and press **Enter** to skip. Press **Enter** again to confirm.

13. Add your keys to the ssh-agent with the following two commands:

```
$ eval "$(ssh-agent -s)"
$ ssh-add ~/.ssh/id_rsa
```

14. Download the public key:

    a. Click the gear at the top-right corner of Cloud Shell.

    b. Click **Download**.

    c. Enter `.ssh/id_rsa.pub` to specify the public key. Click **Download** to confirm. This is the key you will upload to compute instances when provisioning them.

15. Close Cloud Shell.

Create a VM Instance in the Cloud Console

In this practice, you'll use the Cloud Console to create a compute instance in your public subnet.

Create the Compute Instance

1. In the navigation menu, click **Compute**, then **Instances** to go to the compute instances table.
2. Click **Create instance**, and fill in the fields:
    a. Name the instance `IAD-OP-LAB04-1-VM-01`, replacing `IAD` with your appropriate region code.
    b. Under **Create in compartment**, ensure that your compartment is selected.
    c. In the **Placement** box, select **AD 1** for the availability domain.
    d. In the **Image and Shape** box:
        1) For **Image**, select **Oracle Linux 8.x** (latest version).
        2) For **Shape**, select **VM.Standard.A1.Flex** (in the Ampere shape series) and configure it with **1 OCPU** and **6 GB memory**.
    e. In the **Networking** box:
        1) Select your existing VCN, `IAD-OP-LAB04-1-VCN-01`.
        2) Select your existing public subnet, `Public Subnet-IAD-OP-LAB04-1-VCN-01`.
        3) Assign a public IP address.
    f. Upload the public SSH key generated earlier in Cloud Shell.
    g. In the **Boot volume** box, leave all boxes as the default.
    h. Click **Create**.
3. Once it is done provisioning, click **Terminate** to terminate the instance. Select **Permanently delete the attached boot volume** and click **Terminate instance** to confirm.

Create a VM Instance Using Cloud Shell Command Line Interface

In this practice, you'll use the CLI to access Oracle Cloud Infrastructure and carry out service-related tasks in the VCN you created.

Use the OCI CLI Help Option

1. Launch Cloud Shell (the **>_** icon at the top-right corner of the screen, next to the region).

2. Most OCI CLI commands must specify a *service*, followed by a *resource type* and then an *action*. The basic command-line syntax is:

   ```
   $ oci <service> <type> <action> <options>
   ```

   For this practice, this syntax is applied as follows:

   - Compute is the `<service>`.
   - Instance is the resource `<type>`.
   - Launch is the `<action>`.
   - The rest of the command string consists of `<options>`.

3. To see the information for each command, use the `--help` option. For example, try the following:

   ```
   $ oci --help
   $ oci network --help
   $ oci network vcn --help
   $ oci network vcn list --help
   ```

Get Input Templates from the OCI CLI

4. Some options, such as `--shape-config` for launching a compute instance, take a JSON string as input. To see the form of this JSON string, use the `--generate-param-json-input` option:

   ```
   $ oci compute instance launch \
     --generate-param-json-input shape-config
   ```

   We will use this template later.

Launch the Compute Instance

5. To see what is required to launch an instance, enter:

   ```
   $ oci compute instance launch -h
   ```

6. Enter the OCI CLI interactive mode:

```
$ oci -i
```

7. You can use interactive mode to fill in details such as availability domain and compartment OCID by pressing the down arrow; for example, type the following without pressing enter the following **with a trailing space**:

```
$ oci compute instance launch --compartment-id
```

8. Wait a moment, then you can use the up and down arrows to find your compartment. Press Enter and the CLI will autofill the OCID.

9. Enter the following in the OCI CLI interactive mode. Lines are separated by backslashes, but **do not** actually separate them; enter the following as one line. They are separated for display purposes. Compartment OCID, availability domain, subnet OCID, and image ID should all be filled using interactive mode's autofill feature.

```
$ oci compute instance launch \
  --compartment-id <The ID of your compartment> \
  --availability-domain <The name of the first AD> \
  --subnet-id <The ID of "Public Subnet-IAD-OP-LAB04-1-VCN-01">\
  --display-name "IAD-OP-LAB04-1-VM-02" \
  --image-id <Latest Oracle Linux 8 Image for ARM (aarch64)> \
  --shape "VM.Standard.A1.Flex" \
  --shape-config '{"ocpus": "1", "memoryInGbs": "6"}' \
  --assign-public-ip true \
  --ssh-authorized-keys-file "/home/<shell-
user>/.ssh/id_rsa.pub"
```

For example, your command may look like this:

```
$ compute instance launch --compartment-id
ocid1.compartment.oc1..aaaaaaaa4zmhdgjpun266xpps4imcfroe42oxxa24lwv
y4wxxeimgbuafguq --availability-domain CPQY:US-ASHBURN-AD-1 --
subnet-id
ocid1.subnet.oc1.iad.aaaaaaaaabl3sc36muka6dlf5y7hcanvdomss6jthal6kp
h3j5whv7pkq74q --display-name "IAD-OP-LAB04-1-VM-02" --image-id
ocid1.image.oc1.iad.aaaaaaaabta3gqhlfoam3l7rayd4inr6hhg4cmvcqb7p43q
adkiehnszyuka --shape VM.Standard.A1.Flex --shape-config '{"ocpus":
"1", "memoryInGbs": "6"}' --assign-public-ip true –ssh -authorized-
keys-file "/home/x_28012023/.ssh/id_rsa.pub"
```

Record the instance OCID to use in the next few steps.

10. Navigate to compute instances in the navigation menu under **Compute** > **Instances**. You should see an instance provisioning.

11. Get information about compute instance.

```
$ oci compute instance get --instance-id <instance-OCID>
```

12. Stop the compute instance.

```
$ oci compute instance action \
  --action STOP \
  --instance-id <instance-OCID>
```

13. Terminate the compute instance.

```
$ oci compute instance terminate --instance-id <instance-OCID>
```

14. Enter **y** to confirm.

15. Ensure that the compute instance has terminated under **Compute** > **Instances** in the console.

Create VM Instance Using Python SDK

Next, you'll create a VM instance using Python SDK. First, you'll need to create a folder and file using Code Editor in the Cloud Console. You'll then create the VM using a Python script.

Create a Folder and File in Code Editor

1. Open the console's Code Editor, whose icon is at the top-right corner, to the right of the CLI Cloud Shell icon.
2. Expand the Explorer panel with the top icon on the left panel. It looks like two overlapping documents.
3. Expand the drop-down for your home directory if it isn't already expanded. It's okay if it is empty.
4. Create a new folder by clicking **File**, then **New Folder**, and name it `sdk_example`.
5. Create a file in that folder by clicking **File**, then **New File**, and name it `provision_compute.py`. This will also open the file.

Create the VM Using Python Script

1. Enter the following code:

```
import oci

# Load your tenancy, user, etc. information
config = oci.config.from_file()

# Variables to fill in
compartment_id = "" # Fill with your compartment OCID
image_id = "" # Fill with the Oracle Linux 8 OCID for your region
subnet_id = "" # Fill with your public subnet OCID
display_name = "IAD-OP-LAB04-1-VM-3" # Fill with your display name

# Get first availability domain name
iam = oci.identity.IdentityClient(config)
availability_domain =
iam.list_availability_domains(compartment_id).data[0].name

# Package all settings together
launch_instance_details = oci.core.models.LaunchInstanceDetails(
    availability_domain=availability_domain,
```

```
        compartment_id=compartment_id,
        display_name=display_name,
        shape="VM.Standard.A1.Flex",
        shape_config=oci.core.models.LaunchInstanceShapeConfigDetails(
            ocpus=1, memory_in_gbs=6),
        source_details=oci.core.models.InstanceSourceViaImageDetails(
            image_id=image_id, source_type="image"),
        create_vnic_details=oci.core.models.CreateVnicDetails(
            assign_public_ip=True, subnet_id=subnet_id)
    )

    # Launch compute instance
    compute = oci.core.ComputeClient(config)
    compute.launch_instance(launch_instance_details)
```

2. Fill in the required variables.

    a. For compartment OCID, navigate to **Identity & Security** > **Compartments** (minimize Code Editor) and find your compartment in the table.

    b. For the image OCID, find the corresponding OCID for your region here: https://docs.oracle.com/en-us/iaas/images/image/aaca995d-5c38-4d7b-9d9e-0f4c3e05a66c/

    c. For subnet OCID, navigate to **Networking** > **Virtual Cloud Networks** (minimize code editor) and click into your VCN. Click the public subnet to go to its details. Find the OCID.

    d. For display name, enter `IAD-OP-LAB04-1-VM-03`.

3. Save the file with by going to File, then clicking **Save**.

4. Open a new terminal by going to Terminal, then clicking **New Terminal**.

5. In the terminal, enter the following:

```
$ python3 sdk_example/provision_compute.py
```

6. Navigate to compute instances. You should see your compute instance provisioning.

7. Click the compute instance's name to go to its details page.

8. Once it is done provisioning, click **Terminate** to terminate the instance. Select **Permanently delete the attached boot volume** and click **Terminate instance** to confirm.

Common Operational Activities: Automate Backing Up a File to Object Storage

Lab 5-1 Practices

Estimated time: 45 minutes

Get Started

Overview

The Block Volume service offers built-in resilience through automatic replication, as well as disaster recovery through cross-region replication, automatic backups, and manual backups. However, there often arises the need to back up a specific file or folder, rather than the entire drive.

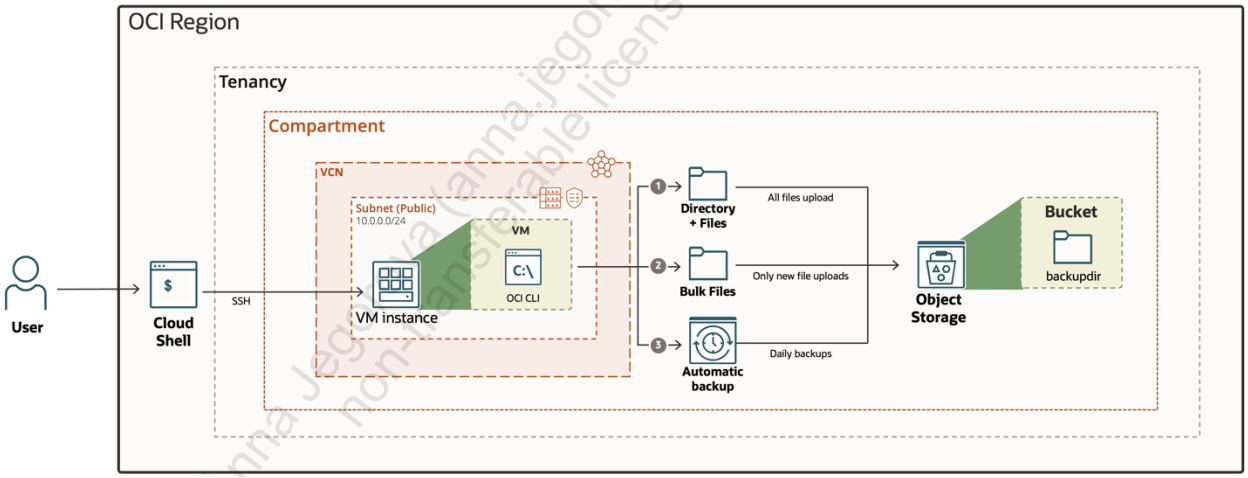
In this lab, you will use the Oracle Cloud Infrastructure (OCI) CLI to write an automation script to back up your desired data onto OCI Object Storage. The CLI is a small-footprint tool that you can use on its own or with the Console to complete Oracle Cloud Infrastructure tasks.

In this lab, you'll:

a. Install and configure the OCI CLI on a compute instance.

b. Manually back up files to Object Storage using CLI.

c. Create an automatic backup of archive to Object Storage using a Cloud Shell script and `cron`.



Prerequisites

- The required IAM policies have been set up for you.

Assumptions

- Familiarity with basic Linux commands, shell scripting, and `cron`

Set Up Lab Environment

To complete this lab, you must first create a VCN and compute instance.

Create a VCN

Use the VCN Wizard to set up a VCN with Internet connectivity. This will house the compute instances you provision later.

1. In the navigation menu, navigate to **Networking**, and click **Virtual Cloud Networks**.
2. Confirm that you are in the proper compartment at the left.
3. Click **Start VCN Wizard**. Select **Create a VCN with Internet Connectivity**.
4. Under **Basic Information**, name the VCN `IAD-OP-LAB05-1-VCN-01`, replacing `IAD` with your appropriate region code.
5. Under **Configure VCN and Subnets**, leave the CIDR blocks and DNS resolution option as the defaults.
6. Click **Next** and confirm the information on the next screen.
7. Click **Create**. Wait for all of the resources to finish provisioning.
8. Click **View Virtual Cloud Network**.

Create SSH Keys for Cloud Shell

Note: If your Cloud Shell VM already has an SSH key pair and you have downloaded the public key, you can skip to creating the compute instance.

9. Open Cloud Shell using the icon at the top-right corner.
10. Use the following command in Cloud Shell to generate an SSH key pair (remember not to include the `$`):

```
$ ssh-keygen -t rsa -b 4096
```

11. It will prompt you for the location to save the files. Leave it empty and press **Enter** to use the default.
12. It will prompt you for a password. For this lab, leave it empty and press **Enter** to skip. Press **Enter** again to confirm.

13. Add your keys to the ssh-agent with the following two commands:

```
$ eval "$(ssh-agent -s)"
$ ssh-add ~/.ssh/id_rsa
```

14. Download the public key:

    a. Click the gear at the top-right corner of Cloud Shell.

    b. Click **Download**.

    c. Enter `.ssh/id_rsa.pub` to specify the public key. Click **Download** to confirm. This is the key you will upload to compute instances when provisioning them.

15. Close Cloud Shell.

Create a Compute Instance

You'll now create a compute instance in your public subnet, which you'll use in the first task, when you set up CLI on it.

16. In the navigation menu, navigate to Compute, then click **Instances**. Click **Create instance**, and fill in the fields:

    a. **Name**: `IAD-OP-LAB05-1-VM-01`

    b. Create in your compartment.

    c. **Availability Domain**: `AD 1`

    d. **Image and Shape**: Oracle Linux 8.X (pick the latest) and VM.Standard.A1.Flex with 1 OCPU and 6 GB memory (select Ampere shape series)

    e. Select your existing IAD region VCN, `IAD-OP-LAB05-1-VCN-01`.

    f. Select your existing IAD region subnet, `Public Subnet-IAD-OP-LAB05-1-VCN-01`.

    g. Upload the public SSH key that you downloaded from Cloud Shell (it should be named `id_rsa.pub`).

17. Click **Create**.

Set Up OCI CLI on Compute Instance

In this practice, you'll install OCI CLI on a compute instance. The CLI provides the same core functionality as the Console, plus additional commands. Some of these, such as the ability to run scripts, extend Console functionality.

Install the OCI CLI

1. Launch Cloud Shell (the **>_** icon at the top-right corner of the screen, next to the region).

2. SSH into the instance in Cloud Shell:

```
$ ssh opc@<public_ip_address>
  # Enter yes to confirm
```

3. Ensure that Python is installed:

```
$ sudo dnf -y update # This can take a moment
$ sudo dnf -y groupinstall "Development Tools"
$ sudo dnf -y install gcc wget openssl-devel bzip2-devel libffi-
devel
$ sudo dnf -y module enable python36
$ sudo dnf -y install python36
```

4. Install the OCI CLI on it by entering the following command:

```
$ sudo pip3 install oci-cli
```

   In general, it is preferable to install the OCI CLI inside of a Python virtual environment. For this lab, you will skip this for simplicity's sake.

5. Check that OCI CLI is installed by entering the following command:

```
$ oci -v
```

Gather Information

6. You need to gather some information so that you can configure OCI. First, you'll need your user OCID. Click the profile icon at the top-right corner and click your username. Copy your user OCID. Record your OCID in a text file.

7. Get your tenancy OCID. Click the profile icon again, then click your tenancy. Copy your tenancy OCID. Record your tenancy OCID, because you will need it later.

8. Next, get your region identifier. Click your region in the header, then scroll down and click **Manage regions**. Copy your region identifier and record it. For US East (Ashburn), it is **us-ashburn-1**.

9. Next, configure OCI CLI. Enter the following command:

```
$ oci setup config
```

   a. When prompted for a location for your config, press **Enter** to choose the default location.

   b. When prompted for your user OCID, tenancy OCID, and region ID, enter the appropriate information.

   c. When asked if you want to generate a new RSA key pair, enter **Y**.

   d. When prompted for the directory to place the RSA key pair, press **Enter** to select the default location (`/home/opc/.oci`).

   e. When prompted for the name of the key, press **Enter** to select the default name (`oci_api_key`).

   f. When prompted for a passphrase, enter a passphrase. Note that it will not display the passphrase as you type it.

   g. Enter the passphrase again to confirm.

   h. When asked whether to save the passphrase to the config file, enter **Y**. Be careful not to display the config file, as it will show the passphrase in plain text. It is better practice to not save the passphrase, but you can take this shortcut for this lab.

Add the API Key to Your OCI Account

10. The OCI setup config command generated an API key. You need to upload this API key into your OCI account for authentication of API calls. Enter this text:

```
$ cat ~/.oci/oci_api_key_public.pem
```

11. Copy the key from Oracle Cloud Shell. Be sure that it is the public key.

12. Click the profile icon again, then your username. Scroll down, click **API Keys**, then click **Add Public Key**. Select **Paste Public Key**, paste it, and click **Add**.

Create Object Storage Backup Files Using OCI-CLI

In this practice, you will create an Object Storage bucket using the OCI CLI and then follow a manual process to back up desired files. In OCI, an object is a file or unstructured data you upload to a bucket within a compartment within an Object Storage namespace.

Create a Bucket

1. Enter the following command to get your tenancy's Object Storage namespace:

```
$ oci os ns get
```

2. Create an Object Storage bucket using the following commands. Replace `<compartment-id>`, `<your-username>`, and `<namespace>` with your information. Bucket names must be unique within the entire namespace, not just the compartment. If you run into a conflict, add numbers at the end.

```
$ oci os bucket create --compartment-id <compartment-id> --name
<your-username>-IAD-OP-LAB05-1-BKT
```

You will get message that a bucket was created successfully. List the objects in the bucket.

```
$ oci os object list --bucket-name <your-username>-IAD-OP-LAB05-
1-BKT --namespace <namespace>
```

Manually Upload Files into the Bucket

3. Create a directory and multiple files to upload in bucket.

```
$ mkdir backupdir
$ cd backupdir
$ touch testfile{1..5}
$ ls
```

4. Upload files in `backupdir` to the bucket you created. Replace `<namespace>` with your namespace.

```
$ oci os object bulk-upload -ns <tenancy> \
  --bucket-name <your-username>-IAD-OP-LAB05-1-BKT \
  --src-dir /home/opc/backupdir
```

You will see the message about files uploaded to Object Storage.

5. Create another file, and bulk upload all files to the bucket created. Replace `<namespace>` with your namespace.

```
$ touch latestfile
$ oci os object bulk-upload \
  -ns <namespace> \
```

```
--bucket-name <your-username>-IAD-OP-LAB05-1-BKT \
--src-dir /home/opc/backupdir \
--no-overwrite
```

You will see the message that only the newly created file is uploaded to Object Storage.

6. Once you see this message, you can delete all files uploaded to Object Storage. Enter the following code. Replace `<namespace>` with your namespace.

```
$ oci os object bulk-delete --namespace <namespace> \
  --bucket-name <your-username>-IAD-OP-LAB05-1-BKT
```

You will get this warning:

```
WARNING: This command will delete at least 6 objects. Are you
sure you wish to continue? [y/N]:
```

7. Enter **y** to delete all files.

Create Object Storage Backup Files Using Shell Script

In this practice, you'll write an automation script to back up desired data on Object Storage.

Create a Changing File to Backup

1. List the already existing bucket in your compartment using the following command. Replace `<compartment-id>` with your compartment OCID.

   ```
   $ oci os bucket list --compartment-id <compartment-id>
   ```

2. Find your path to oci with the following command:

   ```
   $ which oci
   ```

   The location will likely be `/usr/local/bin/oci`

3. Create a Shell script named `daily-backup.sh` in your home directory. You can use any editor you like, but this lab will give the instructions for `vim`.

   ```
   $ cd ~
   $ vim daily-backup.sh
   ```

   Press `i` to enter insert mode in vim.

   In this file, **type** the OCI CLI command you used earlier to upload a directory without overwriting. Be sure to fill in your namespace and unique bucket name. Also replace your path to OCI with the output of the previous step.

   ```
   #!/bin/bash

   /path/to/oci os object bulk-upload \
     -ns <namespace> \
     --bucket-name <your-username>-IAD-OP-LAB05-1-BKT \
     --src-dir /home/opc/backupdir \
     --no-overwrite
   ```

   Exit vim by pressing **Esc**, then typing `:wq`

4. Now, make the Cloud Shell script executable using following command:

   ```
   $ chmod +x daily-backup.sh
   ```

5. Test that the script uploads the test files from earlier:

   ```
   $ ./daily-backup.sh
   ```

Run the Script on a Schedule

6. Set up a `cronjob` to create a new file named the datetime and run the backup script.

```
$ sudo crontab -e
```

This will take you into vim. Press `i` to enter insert mode. Enter the following two lines. The first creates a file named the datetime every two minutes. The second runs the backup script with the OCI CLI command.

```
*/2 * * * * touch /home/opc/backupdir/$(date +'\%Y.\%m.\%d-\%H:\%M').log
*/5 * * * * /bin/bash /home/opc/daily-backup.sh
```

Press **Esc** to exit insert mode, then type `:wq` to write and quit.

7. To verify cronjob is set properly, execute the following command:

```
$ crontab -l
```

You should see the `cronjob` you just wrote.

8. To restart the `crond` service, execute one of the following commands:

```
$ sudo service crond restart
```

9. Wait for five minutes. List the files in the backup directory.

```
$ ls ~/backupdir/
```

10. List the files in the bucket.

```
$ oci os object list \
  --bucket-name <your-username>-IAD-OP-LAB05-1-BKT
```

11. From the navigation menu, under **Storage**, Click **Buckets**, and select your desired bucket. Locate the files in the **Objects** section.

In this lab you learned to install and configure the OCI CLI and manually or automatically back up files to Object Storage.

Resource & Configuration Management: Create a Reusable VCN Configuration with Terraform

Lab 6-1 Practices

Estimated time: 30 minutes

Get Started

Overview

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

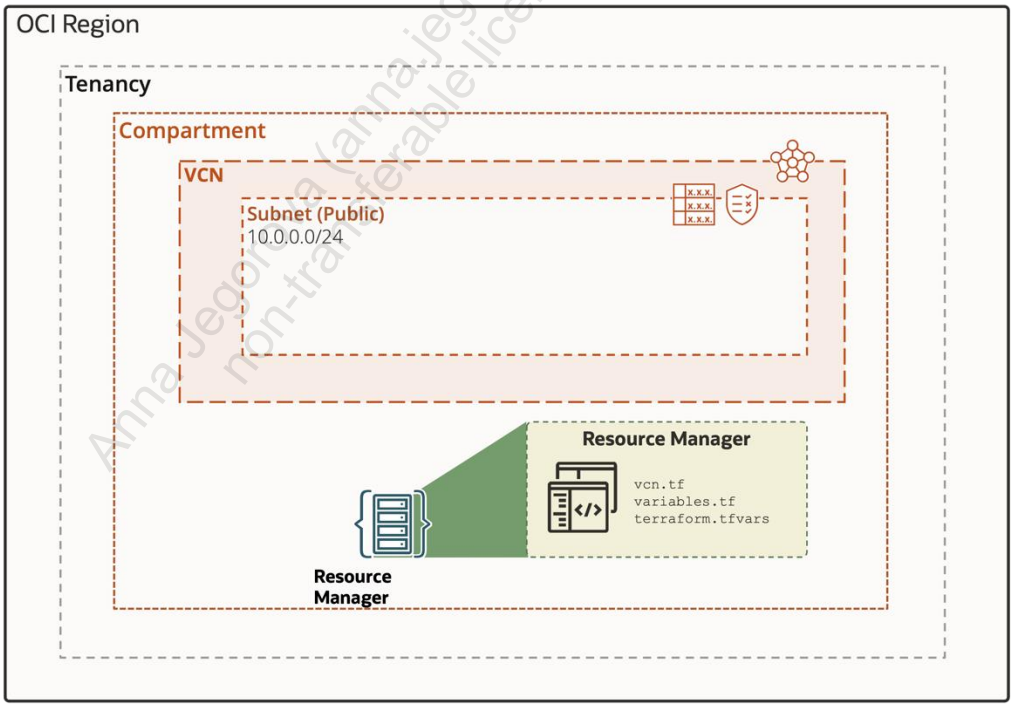
In this lab, you'll launch and destroy a VCN and subnet by creating Terraform automation scripts and issuing commands in Code Editor. Next, you'll download those Terraform scripts and create a stack by uploading them into Oracle Cloud Infrastructure Resource Manager. You'll then use that service to launch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

In this lab, you'll:

a. Create a Terraform folder and file in Code Editor.

b. Create and destroy a VCN using Terraform.

c. Create and destroy a VCN using Resource Manager.



Prerequisites

- The required IAM policies have already been set up for you.

Create a Terraform Folder and File in Code Editor

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

Tasks

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

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

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

11. Enter `cd terraform_vcn/` to change to that directory with.

12. Use `terraform init` to initialize this directory for Terraform.

13. Use `ls -a` and you should see that Terraform created a hidden directory and file.

Create and Destroy a VCN Using Terraform

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

Tasks

Write the Terraform

1. Open the OCI Provider documentation in the Terraform Registry to familiarize yourself with the OCI Terraform provider. As you go along the lab, it may be helpful to try and find the relevant portions of the documentation.

2. Add the following code block to your Terraform script to declare a VCN, replacing *<your_compartment_ocid>* with the proper OCID. The only *strictly* required parameter is the compartment OCID, but you'll add more later.

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

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

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

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

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

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

```
resource "oci_core_vcn" "example_vcn" {
    compartment_id = "<your_compartment_ocid>"
    display_name = "IAD-OP-LAB06-1-VCN-01"
    cidr_blocks = ["10.0.0.0/16"]
}
```

5. Save the changes and run `terraform plan` again. You should see the display name and CIDR block reflected in Terraform's plan.

6. Now add a subnet to this VCN. At the bottom of the file, add the following block:

```
resource "oci_core_subnet" "example_subnet" {
    compartment_id = "<your_compartment_ocid>"
    display_name = "IAD-OP-LAB06-1-SNT-01"
    vcn_id = oci_core_vcn.example_vcn.id
    cidr_block = "10.0.0.0/24"
}
```

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

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

Add Variables

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

9. In `variables.tf`, declare a variable named `compartment_id`:

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

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

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

resource "oci_core_vcn" "example_vcn" {
    compartment_id = var.compartment_id
    display_name = "IAD-OP-LAB06-1-VCN-01"
    cidr_blocks = ["10.0.0.0/16"]
}

resource "oci_core_subnet" "example_subnet" {
    compartment_id = var.compartment_id
    display_name = "IAD-OP-LAB06-1-SNT-01"
    vcn_id = oci_core_vcn.example_vcn.id
    cidr_block = "10.0.0.0/24"
}
```

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

11. If you were to run `terraform plan` or `apply` now, Terraform would see a variable and provide you a prompt to input the compartment OCID. Instead, you'll provide the variable value in a dedicated file. Create a file named exactly `terraform.tfvars`

12. Terraform will automatically load values provided in a file with this name. If you were to use a different name, you would have to provide the file name to the Terraform CLI. Add the value for the compartment ID in this file:

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

Be sure to save the file.

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

Provision the VCN

14. Run `terraform apply` and confirm that you want to make the changes by entering **yes** at the prompt.

15. Navigate to VCNs in the console. Ensure that you have the right compartment selected. You should see your VCN. Click its name to see the details. You should see its subnet listed.

Terminate the VCN

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

Create and Destroy a VCN Using Resource Manager

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

Tasks

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

Create a Stack

2. Navigate to Resource Manager in the Console's navigation menu under **Developer Services**. Go to the **Stacks** page.

3. Click **Create stack**.

    a. The first page of the form will be for stack information.

        1) For the origin of the Terraform configuration, keep **My configuration** selected.

        2) Under **Stack configuration**, upload your `terraform_vcn` folder.

        3) Under **Custom providers**, keep **Use custom Terraform providers** deselected.

        4) Name the stack and give it a description.

        5) Ensure that your compartment is selected.

        6) Click **Next**.

    b. The second page will be for variables.

        1) Because you uploaded a `terraform.tfvars` file, Resource Manager will auto-populate the variable for compartment OCID.

        2) Click **Next**.

c. The third page will be for review.

1) Keep **Run apply** deselected.

2) Click **Create**. This will take you to the stack's details page.

Run a Plan Job

4. The stack itself is only a bookkeeping resource—no infrastructure was provisioned yet. You should be on the stack's page. Click **Plan**. A form will pop up.

   a. Name the job `RM-Plan-01`.

   b. Click **Plan** again at the bottom to submit a job for Resource Manager to run `terraform plan`. This will take you to the job's details page.

5. Wait for the job to complete, and then view the logs. They should match what you saw when you ran Terraform in Code Editor.

Run an Apply Job

6. Go back to the stack's details page (use the breadcrumbs). Click **Apply**. A form will pop up.

   a. Name the job `RM-Apply-01`.

   b. Under **Apply job plan resolution**, select the plan job we just ran (instead of "Automatically approve"). This makes it execute based on the previous plan, instead of running a new one.

   c. Click **Apply** to submit a job for Resource Manager to run `terraform apply`. This will take you to the job's details page.

7. Wait for the job to finish. View the logs and confirm that it was successful.

View the VCN

8. Navigate to VCNs in the Console through the navigation menu under **Networking** and **Virtual Cloud Networks**.

9. You should see the VCN listed in the table. Click its name to go to its **Details** page.

10. You should see the subnet listed.

Run a Destroy Job

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

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

Resource & Configuration Management: Replicate an Existing Environment

Lab 7-1 Practices

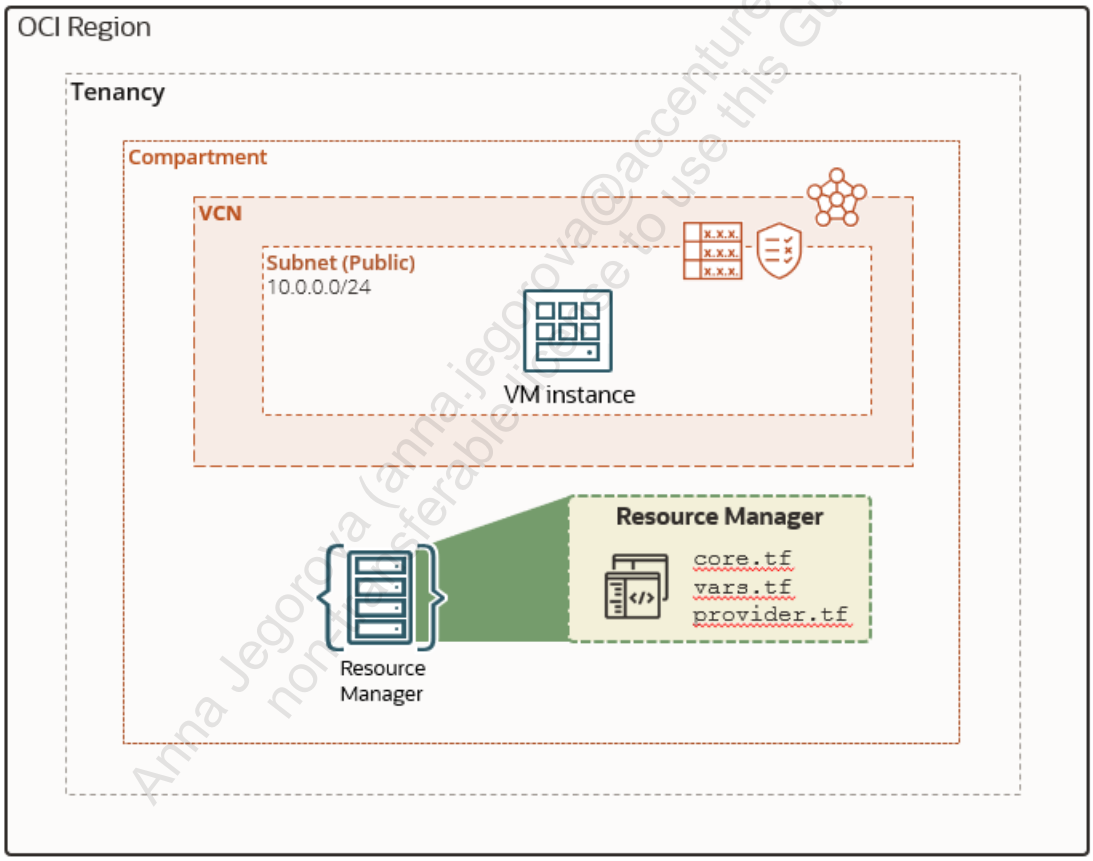
Estimated Time: 45 minutes

Get Started

Overview

Resource Manager's resource discovery allows you to generate Terraform based on existing infrastructure. This allows use cases such as manually provisioning infrastructure during a development cycle, then moving to Terraform for a deployment cycle. It also enables use cases such as migrating environments between regions or replicating environments for different purposes (for example, development, QA, or production).

In this lab, you'll first manually provision a VCN with a compute instance. Then, you'll use Resource Manager to generate Terraform for that infrastructure. Finally, you'll use that Terraform to replicate the VCN and compute instance.



In this lab, you'll:

a. Generate Terraform with Resource Manager.

b. Edit auto-generated Terraform.

c. Provision infrastructure based on the auto-generated Terraform.

Set Up Lab Environment

Create a Virtual Cloud Network

Use the VCN Wizard to set up a VCN with internet connectivity.

1. In the navigation menu, navigate to **Networking**, and click **Virtual Cloud Networks**.
2. Confirm that you are in the proper compartment at the left.
3. Click **Start VCN Wizard**. Select **Create a VCN with Internet Connectivity**.
4. Under **Basic Information**, name the VCN `IAD-OP-LAB07-1-VCN-01`, replacing `IAD` with your appropriate region code.
5. Under **Configure VCN and Subnets**, leave the CIDR blocks and DNS resolution option as the defaults.
6. Click **Next** and confirm the information on the next screen.
7. Click **Create**. Wait for all of the resources to finish provisioning.
8. Click **View Virtual Cloud Network**.

Create a Compute Instance

You'll now create an instance in your public subnet.

1. In the navigation menu, click **Compute**, then **Instances** to go to the compute instances table.
2. Click **Create instance**, and fill in the fields:
   a. Name the instance `IAD-OP-LAB07-1-VM-01`, replacing `IAD` with your appropriate region code.
   b. Under **Create in compartment**, ensure that your compartment is selected.
   c. In the **Placement** box, select **AD 1** for the availability domain.

d. In the **Image and Shape** box:

   1) For **Image**, select **Oracle Linux 8.x** (latest version).

   2) For **Shape**, select **VM.Standard.A1.Flex** (in the Ampere shape series) and configure it with **1 OCPU** and **6 GB memory**.

e. In the **Networking** box:

   1) Select your existing VCN, `IAD-OP-LAB07-1-VCN-01`.

   2) Select your existing public subnet, `Public Subnet-IAD-OP-LAB07-1-VCN-01`.

   3) Assign a public IP address.

f. Generate or upload SSH keys.

g. In the **Boot volume** box, leave all boxes as default.

h. Click **Create**. Wait for the instance to transition to the *RUNNING* state.

Replicate an Existing Environment with Resource Manager

In this practice, you'll create a stack in Resource Manager based on an existing compartment. You'll use this stack to generate a Terraform configuration that describes the compartment's resources. Finally, you'll update the Terraform files to make them reusable.

Create a Stack from Existing Infrastructure

1. In the navigation menu, click **Developer Services**, then click **Stacks** under **Resource Manager**.
2. Click **Create stack**.
    a. For the origin of the Terraform configuration, select **Existing compartment**.
    b. Under **Stack configuration**:
        1) Select your compartment.
        2) Select your region.
        3) Choose **Selected** for **Terraform provider services**.
        4) For **Services**, select **core**. Core services include Compute, Block Volume, and Networking.
    c. Do not use a custom Terraform provider.
    d. Name the stack `IAD-OP-LAB07-1-Stack-1`.
    e. Add a short description.
    f. Ensure that your compartment is selected for **Create in compartment**.
    g. Click **Next** to progress from **Stack information** to **Configure variables**. There will be no variables to configure.
    h. Click **Next** to progress from **Configure variables** to **Review**.
    i. Confirm that the only service listed for **Terraform provider services** is **core**.
    j. Click **Create**.

3. Wait for the stack to finish creating. It will query each of the selected services for the resources in your compartment.

Download Terraform Configuration

1. Under the **Stack information** tab, click the download link for the Terraform configuration.

2. There will be three files in the downloaded folder: `vars.tf`, `provider.tf`, and `core.tf`. Open `core.tf`.

3. Scroll through the code and identify different resources, such as the VCN and compute instance. The VCN's Terraform should look similar to this:

```
resource oci_core_vcn export_IAD-OPS-LAB07-1-VCN-1 {
  #cidr_block = <<Optional value not found in discovery>>
  cidr_blocks = [
    "10.0.0.0/16",
  ]
  compartment_id = var.compartment_ocid
  defined_tags = {
    Owner.CreatedOn     = <Time created>
    Owner.Creator       = <Your username>
    Owner.PrincipalType = user
  }
  display_name = IAD-OPS-LAB07-1-VCN-1
  dns_label    = iadopslab071vcn
  freeform_tags = {
    VCN = #<auto-generated tag>
  }
  ipv6private_cidr_blocks = [
  ]
  #is_ipv6enabled = <<Optional value not found in discovery>>
}
```

Edit the Terraform Configuration to Be Reusable

Before you can reupload the Terraform generated by resource discovery, there are several fields that need to be removed. This is because resource discovery documents properties such as certain public IPs, private IPs, and instance launch options that are not manually configurable.

1. Remove the exported private IP for the compute instance. For this configuration to be reusable, it needs to allow OCI to allocate a new private IP to the instance instead of referring to the existing VNIC.

a. Find the resource block of type `oci_core_private_ip`. It should look similar to this:

```
resource oci_core_private_ip export_IAD-OP-LAB07-1-VM-01 {
  defined_tags = {
    "Oracle-Tags.CreatedBy" = <Your username>
    "Oracle-Tags.CreatedOn" = <Time created>
  }
  display_name = "IAD-OP-LAB07-1-VM-01"
  freeform_tags = {
  }
  hostname_label = "iad-op-lab07-1-vm-01"
  ip_address     = <The IP that OCI assigned>
  #vlan_id = <<Optional value not found in discovery>>
  vnic_id = <The VNIC OCID>
}
```

b. Comment or delete this entire block.

2. Remove the IP address, NVME count, and launch options from the compute instance.

a. Find the compute instance block (it will be of type `oci_core_instance`).

b. Inside of the compute instance block, find the field for `private_ip`. It should look similar to this:

```
resource oci_core_instance export_IAD-OP-LAB07-1-VM-01_1 {
  <Various fields…>
  private_ip = <The IP that OCI assigned>
  <Various fields…>
}
```

c. Comment or delete the specific line for `private_ip`, (bolded above).

d. Inside of the compute instance block, find the fields for `launch_options`. It should look similar to this:

```
resource oci_core_instance export_IAD-OP-LAB07-1-VM-01_1 {
  <Various fields…>
  launch_options {
    boot_volume_type                    = "PARAVIRTUALIZED"
    firmware                            = "UEFI_64"
    is_consistent_volume_naming_enabled = "true"
    is_pv_encryption_in_transit_enabled = "true"
    network_type                        = "PARAVIRTUALIZED"
    remote_data_volume_type             = "PARAVIRTUALIZED"
  }
  <Various fields…>
}
```

e. Comment or delete this entire block of configurations (bolded above).

f. Inside of the compute instance block, find the fields for shape_config. It should look similar to this:

```
resource oci_core_instance export_IAD-OP-LAB07-1-VM-01_1 {
  <Various fields…>
  shape_config {
    baseline_ocpu_utilization = ""
    memory_in_gbs             = "6"
    nvmes                     = "0"
    ocpus                     = "1"
  }
  <Various fields…>
}
```

g. Comment or delete the lines for `nvmes` and `baseline_ocpu_utilization` (bolded above).

3. Remove the public IP address reference from the NAT gateway.

a. Find the block for the NAT gateway. It will look similar to this:

```
resource oci_core_nat_gateway export_NAT-Gateway-IAD-OP-LAB07-1-VCN-1 {
  block_traffic  = "false"
  compartment_id = var.compartment_ocid
  defined_tags = {
    "Oracle-Tags.CreatedBy" = <Your username>
    "Oracle-Tags.CreatedOn" = <Time created>
  }
  display_name = "NAT Gateway-IAD-OP-LAB07-1-VCN-1"
  freeform_tags = {
    "VCN" = <Time created>
  }
  public_ip_id = <The public IP that OCI assigned>
  #route_table_id = <<Optional value not found in discovery>>
  vcn_id = oci_core_vcn.export_IAD-OP-LAB07-1-VCN-1.id
}
```

b. Comment or delete the line for the `public_ip_id` (bolded above).

Create a New Stack from the Terraform

1. In the navigation menu, under **Developer Services**, click **Stacks** (under **Resource Manager**).

2. Click **Create stack**.

   a. The first page of the form will be for stack information.

      1) For the origin of the Terraform configuration, keep **My configuration** selected.

      2) Under **Stack configuration**, select **Folder** as the source, and upload the folder containing `vars.tf, provider.tf, and core.tf` (move them to their own folder if needed).

      3) Under **Custom providers**, keep **Use custom Terraform providers** deselected.

      4) Name the stack `IAD-OP-LAB07-1-Stack-2`

      5) Give it a short description.

      6) Ensure that your compartment is selected

      7) Click **Next**.

   b. The second page will be for variables.

      1) The variables should all be automatically populated.

      2) Click **Next**.

   c. The third page will be for review.

      1) Select **Run apply**.

      2) Click **Create**. This will take you to the stack's details page.

   d. Wait for the apply job to finish.

3. View the new VCN in the navigation menu under **Networking,** then **Virtual Cloud Networks**. It will have the same name as the previous one, but this could have been changed in the Terraform to a variable.

4. View the new compute instance in the navigation menu under **Compute**, then **Instances**. It will have the same name as the previous one, but this could have been changed in the Terraform to a variable.

Resource and Configuration Management: Use Ansible to Deploy Apache Application to Multiple Instances

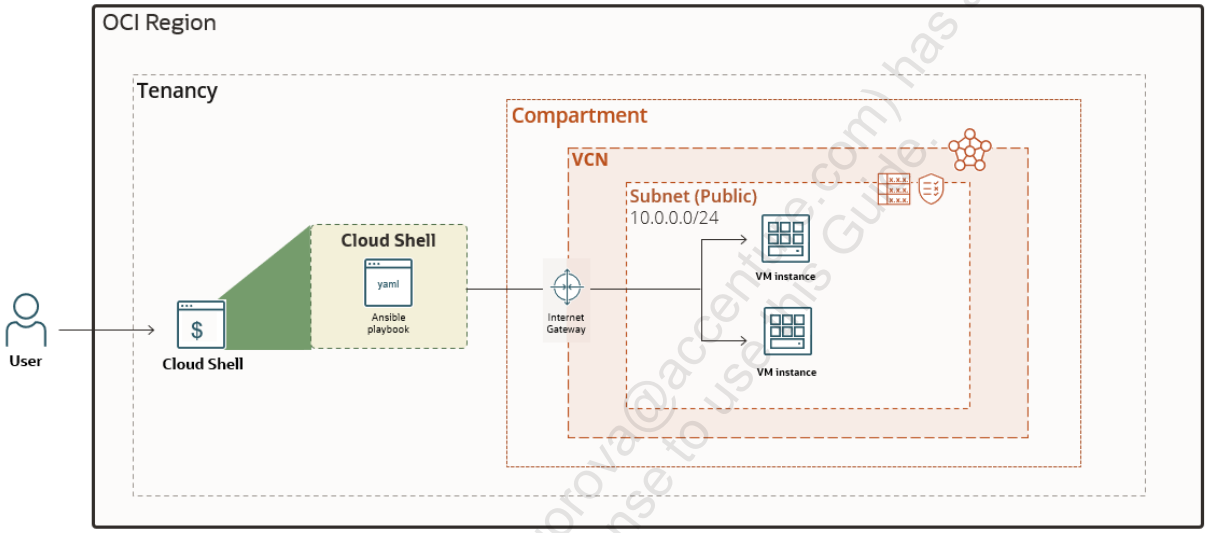
Lab 8-1 Practices

Estimated Time: 45 minutes

Get Started

Overview

Oracle Cloud Infrastructure automation helps configure resources in a faster and more efficient way. In this lab, you will write an automation script to install and configure an httpd webserver. You will then write an Ansible playbook to spin up Apache applications and deploy them to two compute instances.



In this lab, you'll:

a. Launch Code Editor and configure Ansible.

b. Write an Ansible playbook to install and configure Apache hosts.

Prerequisites

- The required IAM policies have been set up for you.

Assumptions

- Learner knows basic Linux commands.

Set Up Lab Environment

For this lab, you will need to create a VCN with two compute instances in a public subnet.

Create VCN

First, you will create a VCN to house the compute instances.

1. In the navigation menu, click **Networking**, then click **Virtual Cloud Networks**.
2. Confirm that you are in the proper compartment at the left.
3. Click **Start VCN Wizard**.
    a. Name the VCN `IAD-OP-LAB08-1-VCN-01`.
    b. Leave the CIDR blocks as their defaults.
    c. Click **Next** and confirm the information on the next screen.
    d. Click **Create**. Wait for all of the resources to finish creating.
    e. Click **View Virtual Cloud Network** to go to its details page.
2. On the left navigation bar under Resources, click **Security Lists** to view the VCN's security lists.
3. Click the **Default Security List** to go to its details page.
4. Here you need to open port 80 for HTTP. Click Add Ingress Rule to open a form.
    a. Leave the box for **Stateless** deselected.
    b. Leave **Source Type** as **CIDR**.
    c. For **Source CIDR**, enter `0.0.0.0/0`
    d. For **IP Protocol**, select **TCP**.
    e. Leave **Source Port Range** empty (this indicates **All**).
    f. For **Destination Port Range**, enter `80`
    g. Click **Add Ingress Rules** at the bottom.

Create SSH Keys for Cloud Shell

Note: If your Cloud Shell VM already has an SSH key pair and you have downloaded the public key, you can skip to creating the compute instances. You are likely to have done this in previous labs.

1. Open Cloud Shell using the icon at the top-right corner.
2. Use the following command in Cloud Shell to generate an SSH key pair (remember not to include the `$`):
   ```
   $ ssh-keygen -t rsa -b 4096
   ```
3. It will prompt you for the location to save the files. Leave it empty and press **Enter** to use the default.
4. It will prompt you for a password. For this lab, leave it empty and press **Enter** to skip. Press **Enter** again to confirm.
5. Add your keys to the ssh-agent with the following two commands:
   ```
   $ eval "$(ssh-agent -s)"
   $ ssh-add ~/.ssh/id_rsa
   ```
6. Download the public key:
   a. Click the gear at the top-right corner of Cloud Shell.
   b. Click **Download**.
   c. Enter `.ssh/id_rsa.pub` to specify the public key. Click **Download** to confirm. This is the key you will upload to compute instances when provisioning them.
7. Close Cloud Shell.

Create the First Compute Instance

1. In the navigation menu, click **Compute**, then **Instances** to go to the compute instances table.
2. Click **Create instance**, and fill in the fields:
   a. Name the instance `IAD-OP-LAB08-1-VM-01`, replacing `IAD` with your appropriate region code.

b. Under **Create in compartment**, ensure that your compartment is selected.

c. In the **Placement** box, select **AD 1** for the availability domain.

d. In the **Image and Shape** box:

   1) For **Image**, select **Oracle Linux 8.x** (latest version).

   2) For **Shape**, select **VM.Standard.A1.Flex** (in the Ampere shape series) and configure it with **1 OCPU** and **6 GB memory**.

e. In the **Networking** box:

   1) Select your existing VCN, `IAD-OP-LAB08-1-VCN-01`.

   2) Select your existing public subnet, `Public Subnet-IAD-OP-LAB08-1-VCN-01`.

   3) Assign a public IP address.

f. Upload the SSH key generated earlier.

g. In the **Boot volume** box, leave all boxes as default.

h. Click **Create**.

Create the Second Compute Instance

1. Repeat steps 6 and 7 above to create a compute instance named `IAD-OP-LAB08-1-VM-01`, replacing `IAD` with your appropriate region code.

   a. Name the instance `IAD-OP-LAB08-1-VM-02`, replacing `IAD` with your appropriate region code.

   b. Under **Create in compartment**, ensure that your compartment is selected.

   c. In the **Placement** box, select **AD 1** for the availability domain.

   d. In the **Image and Shape** box:

      1) For **Image**, select **Oracle Linux 8.x** (latest version).

2) For **Shape**, select **VM.Standard.A1.Flex** (in the Ampere shape series) and configure it with **1 OCPU** and **6 GB memory**.

e. In the **Networking** box:

1) Select your existing VCN, `IAD-OP-LAB08-1-VCN-01`.

2) Select your existing public subnet, `Public Subnet-IAD-OP-LAB08-1-VCN-01`.

3) Assign a public IP address.

f. Upload the SSH key generated earlier.

g. In the **Boot volume** box, leave all boxes as default.

h. Click **Create**. Wait for the instance to transition to the *RUNNING* state.

Launch Cloud Shell and Configure Ansible Clients

In this practice, you will set up Ansible clients, that you will install and configure webserver.

Open Code Editor

1. Log in to your tenancy in the Cloud Console and open the Code Editor, whose icon is at the top-right corner, to the right of the CLI Cloud Shell icon.
2. Expand the Explorer panel with the top icon on the left panel. It looks like two overlapping documents.
3. Expand the drop-down list for your home directory if it isn't already expanded. It's okay if it is empty.
4. Create a new folder by clicking **File**, then **New Folder**, and name it `ansible_apache_lab`.

Configure Ansible Hosts

1. Create a file in that folder by clicking **File**, then **New File**, and name it `hosts.yaml`. To make Code Editor create the file in the correct folder, click the folder name in your home directory to highlight it.
2. In this file, you will record the public IP addresses of the VMs for Ansible. Add the following code, replacing the two IP addresses in the code with the ones from the VMs you provisioned earlier.

```
all:
  hosts:
    <First-IP-address>:
    <Second-IP-address>:
```

Note: Make sure that you follow the indentation properly, because YAML files are sensitive to code indentation.

3. Save the changes by clicking **File**, then **Save**.

Write the Ansible Playbook

Note: The code below is added piece by piece to explain the process, but the entire code can be added at once. You can copy it from this page into a text editor to ensure that invisible characters aren't present and line breaks are Linux-style.

1. Create another file in `ansible_apache_lab` by clicking **File**, then **New File**, and name it `playbook.yaml`. To make Code Editor, create the file in the correct folder, click the folder name in your home directory to highlight it.

2. Enter the following code to install and start Apache on the VMs.

```
- name: Install and Configure web servers
  hosts: all
  remote_user: opc
  tasks:
  - name: Ensure apache is at the latest version
    ansible.builtin.yum:
      name: httpd
      state: latest
    become: yes
  - name: Ensure apache is running
    ansible.builtin.service:
      name: httpd
      state: restarted
    become: yes
```

3. Open a terminal panel in Code Editor by right-clicking `ansible_apache_lab` in the Explorer panel and selecting **Open in terminal**. In the terminal, execute the playbook:

```
$ ansible-playbook -i ~/ansible_apache_lab/hosts.yaml
~/ansible_apache_lab/playbook.yaml
```

4. It will prompt you to confirm that you want to connect. Type `yes` and press **Enter** twice. (Once for each machine). The two prompts may be intertwined.

   The output should look similar to this (IP addresses have been censored):

```
TASK [Ensure apache is at the latest version] ***********************************************************
changed: [redacted]
changed: [redacted]

TASK [Ensure apache is running] *************************************************************************
changed: [redacted]
changed: [redacted]

PLAY RECAP **********************************************************************************************
[redacted]                 : ok=3    changed=2    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
[redacted]                 : ok=3    changed=2    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

5. Now, both VMs should have Apache installed, but you still need to open the Linux firewall to HTTP traffic. Edit the YAML script to configure the firewall to allow incoming HTTP traffic. Add the following block of code to `playbook.yaml`. Note that it's indented to be one of the `tasks`:

```
    - name: Permit traffic in default zone for http service
      ansible.posix.firewalld:
        service: http
        permanent: yes
        state: enabled
        immediate: true
      become: yes
```

6. Save and execute the playbook again in the terminal:

```
$ ansible-playbook -i ~/ansible_apache_lab/hosts.yaml
~/ansible_apache_lab/playbook.yaml
```

The output should look similar to this (IP addresses have been censored):

7. To test whether the webserver is running, enter the public IP addresses of the two Ansible clients into your browser.

You should get a webpage that looks like this:

Apache 2 Test Page
powered by the **Apache httpd server**

This page is used to test the proper operation of the Apache HTTP server after it has been installed. If you can read this page it means that the Apache HTTP server installed at this site is working properly.

If you are a member of the general public:

The fact that you are seeing this page indicates that the website you just visited is either experiencing problems or is undergoing routine maintenance.

If you would like to let the administrators of this website know that you've seen this page instead of the page you expected, you should send them e-mail. In general, mail sent to the name "webmaster" and directed to the website's domain should reach the appropriate person.

For example, if you experienced problems while visiting www.example.com, you should send e-mail to "webmaster@example.com".

If you are the website administrator:

You may now add content to the directory `/var/www/html/`. Note that until you do so, people visiting your website will see this page and not your content. To prevent this page from ever being used, follow the instructions in the file `/etc/httpd/conf.d/welcome.conf`.

You are free to use the images below on Apache Linux powered HTTP servers. Thanks for using Apache!

Security Services: Configure Vulnerability Scanning with Cloud Guard

Lab 9-1 Practices

Estimated time: 30 mins

Get Started

Overview

Oracle Cloud Infrastructure Vulnerability Scanning Service improves your security posture by checking hosts for potential vulnerabilities on a regular schedule. The service provides comprehensive visibility into misconfigured or vulnerable resources and creates reports with metrics and information about these vulnerabilities, including remediation information, for developers, operations, and security administrators.

In this lab, you will:

a. Create a Virtual Cloud Network.

b. Create a Compute Instance.

c. Create Scan Recipe.

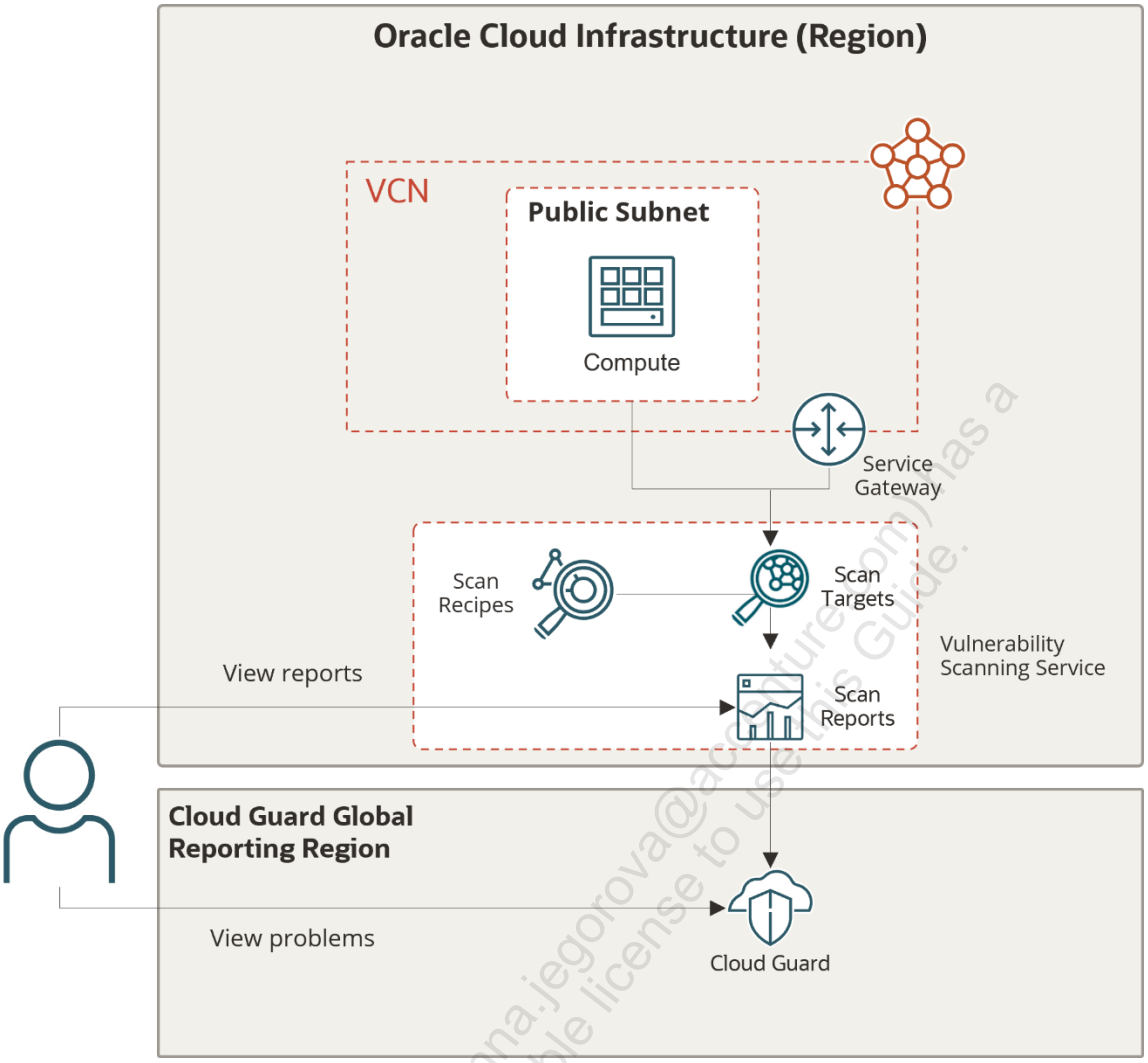
d. Create Vulnerability Scanning Target.

e. View Scan result.

f. View Vulnerability Reports.

g. Configure Cloud Guard.

h. View Vulnerability Scanning problem in Cloud Guard.



Prerequisites

- You must have access to the OCI Console.
- The Oracle University lab team set up all the IAM policies required for you to complete this lab.

Assumptions

- Select the region that's available in the tenancy allotted to you. In this lab, we are considering US East (Ashburn) (IAD) as your region.
- You must be familiar with navigating the OCI Console.

Create a Virtual Cloud Network

In this practice, you will create a Virtual Cloud Network in OCI with a public and a private subnet using the VCN wizard. The compute instance that you will create later will be hosted in this VCN's public subnet.

Note: If you have already created VCN in the previous practice and have it in your compartment, you can skip this practice.

Tasks

1. Open the navigation menu and click **Networking**. Under Networking, click **Virtual Cloud Network**.
2. In the left navigation pane, under **List Scope**, Select the assigned **compartment** from the drop-down menu.
3. Click **Start VCN Wizard.**
4. Select **Create VCN with Internet Connectivity** and click **Start VCN Wizard.**
5. On the Configuration page, enter the following:
   a. **Name**: `IAD-OP-LAB09-1-VCN-01`
   b. **Compartment:** Select the <*compartment name*> assigned to you.

      Note: Leave all the other options in their default setting.
   c. Click **Next.**
   d. Verify the details on the **Review and Create** page.
6. Click **Create** to start creating the VCN and its resources.
7. Click **View Virtual Cloud Network** to verify the creation of the VCN and its resources.

   You can now see that the VCN was successfully created and is in the Available state, with the following components:

   VCN, Public subnet, Private subnet, Internet gateway, NAT gateway, and Service gateway.

Create a Compute Instance

In this practice, you will provision compute instances with vulnerability scanning plugin enabled.

Note: If you already created/provisioned a compute instance in the previous practice and have it in your compartment, you can skip this practice. Refer to step 7 in this task to enable the vulnerability scanning plugin.

Tasks

1. Open the navigation menu and click **Compute**. Under Compute, click **Instances**.
2. In the left navigation pane, under **List Scope**, select the assigned **compartment** from the drop-down menu.
3. Click **Create Instance**. On the **Create Instance** dialog box, provide the following details:
   a. **Name**: `IAD-OP-LAB09-1-VM-01`
   b. **Create in compartment**: Select the <*compartment name*> assigned to you.
   c. **Placement:** Select Availability Domain AD1
   d. Note: If Service limit error is displayed, choose a different Availability Domain.
   e. **Image**: Select the image Oracle Linux 8.
   f. **Shape:** Click **Change shape**.
   g. In the **Browse all shapes** dialog box:
      1) Select the **Ampere** shape series.
      2) Select **VM.Standard.A1.Flex**.
      3) Keep **1 OCPU** and **6 GB** memory selected.
      4) Click **Select shape**.
   h. **Networking**: Pick your VCN **IAD-OP-LAB09-1-VCN-01** and Public Subnet.
   i. **Public IP address** – Assign a public IPv4 address.

j. **Generate (or upload) SSH Keys**:

1) Click **Generate a key pair for me**.

2) Click **Save private key**. This will save the private key to your local workstation.

3) Click **Save public key**. This will save the public key to your local workstation.

4) Select the **Upload public key (.pub)** option button. Upload the recently downloaded public key. Either drag the key to the **Drop .pub files here** window, or click **Browse**, select the key and click Upload.

Note: Leave all the other options in their default setting.

4. Click **Show Advanced Options**.

5. On the **Oracle Cloud Agent** tab, select the **Vulnerability Scanning** check box.

6. Click **Create.**

Note: After a couple of minutes, you can see that the instance is successfully created, and the state is Running.

7. On Instance details page, click the **Oracle Cloud Agent** tab.

8. Toggle the **Enable Plugin** switch to **Enabled** for **Vulnerability Scanning** plugin Name, if the switch is disabled.

It can take up to 5-10 minutes for the change to take effect. After a few moments, the status **Running** for Vulnerability Scanning enabled service will be displayed.

Create Scan Recipe

In this practice, you will use Oracle Cloud Infrastructure Vulnerability Scanning Service to create and manage recipes that scan target compute instances (hosts) for potential security vulnerabilities.

Tasks

1. Open the navigation menu and click **Identity & Security**. Under **Scanning**, click **Scan Recipes.**
2. In the left navigation pane, under **List Scope**, select your assigned **compartment** from the drop-down menu.
3. Click the **Hosts** tab, and then click **Create.**
4. On the **Create scan recipe** dialog box, enter the following:
   a. **Type**: Select **Compute.**
   b. **Name:** `IAD-OP-LAB09-1-CSC-01`
   c. **Create in compartment**: Select the <*compartment name*> assigned to you.
   d. **Public IP port scanning:** Select **Standard (Top 1000 ports).**
   e. Select the **Agent based scanning** check box.
   f. Under **Agent based scanning**, configure CIS benchmark scanning.
      1) **CIS benchmark profile**: Select **Strict (More than 20% of the benchmarks failing is a critical risk).**
      2) Deselect the **Enable file scans** check box.
   g. **Schedule**: Select **Daily.**
5. Click **Create scan recipe.**

   You will see that the scan recipe is successfully created, and the status is Active.

Create Vulnerability Scanning Target

In this practice, you will use Oracle Cloud Infrastructure Vulnerability Scanning Service to create compute (host) targets and to assign them to compute scan recipes. A target is a collection of instances that you want routinely scanned for security vulnerabilities.

Tasks

1. Open the navigation menu and click **Identity & Security**. Under **Scanning**, click **Targets.**
2. In the left navigation pane, under **List Scope**, select your assigned **compartment** from the drop-down menu.
3. Click the **Hosts** tab, and then click **Create.**
4. In the **Create target** dialog box, enter the following:
   a. **Type**: Select **Compute.**
   b. **Name:** `IAD-OP-LAB09-1-CTRG-01`
   c. **Create in compartment**: Select the <*compartment name*> assigned to you.
   d. **Description**: Add meaningful description.
   e. **Scan recipe in**: Select **IAD-OP-LAB09-1-CSC-01.**

      Note: Click **Change compartment** and select assigned compartment to locate scan recipe, if not available by default.
   f. **Target compartment**: Select the <*compartment name*> assigned to you.
   g. Under **Targets**:
      1) Choose **Selected compute instances in the selected target compartment**.
      2) Targets: Select **IAD-OP-LAB09-1-VM-01** instance as target.
5. Click **Create target**.

   You will see that the target is successfully created, and the status is Active.

   Scanning service may take up to 10-15 minutes to check your compute instance for security vulnerabilities and open ports, based on the parameters and schedule configured in the scan recipe.

View Scan result

In this practice, you will view and explore security vulnerabilities discovered in your compute instance, such as open ports, critical OS patches, and failed benchmark tests.

Tasks

1. Open the navigation menu and click **Identity & Security**. Under **Scanning**, click **Scanning Reports**.
2. In the left navigation pane, under **List Scope**, select your assigned **compartment** from the drop-down menu.
3. Click the **Hosts** tab.
4. Locate the **Risk level** filter drop-down menu. Select **All**.
5. Locate the **Scan start date** and **Scan end date** filter drop-down menus.

   By default, only the most recent scan reports are displayed. To view older reports, choose specific start and end dates.
6. Locate the **Reset** button. Click **Reset** at any time to set the risk level and date ranges back to the default values.
7. (Optional) Click the table columns to sort the container image scans by:
   - Risk level
   - Issues found
   - Scan completed
8. To view a compute scan report, click the name of the compute instance.

   Example: IAD-OP-LAB09-1-VM-01

   A host scan includes metrics, open ports, vulnerabilities, and benchmarks for a selected Compute instance.
9. In the left navigation pane, under **Resources**, click **Metrics** if not already selected.
   a. In Host scan information tab, locate the number of **CIS benchmarks passed**.

b. The Vulnerabilities panel shows the number of security vulnerabilities of each risk level that were detected during the most recent scan of the selected compute instance.

10. In the left navigation pane, under **Resources**, click **Open ports**.

    a. The first panel shows the number of open ports that are detected on each Virtual Network Interface Card (VNIC) in the selected compute instance.

    b. The second panel shows the specific port numbers that were detected in this compute instance.

11. In the left navigation pane, under **Resources**, click **Vulnerabilities**.

12. The following details are shown for each issue that were detected in the selected compute instance:
    - CVE ID
    - Risk level
    - CVE description
    - Last detected
    - First detected
    - Cause and remediation

    Click any **View detail** button in the Cause and remediation column to get more information on how to address this vulnerability.

13. In the left navigation pane, under **Resources**, click **CIS benchmarks**.

14. The following details are shown for each CIS benchmark the Scanning service tested on the selected compute instance:
    - Benchmark ID
    - Result - Pass or Fail
    - Summary

View Vulnerability Reports

In this practice, you will view and explore Vulnerability Reports, accessing information about specific vulnerabilities that were detected in compute instance targets.

Tasks

1. Open the navigation menu and click **Identity & Security**. Under **Scanning**, click **Vulnerability Reports**.
2. In the left navigation pane, under **List Scope**, Select your assigned **compartment** from the drop-down menu.
3. In the left navigation pane, under **Filters**, select the Risk level, **All.**
4. Click the **Risk level** column header to sort by risk level.
5. To view a description of a specific vulnerability, click **Show** in the CVE description column.
6. To view details about a specific vulnerability, click a reported **CVE ID**.

   Example: CVE-2022-40674 under CVE ID column.

   This will show a vulnerability report for the selected CVE, which includes details about the affected resources and CVE information.
7. On the Vulnerabilities report page, under **Vulnerability information** tab, click the **CVE ID** link.

   It will redirect to the source of the CVEs database and provide more information about it.
8. In the left navigation pane, under **Resources**, select **Hosts** to view a list of compute instance that are affected by the selected vulnerability.

Configure Cloud Guard

In this practice, you will configure and use Cloud Guard to monitor security problems detected in Vulnerability Scanning.

Note: Before using Cloud Guard, at least one Scanning target must exist before the Scanning service creates any reports. These reports are used by the Cloud Guard detector.

Tasks

1. Open the navigation menu and click **Identity & Security**. Under **Identity & Security**, click **Cloud Guard**.
2. In the left navigation pane, under **Scope**, Select the ***<Tenancy Name>*root compartment** from the drop-down menu.
3. In the left navigation pane, under **Cloud Guard**, click **Detector Recipes.**
4. Click **OCI Configuration Detector Recipe (Oracle managed)**.

   View the detector rules that are included in this recipe.
5. Under **Detector Rules**, in the **Filter by detector rule** field, enter `scan`.
   a. Verify that the following Vulnerability Scanning rules are enabled:
      - Scanned container image has vulnerabilities
      - Scanned host has vulnerabilities
      - Scanned host has open ports
6. In the left navigation pane, under **Cloud Guard**, click **Targets.**

   Note: If you already have a specific target set for your compartment, delete it.
7. Click **Create New Target**.
8. Enter the following:
   a. **Target Name:** `IAD-OP-LAB09-1-CG-01`
   b. **Description:** Enter a meaningful description.
   c. **Compartment:** Select the <*compartment name*> assigned to you**.**

d. **Configuration detector recipe:** OCI Configuration Detector Recipe (Oracle managed)

e. **Threat detector recipe:** OCI Threat Detector Recipe (Oracle managed)

f. **Activity Detector Recipe:** Oracle Activity Detector Recipe (Oracle managed)

g. **Responder recipe:** OCI Responder Recipe (Oracle managed)

9. Click **Create** to create target.

   The detail page for the new target will be displayed.

View Vulnerability Scanning Problem in Cloud Guard

In this practice, you will view and explore Cloud Guard reported security problems identified through vulnerability scanning.

Note: Before using Cloud Guard, at least one Scanning target must exist before the Scanning service creates any reports. These reports are used by the Cloud Guard detector.

Tasks

1. Open the navigation menu and click **Identity & Security**. Under **Identity & Security**, click **Cloud Guard**.

2. In the left navigation pane, under **List Scope**, Select the assigned **compartment** from the drop-down menu.

3. In the left navigation pane, under **Cloud Guard**, click **Problems.**

4. View the list of problems Cloud Guard has identified with the resources in your assigned compartment based on your previous practices. The **Problems** page displays information about each problem, including:

    - Problem Name
    - Risk Level
    - Detector Type
    - Resource affected
    - Target
    - Region
    - Labels
    - First Detected
    - Last Detected

5. To show only Vulnerability Scanning problems, set Filters to:

    Example: `Labels = VSS` (case-sensitive)

6. Click the name of a Vulnerability Scanning problem to view its details.

    Example:

    - Scanned host has vulnerabilities
    - Scanned host has open ports

Note: If no Vulnerability Scanning problems are displayed in Cloud Guard, then consider the following scenarios.

- The Vulnerability Scanning service did not create any reports yet. The schedule (daily/weekly) is configured in the Scanning target.
- You recently enabled Cloud Guard or the Vulnerability Scanning detector rules, and Cloud Guard has not run them yet.

7. You can take the necessary steps to eliminate the detected vulnerability and mark the problem as resolved.

Identity & Access Management: Create Common Policies

Lab 10-1 Practices

Estimated Time: 30 minutes

Get Started

Overview

Unlike other labs, this lab provides a scenario and asks you to determine the policies required to perform the scenario successfully. Because you can't add policies in the Cloud Console, just write them down and check them against the solution provided below.

Scenario

You are the default administrator for your software development company's Oracle Cloud Infrastructure tenancy, so you have access to all API operations and resources in that tenancy. You set up the following compartment hierarchy, with levels indicated in parenthesis.

Root (1)
- Applications (2)
  - Applications-Prod (3)
  - Applications-NonProd (3)
- Sales (2)

You must set up Identity & Access Management (IAM) policies for several groups in your company, with varying levels and types of access, depending on their role. Consider the following access requirements and how you would create policies that allow them. Assume all policies are created in the root compartment.

- A group of compute administrators (ComputeAdmins) must be able to launch compute instances and mount block volumes.
- Network administrators (NetworkAdmins) need full control over all virtual cloud networks and load balancers.
- A group of storage admins (StorageAdmins) needs full control over object storage and block volume resources, but only in the Applications-NonProd compartment.
- Two administrators (SalesAdmins) working on a separate sales project need access to all resources in the Sales compartment within the US West (Phoenix) / PHX region only.
- A group of nonadmin HR users (HR-Users) needs to download files from the bucket "HR-Log-Bucket" in the US West (Phoenix) / PHX region only.
- These nonadmin users also all require Cloud Shell access.

Assumptions

- You must construct the policies to allow the least necessary privileges required to accomplish the task.

Resources

How Policies Work: https://docs.oracle.com/en-us/iaas/Content/Identity/Concepts/policies.htm

Policy Syntax: https://docs.oracle.com/en-us/iaas/Content/Identity/Concepts/policysyntax.htm

Common Policies: https://docs.oracle.com/en-us/iaas/Content/Identity/Concepts/commonpolicies.htm

Policies for Core Services: https://docs.oracle.com/en-us/iaas/Content/Identity/Reference/corepolicyreference.htm

Solution

Because you want to provide the least possible access, use the READ or USE verbs rather than MANAGE where possible, and restrict access to a particular compartment, region, or bucket where needed.

Policies

Compute Admin

```
Allow ComputeAdmins to use virtual-network-family in tenancy

Allow ComputeAdmins to manage instance-family in tenancy

Allow ComputeAdmins to use volume-family in tenancy
```

Network Admins

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

Allow group NetworkAdmins to manage load-balancers in tenancy
```

Storage Admins

```
Allow group StorageAdmins to manage object-family in compartment
Applications:Applications-NonProd

Allow group StorageAdmins to manage volume-family in compartment
Applications:Applications-NonProd

Allow group StorageAdmins to use instance-family in compartment
Applications:Applications-NonProd
```

Phoenix Region Sales Admins

```
Allow group SalesAdmins to manage all-resources in compartment
Sales where request.region='phx'
```

Phoenix Nonadmin Users

```
Allow group HR-Users to read buckets in tenancy where all
{target.bucket.name='HR-Log-Bucket',request.region='phx'}

Allow group HR-Users to read objects in tenancy where all
{target.bucket.name='HR-Log-Bucket',request.region='phx'}

Allow group HR-Users to use cloud-shell in tenancy
```

Identity & Access Management: Write an Advanced Policy

Lab 11-1 Practices

Estimated Time: 20 minutes

Get Started

Overview

Unlike other labs, this lab provides a scenario and asks you to determine the policies required to perform the scenario successfully. Because you can't add policies in the Cloud Console, just write them down and check them against the solution provided below.

Scenario

Your team has developed a microservices-based application on Oracle Cloud Infrastructure (OCI). For high-availability and simplified bootstrapping, your team has designed the application to run in a partially degraded state. That is, each of the services can run independently and offer a limited set of features based on which other services are available.

To accomplish this, each of the microservices runs on its own compute instance and periodically queries OCI to list the other compute instances within its VCN. It could do that with this command:

```
$ oci compute instance list --compartment-id <compartment_id>
```

The OCI Compute API would respond with a list of compute instances in the compartment, including their OCIDs. For each of those compute instances, the microservice could list the VNICs attached to it with this command:

```
$ oci compute vnic-attachment list -c <compartment_id> --
instance-id <instance_ocid>
```

The OCI Compute API would then respond with the VNICs attached to each compute instance, as well as the OCID of the subnet in which they reside. For each of the subnets, the microservice could get its parent VCN with this command:

```
$ oci network subnet get --subnet-id <subnet_ocid>
```

Finally, based on the responses, the microservice could filter for the instances that are in the same VCN and use the IP from the VNIC.

Write the IAM policies that would allow the microservice to do this.

Assumptions

- Your instance is in the Ashburn (IAD) region. You want your instance to be able to view only the other instances in that region.
- Your tenancy has an additional application and instances in the Phoenix (PHX) region.

- You must construct the policies to allow the least necessary privileges required to accomplish the task.

After you have written those policies, update them so that only one instance is allowed to view all of the other instances in its region.

Resources

Managing Dynamic Groups: https://docs.oracle.com/en-us/iaas/Content/Identity/Tasks/managingdynamicgroups.htm

Advanced Policies: https://docs.oracle.com/en-us/iaas/Content/Identity/Concepts/policyadvancedfeatures.htm

How Policies Work: https://docs.oracle.com/en-us/iaas/Content/Identity/Concepts/policies.htm

Policy Syntax: https://docs.oracle.com/en-us/iaas/Content/Identity/Concepts/policysyntax.htm

Common Policies: https://docs.oracle.com/en-us/iaas/Content/Identity/Concepts/commonpolicies.htm

Solution

You will need the OCIDs of your instance and compartment to create the policies.

Dynamic Group

You need to create a dynamic group for your instance to use in the policies.

To allow any instance in a compartment to view information about other instances:

```
All {instance.compartment.id = '<compartment_ocid>'}
```

To allow a specific instance to view information about other instances (second scenario):

```
All {instance.id = '<instance-ocid>'}
```

Policies

Allow your instance to view information about instances in a compartment in its region.

```
Allow dynamic-group <dynamicgroup> to inspect instances in
compartment <compartment> where region.id=us-ashburn-1
```

Allow your VM to view information about subnets in a compartment in its region.

```
Allow dynamic-group <dynamicgroup> to inspect subnets in
compartment <compartment> where region.id=us-ashburn-1
```

Deploy Ruby on Rails with Terraform

Lab 12 Practices

Estimated time: 120 mins

Set Up a Terraform Directory in Code Editor

1. Log in to your tenancy in the Cloud Console.
2. Open Code Editor using the Developer Tools icon on the top-right corner.
3. Expand the Explorer panel with the top icon on the left panel. It looks like two overlapping documents.
4. Expand the drop-down menu for your home directory if it isn't already expanded. It's okay if this is empty.
5. Create a new folder by clicking **File**, then click **New Folder**. Name it `terraform-ror`.
6. Create a file in that folder by right-clicking the folder name and clicking **New File**. Name it `provider.tf`.
7. Paste the following code into the `provider.tf` to configure Terraform to install the OCI Provider.

```
terraform {
  required_providers {
    oci = {
      source  = "oracle/oci"
      version = ">=4.100.0"
    }
  }
  required_version = ">= 1.0.0"
}
```

8. Save the file. Right-click the folder name on the left (`terraform-ror`) and click **Open in Terminal**.
9. In the terminal, enter `terraform init` to initialize the directory for Terraform.

   Note: You can run `terraform fmt` in the terminal to have Terraform format all the files in the directory.

Set Up Commonly Used Variables and Values

In this section, you'll set up variables that are used across many resources, for example, region, compartment OCID, and availability domain.

1. Make two more files in `terraform-ror` for declaring and defining variables: `variables.tf` and `terraform.tfvars`

Declare and Define a Variable for Compartment OCID

2. In `variables.tf`, declare a variable for your compartment OCID. Add a flag for sensitive information so that this is not printed in any outputs.

```
# Common variable for compartment
variable "compartment_id" {
  type      = string
  sensitive = true
}
```

3. In `terraform.tfvars`, define your compartment OCID. To find your compartment OCID, minimize Code Editor and navigate to **Identity & Security** > **Compartments** in the main menu. Find your compartment in the table and copy the OCID.

```
# Common variable for compartment
compartment_id = "ocid1.compartment.oc1..exampleid"
```

Declare and Define a Variable for Region

4. Back in `variables.tf`, add a variable for region.

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

5. In `terraform.tfvars`, add the value for your region. Use the OCI documentation for regions and availability domains to find the appropriate region identifier for your region. This example will use US West (Phoenix) and, therefore,"us-phoenix-1" but be sure to replace this based on your account.

```
region = "us-phoenix-1"
```

Declare and Define a Variable for Your Availability Domains

For the availability domain, different tenancies have different "identifiers" for the ADs. This allows OCI to distribute the load more evenly. You'll need to find and set your AD identifiers for your chosen region.

6. Declare a variable for the AD names in `variables.tf`.

```
variable "ads" {
  type = list(string)
}
```

7. To see your identifiers, use the OCI CLI in the terminal in Code Editor:

```
$ oci iam availability-domain list
```

8. This will return a JSON string with a "name" field. Find this field to set this variable in `terraform.tfvars`. Note that your values will be different than these.

```
ads = ["aBCD:PHX-AD-1", "aBCD:PHX-AD-2", "aBCD:PHX-AD-3"]
```

Note: In this lab, both region and availability domain have been configured as independent variables. In more robust code, you could use data sources to make the list of ADs region dependent. However, care should be taken to avoid a common anti-pattern regarding region-agnostic code that can lead to accidentally using the wrong AD.

Verify Your Variable Declarations and Definitions

9. Check that your `variables.tf` looks like this:

```
# Common variables
variable "compartment_id" {
  type      = string
  sensitive = true
}
variable "region" {
  type = string
}
variable "ads" {
  type = list(string)
}
```

10. Check that your `terraform.tfvars` looks like this:

```
# Common variables
compartment_id = "ocid1.compartment.oc1..exampleid"
region         = "us-phoenix-1"
ads            = ["aBCD:PHX-AD-1", "aBCD:PHX-AD-2", "aBCD:PHX-AD-3"]
```

Deploy the Network with Terraform

Create a VCN with Terraform

First, create a VCN to house all of the other resources.

1. Open the documentation for a VCN in Terraform.

2. Look at the arguments that this resource block can take.

3. The only mandatory variable is compartment OCID, which you already have a variable for. The three optional arguments that you'll set in this lab are for the CIDR blocks, the DNS label, and the display name. Go to the documentation for each of these arguments and find its type.

4. Create these variables in `variables.tf` as below. Note that this example uses a slightly more complicated syntax to group the variables together into one object. This practice will help keep the variables more organized throughout the lab.

```
# VCN variables
variable "vcn_config" {
  type = object({
    cidr_blocks  = list(string)
    display_name = string
    dns_label    = string
  })
}
```

5. Define these values in `terraform.tfvars`. Notice the difference in syntax between the type *string* and type *list of strings*.

```
# VCN variables
vcn_config = {
  cidr_blocks  = ["10.0.0.0/16"]
  display_name = "PHX-OP-LAB15-1-VCN-ROR"
  dns_label    = "vcnror"
}
```

6. With those variables set, you can now declare your VCN. Make a file in `terraform-ror` for declaring network resources named `network.tf`.

7. In `network.tf`, add a resource block for the VCN.

```
# Virtual Cloud Network resource block
resource "oci_core_vcn" "ror_vcn" {
  compartment_id = var.compartment_id

  display_name = var.vcn_config.display_name
  cidr_blocks  = var.vcn_config.cidr_blocks
  dns_label    = var.vcn_config.dns_label
}
```

8. Test the VCN declaration by running `terraform plan` in the terminal. It should plan to create a VCN. Note that—just like in the Console—this would also create a default route table (with no route rules) and a default security list (with rules for ICMP and SSH).

Add Gateways to the VCN with Terraform

Next, add gateways to allow traffic in and out of your VCN.

9. Open the documentation for an internet gateway and look at the arguments. The optional arguments that you'll set in this lab are the display name and whether the IG is enabled. Add these variables to `variables.tf`.

```
# Internet gateway variables
variable "ig_config" {
  type = object({
    display_name = string
    enabled      = bool
  })
}
```

10. Define them in `terraform.tfvars`. Notice that `ig_enabled` does not have quotation marks around its definition because it is a *bool* rather than a *string*.

```
# Internet gateway variables
ig_config = {
  display_name = "PHX-OP-LAB15-1-IG"
  enabled      = true
}
```

11. Add the resource block for an Internet gateway in `network.tf`.

```
# Internet gateway resource block
resource "oci_core_internet_gateway" "ror_ig" {
  compartment_id = var.compartment_id
  vcn_id         = oci_core_vcn.ror_vcn.id

  display_name = var.ig_config.display_name
  enabled      = var.ig_config.enabled
}
```

12. Open the documentation for a NAT gateway and look at the arguments. The only optional argument that you'll want to set is for the display name. Add the variable to `variables.tf`.

```
# NAT gateway variables
variable "ng_display_name" {
  type = string
}
```

13. Define the display name in `terraform.tfvars`.

```
# NAT gateway variables
ng_display_name = "PHX-OP-LAB15-1-NG"
```

14. Add the resource block for the NAT gateway to `network.tf`.

```
# NAT gateway resource block
resource "oci_core_nat_gateway" "ror_ng" {
  compartment_id = var.compartment_id
  vcn_id         = oci_core_vcn.ror_vcn.id

  display_name = var.ng_display_name
}
```

15. Test the configuration so far by running `terraform plan`.

Add Route Tables to Utilize the Gateways

For subnets to use gateways, they need route rules. The default route table is empty, so you'll create two new ones: one for the public subnets and one for the private subnets.

16. Open the documentation for route tables in Terraform. The optional arguments that you'll want to set are the display name and route rules. However, because route rules require referencing the OCID of an element that will be provisioned, you won't be able to declare and define a variable for it as normal. More advanced Terraform can deal with this, but for now, you can skip using a variable for route rules. Only add variables for the display names in `variables.tf`.

```
# Route table variables
variable "rt_public_display_name" {
  type = string
}
variable "rt_private_display_name" {
  type = string
}
```

17. Define the display names in `terraform.tfvars`.

```
# Route table variables
rt_public_display_name = "PHX-OP-LAB15-1-RT-PUBLIC"
rt_private_display_name = "PHX-OP-LAB15-1-RT-PRIVATE"
```

18. Add the block for the public route table to `network.tf`.

```
# Public route table resource block
resource "oci_core_route_table" "ror_rt_public" {
  compartment_id = var.compartment_id
  vcn_id         = oci_core_vcn.ror_vcn.id

  display_name = var.rt_public_display_name
  route_rules {
    network_entity_id = oci_core_internet_gateway.ror_ig.id
    destination       = "0.0.0.0/0"
    destination_type  = "CIDR_BLOCK"
  }
}
```

Note: Above, you can see the line where the target OCID (network_entity_id) is declared. This is the line that prevented you from declaring the route rule as a variable. An example of using more advanced Terraform to deal with this and still declare a variable can be found in the official Oracle-provided Terraform modules. For the curious, there are two tricks that have been used in this example: first, the key used is dynamic blocks to unwrap and repackage the variable with the needed OCID, and second, a filter to *ad hoc* create a keyword to filter for the appropriate entity OCID has been used.

19. Add the block for the private route table to `network.tf`. Notice that the only differences are the Terraform label, and the display name, and that the private route table targets the NAT gateway instead of the Internet gateway.

```
# Private route table resource block
resource "oci_core_route_table" "ror_rt_private" {
  compartment_id = var.compartment_id
  vcn_id         = oci_core_vcn.ror_vcn.id

  display_name = var.rt_private_display_name
  route_rules {
    network_entity_id = oci_core_nat_gateway.ror_ng.id
    destination       = "0.0.0.0/0"
    destination_type  = "CIDR_BLOCK"
  }
}
```

20. Test the configuration with `terraform plan` in the terminal.

Add Subnets to a VCN with Terraform

21. Open the documentation for a subnet in Terraform. Look at the arguments that this resource block can take. The ones you'll need to set are for the display name, CIDR block, DNS Label, and route table. Create an object variable for each of the subnets in `variables.tf`:

```
# Load balancer subnet config
variable "snt_lb_config" {
  type = object({
    display_name = string
    dns_label    = string
    cidr_block   = string
```

```
  })
}

# App server subnet config
variable "snt_app_config" {
  type = object({
    display_name = string
    dns_label    = string
    cidr_block   = string
  })
}

# Database subnet config
variable "snt_db_config" {
  type = object({
    display_name = string
    dns_label    = string
    cidr_block   = string
  })
}
```

22. Define all of their values in `terraform.tfvars`.

```
# Load balancer subnet config
snt_lb_config = {
  display_name = "PHX-OP-LAB15-1-SNT-LB"
  dns_label    = "sntlb"
  cidr_block   = "10.0.1.0/24"
}

# App server subnet config
snt_app_config = {
  display_name = "PHX-OP-LAB15-1-SNT-APP"
  dns_label    = "sntapp"
  cidr_block   = "10.0.2.0/24"
}

# Database subnet config
snt_db_config = {
  display_name = "PHX-OP-LAB15-1-SNT-DB"
  dns_label    = "sntdb"
  cidr_block   = "10.0.3.0/24"
}
```

23. Add the corresponding resource block for the public subnet in `network.tf`. Note that the argument for prohibit public IP addresses is set to *false* and the route table is pointed to the public route table.

```
# Load balancer subnet
resource "oci_core_subnet" "ror_snt_lb" {
  compartment_id = var.compartment_id
  vcn_id         = oci_core_vcn.ror_vcn.id
  cidr_block     = var.snt_lb_config.cidr_block

  display_name               = var.snt_lb_config.display_name
  dns_label                  = var.snt_lb_config.dns_label
  prohibit_public_ip_on_vnic = false

  route_table_id = oci_core_route_table.ror_rt_public.id
}
```

24. Add the corresponding resource blocks for the private subnets in `network.tf`. Note that the argument for prohibit public IP addresses is set to *true* and the route table is pointed to the private route table.

```
# App subnet
resource "oci_core_subnet" "ror_snt_app" {
  compartment_id = var.compartment_id
  vcn_id         = oci_core_vcn.ror_vcn.id
  cidr_block     = var.snt_app_config.cidr_block

  display_name               = var.snt_app_config.display_name
  dns_label                  = var.snt_app_config.dns_label
  prohibit_public_ip_on_vnic = true

  route_table_id = oci_core_route_table.ror_rt_private.id
}

# Database subnet
resource "oci_core_subnet" "ror_snt_db" {
  compartment_id = var.compartment_id
  cidr_block     = var.snt_db_config.cidr_block
  vcn_id         = oci_core_vcn.ror_vcn.id

  display_name               = var.snt_db_config.display_name
  dns_label                  = var.snt_db_config.dns_label
  prohibit_public_ip_on_vnic = true
```

```
  route_table_id = oci_core_route_table.ror_rt_private.id
}
```

25. Run `terraform plan` to test the configuration so far.

Create the Network

26. Run `terraform apply` in the terminal to create the network. Enter `yes` to confirm.

27. Minimize Code Editor. In the main menu, navigate to **Networking**, then **Virtual Cloud Networks**. Click your VCN (`PHX-OP-LAB15-1-VCN-ROR`) to go to its details page.

28. Look at the subnet table and confirm that you have all three subnets (load balancer, app, and database).

29. Verify the route tables and gateways:

    a. Click **Route Tables** on the left navigation pane under **Resources**. Confirm that there are three route tables (default, public, and private).

    b. Click the public route table and confirm that it has one rule to the Internet gateway for all IP addresses. Use the breadcrumbs on the top-left corner to return to the VCN details.

    c. Click the private route table and confirm that it has one rule to the NAT gateway for all IP addresses. Use the breadcrumbs on the top-left corner to return to the VCN details.

30. Verify the subnets have the correct route table attached:

    a. Click **Subnets** on the left navigation pane under **Resources**.

    b. Look in the table to confirm that the load balancer subnet is **Public (Regional)**.

    c. Click the load balancer subnet and use the **Subnet Information** tab in the center to confirm that it has the public route table attached.

    d. Use the breadcrumbs on the top-left corner to return to the VCN details.

    e. Look in the subnet table to confirm that the app and database subnets are **Private (Regional)**.

    f. Click *each* of these subnets and use the **Subnet Information** tab in the center to confirm that it has the private route table attached.

31. If you find any differences, compare your Terraform to the code available in this GitHub repository. After fixing any errors, re-run `terraform apply` in the Code Editor terminal to fix your stack. Do not edit via the Console, because this will cause Terraform's state to become out of sync.

Configure Network Security Rules

In the previous section, you configured three subnets: one with Internet access and two without. All of the subnets only have the default security list attached, enabling ingress SSH and ICMP codes 3 and 4. The default security list also enables all egress. In order for Ruby on Rails to work, you'll need to open several ports.

- The load balancer needs to listen on TCP ports 80 and 443 for HTTP(S).
- The app servers need to be accessible on TCP port 8080 for Rails (this is configurable).
- The database servers need to be accessible on TCP ports 3306 and 33060 for MySQL.

For the load balancer and app server, you'll use network security groups to open these ports. You'll open the appropriate ports through security lists instead of the database server.

> **Note:** NSGs are generally more flexible than security lists, so are recommended when possible. This lab is only using security lists for demonstration purposes.

1. Create a new file named `network_security.tf` in `terraform_ror` for the NSGs.

2. Open the documentation for NSGs in Terraform. The only argument that requires a new variable is for the display name. In `variables.tf`, add variables for each of the NSGs (bastion, load balancer, app, and database).

```
# Network security variables
variable "nsg_lb_display_name" {
  type = string
}
variable "nsg_app_display_name" {
  type = string
}
variable "sl_db_display_name" {
  type = string
}
```

3. Define each of these variables in `terraform.tfvars`.

```
# Network security variables
nsg_lb_display_name      = "PHX-OP-LAB15-1-NSG-LB"
nsg_app_display_name     = "PHX-OP-LAB15-1-NSG-APP"
sl_db_display_name       = "PHX-OP-LAB15-1-SL-DB"
```

Create the Load Balancer Network Security Group

4. In `network_security.tf`, add a resource block for the load balancer NSG.

```
# NSG for the load balancer
resource "oci_core_network_security_group" "ror_nsg_lb" {
  compartment_id = var.compartment_id
  vcn_id         = oci_core_vcn.ror_vcn.id
  display_name   = var.nsg_lb_display_name
}
```

5. Open the documentation for an NSG rule in Terraform. Add a rule to open TCP port 80. To specify a rule for TCP, set `protocol` to `"6"`.

```
# Allow HTTP (TCP port 80)
resource "oci_core_network_security_group_security_rule"
"ror_nsg_lb_http" {
  network_security_group_id =
oci_core_network_security_group.ror_nsg_lb.id
  direction                 = "INGRESS"
  protocol                  = "6"
  source                    = "0.0.0.0/0"
  source_type               = "CIDR_BLOCK"
  tcp_options {
    destination_port_range {
      max = 80
      min = 80
    }
  }
}
```

6. Following this pattern, open TCP port 443 for the load balancer NSG.

```
# Allow HTTPS (TCP port 443)
resource "oci_core_network_security_group_security_rule"
"ror_nsg_lb_https" {
  network_security_group_id =
oci_core_network_security_group.ror_nsg_lb.id
  direction                 = "INGRESS"
  protocol                  = "6"
  source                    = "0.0.0.0/0"
  source_type               = "CIDR_BLOCK"
  tcp_options {
    destination_port_range {
```

```
      max = 443
      min = 443
    }
  }
}
```

7. Check that `network_security.tf` looks like this so far:

```
# NSG for the load balancer
resource "oci_core_network_security_group" "ror_nsg_lb" {
  compartment_id = var.compartment_id
  vcn_id         = oci_core_vcn.ror_vcn.id
  display_name   = var.nsg_lb_display_name
}
# Allow HTTP (TCP port 80)
resource "oci_core_network_security_group_security_rule"
"ror_nsg_lb_http" {
  network_security_group_id =
oci_core_network_security_group.ror_nsg_lb.id
  direction                 = "INGRESS"
  protocol                  = "6"
  source                    = "0.0.0.0/0"
  source_type               = "CIDR_BLOCK"
  tcp_options {
    destination_port_range {
      max = 80
      min = 80
    }
  }
}
# Allow HTTPS (TCP port 443)
resource "oci_core_network_security_group_security_rule"
"ror_nsg_lb_https" {
  network_security_group_id =
oci_core_network_security_group.ror_nsg_lb.id
  direction                 = "INGRESS"
  protocol                  = "6"
  source                    = "0.0.0.0/0"
  source_type               = "CIDR_BLOCK"
  tcp_options {
    destination_port_range {
      max = 443
      min = 443
    }
  }
}
```

Create the Rails Network Security Group

8. In `network_security.tf`, create an NSG for Rails and add a rule to allow TCP port 8080:

```
# NSG for the app servers
resource "oci_core_network_security_group" "ror_nsg_app" {
  compartment_id = var.compartment_id
  vcn_id         = oci_core_vcn.ror_vcn.id
  display_name   = var.nsg_app_display_name
}
# Allow Rails (TCP port 8080)
resource "oci_core_network_security_group_security_rule"
"ror_nsg_app_rails" {
  network_security_group_id =
oci_core_network_security_group.ror_nsg_app.id
  direction                 = "INGRESS"
  protocol                  = "6"
  source                    = "0.0.0.0/0"
  source_type               = "CIDR_BLOCK"
  tcp_options {
    destination_port_range {
      max = 8080
      min = 8080
    }
  }
}
```

Create and Attach the MySQL Security List

9. Open the documentation for a security list in Terraform.

10. Add a block with no rules for the database security list (you'll set rules in the next couple steps).

```
# Security list for the database servers
resource "oci_core_security_list" "ror_sl_db" {
  compartment_id = var.compartment_id
  vcn_id         = oci_core_vcn.ror_vcn.id
  display_name   = var.sl_db_display_name

  # Allow TCP port 3306 for MySQL
  # [Placeholder for security rules]

  # Allow TCP port 33060 for MySQL
  # [Placeholder for security rules]
}
```

11. To add an ingress rule, add an `ingress_security_rules` block to the security list. Like for the NSG, use protocol 6 to select TCP:

```
# Security list for the database servers
resource "oci_core_security_list" "ror_sl_db" {
  compartment_id = var.compartment_id
  vcn_id         = oci_core_vcn.ror_vcn.id
  display_name   = var.sl_db_display_name

  # Allow TCP port 3306 for MySQL
  ingress_security_rules {
    protocol    = "6"
    source      = "0.0.0.0/0"
    source_type = "CIDR_BLOCK"
    tcp_options {
      max = 3306
      min = 3306
    }
  }

  # Allow TCP port 33060 for MySQL
  # [Placeholder for security rules]
}
```

12. To add a second rule, simply create another `ingress_security_rules` block for TCP port 33060:

```
# Security list for the database servers
resource "oci_core_security_list" "ror_sl_db" {
  compartment_id = var.compartment_id
  vcn_id         = oci_core_vcn.ror_vcn.id
  display_name   = var.sl_db_display_name

  # Allow TCP port 3306 for MySQL
  ingress_security_rules {
    protocol    = "6"
    source      = "0.0.0.0/0"
    source_type = "CIDR_BLOCK"
    tcp_options {
      max = 3306
      min = 3306
    }
  }

  # Allow TCP port 33060 for MySQL
  ingress_security_rules {
    protocol    = "6"
    source      = "0.0.0.0/0"
    source_type = "CIDR_BLOCK"
    tcp_options {
      max = 33060
      min = 33060
    }
  }
}
```

13. To attach this security list, go back to `network.tf` and find the resource block for the database subnet. It should currently look like this:

```
# Database subnet
resource "oci_core_subnet" "ror_snt_db" {
  compartment_id = var.compartment_id
  cidr_block     = var.snt_db_config.cidr_block
  vcn_id         = oci_core_vcn.ror_vcn.id

  display_name               = var.snt_db_config.display_name
  dns_label                  = var.snt_db_config.dns_label
```

```
  prohibit_public_ip_on_vnic = true

  route_table_id = oci_core_route_table.ror_rt_private.id
}
```

14. Add a list of security lists to the bottom of the resource block to attach *both* the default security list and the database security list.

```
# Database subnet
resource "oci_core_subnet" "ror_snt_db" {
  compartment_id = var.compartment_id
  cidr_block     = var.snt_db_config.cidr_block
  vcn_id         = oci_core_vcn.ror_vcn.id

  display_name               = var.snt_db_config.display_name
  dns_label                  = var.snt_db_config.dns_label
  prohibit_public_ip_on_vnic = true

  route_table_id = oci_core_route_table.ror_rt_private.id
  # Add the security lists here:
  security_list_ids = [
    oci_core_vcn.ror_vcn.default_security_list_id,
    oci_core_security_list.ror_sl_db.id
  ]
}
```

15. Run `terraform plan` to check the configuration.

16. In the terminal, run `terraform apply` to create the NSGs. Enter `yes` to confirm.

Verify Your Network Security Rules

17. Navigate to your VCN's details page. You can use the main menu to click **Networking**, then **Virtual Cloud Networks**. Then you can select your VCN in the table.

18. Click **Network Security Groups** on the left navigation panel under **Resources**.

    a. Click the load balancer NSG. Verify that it has one rule to allow ingress on TCP port 80 and another to allow ingress on TCP port 443. Use the breadcrumbs to return to the VCN's details page.

    b. Click the app server NSG. Verify that it has one rule to allow ingress on TCP port 8080. Use the breadcrumbs to return to the VCN's details page.

19. In the **Resources** list on the left side, click **Security Lists**.

    a. Click the default security list. Verify that it has ingress rules for TCP port 22, ICMP code 3 and 4, and ICMP code 3. Click **Egress Rules** on the left and verify that it has one rule for all egress. Use the breadcrumbs to return to the VCN's details page.

    b. Click the database server security list. Verify that it has one rule to allow ingress on TCP port 3306 and another to allow ingress on TCP port 33060. Use the breadcrumbs to return to the VCN's details page.

20. In the **Resources** list on the left side, click **Subnets**. Click the database subnet and confirm that it has both the default security list and the database security list attached.

21. If you find any differences, compare your Terraform to the code available in this GitHub repository. After fixing any errors, re-run `terraform apply` in the Code Editor terminal to fix your stack. Do not edit via the Console, because this will cause Terraform's state to become out of sync.

Provision the MySQL Database VM

1. Create a file in `terraform-ror` named `db.tf`.

2. Open the documentation for a compute instance in Terraform. First, find that the required parameters are (a) availability domain, (b) compartment OCID, and (c) shape. Ignore the rest for now.

3. You already have variables for availability domain and compartment OCID, so add the following block to `db.tf` to get started:

```
# Database (MySQL) compute instance
resource "oci_core_instance" "instance_db" {
  compartment_id      = var.compartment_id
  availability_domain = var.ads[0]
  # [Placeholder for display name]
  # [Placeholder for shape]
  # [Placeholder for shape config]
  # [Placeholder for source details]
  # [Placeholder for VNIC details]
  # [Placeholder for metadata]
}
```

4. Add a variable in `variables.tf` for a display name:

```
# Display name for the database server
variable "instance_db_name" {
  type = string
}
```

5. Define it in `terraform.tfvars`.

```
instance_db_name  = "PHX-OP-LAB15-1-VM-DB"
```

6. Add a line to define the database's display name in `db.tf`.

```
# Database (MySQL) compute instance
resource "oci_core_instance" "instance_db" {
  compartment_id      = var.compartment_id
  availability_domain = var.ads[0]
  display_name        = var.instance_db_name
  # [Placeholder for shape]
  # [Placeholder for shape config]
  # [Placeholder for source details]
  # [Placeholder for VNIC details]
  # [Placeholder for metadata]
}
```

Configure Compute Instance Shape

7. A quick glance at the documentation for a compute instance would show that the only required field for configuring shape is `shape`. However certain shapes require you to provide additional information in `shape_config`. For flexible shapes, this additional information will require both OCPU count and memory size. Create a variable in `variables.tf` to package these fields together.

```
# Compute instance shape configuration
variable "instance_shape" {
  type = object({
    name          = string
    ocpus         = string
    memory_in_gbs = string
  })
}
```

8. Set these variables in `terraform.tfvars`. The list of shapes is available in the documentation. This lab will use `VM.Standard.A1.Flex` with one OCPU and 6 GB of memory.

```
# Compute shape configuration
instance_shape = {
  name          = "VM.Standard.A1.Flex"
  ocpus         = "1"
  memory_in_gbs = "6"
}
```

9. Edit the compute instance's resource block to configure the shape:

```
# Database (MySQL) compute instance
resource "oci_core_instance" "instance_db" {
  compartment_id      = var.compartment_id
  availability_domain = var.ads[0]
  display_name        = var.instance_db_name
  shape               = var.instance_shape.name
  shape_config {
    ocpus         = var.instance_shape.ocpus
    memory_in_gbs = var.instance_shape.memory_in_gbs
  }
  # [Placeholder for source details]
  # [Placeholder for VNIC details]
  # [Placeholder for metadata]
}
```

10. Run `terraform plan` in the terminal in Code Editor to test the configuration so far. It should plan to create one resource, the compute instance. Do *not* run `terraform apply` yet (it will fail).

Configure Compute Instance Image

To launch a compute instance, OCI requires that you choose an image. Looking at the documentation, however, you'll notice that this is not listed as a required parameter. This is because there are two options to configure the image: `source_details` and `image`. The image parameter is deprecated but maintained for compatibility.

11. Add a variable in `variables.tf` for the image OCID.

```
# Compute instance image OCID
variable "instance_db_image_id" {
  type = string
}
```

12. Use the documentation to find the OCID for the latest image of Oracle Linux 8.6 for ARM (aarch64) in your region. Set the variable for the image OCID in `terraform.tfvars`.

```
# Compute image OCID
instance_db_image_id = "ocid1.image.oc1.region.exampleid"
```

13. Edit the compute instance's resource block to configure the `source_details` block:

```
# Database (MySQL) compute instance
resource "oci_core_instance" "instance_db" {
  compartment_id      = var.compartment_id
  availability_domain = var.ads[0]
  display_name        = var.instance_db_name
  shape               = var.instance_shape.name
  shape_config {
    ocpus         = var.instance_shape.ocpus
    memory_in_gbs = var.instance_shape.memory_in_gbs
  }
  source_details {
    source_type = "image"
    source_id   = var.instance_db_image_id
  }
  # [Placeholder for VNIC details]
  # [Placeholder for metadata]
}
```

14. Run `terraform plan` to test the configuration. Do not run `terraform apply` (it will still fail).

Configure the Compute Instance VNIC

The next thing required to configure for this compute instance is the VNIC. Similar to configuring the image, the VNIC configuration is listed as optional because there are two options: `create_vnic_details` and `subnet_id` (outside of `create_vnic_details`).

15. Edit the compute instance's resource block to configure the `create_vnic_details` block. Set the subnet OCID and choose not to assign a public IP.

```
# Database (MySQL) compute instance
resource "oci_core_instance" "instance_db" {
  compartment_id      = var.compartment_id
  availability_domain = var.ads[0]
  display_name        = var.instance_db_name
  shape               = var.instance_shape.name
  shape_config {
    ocpus         = var.instance_shape.ocpus
    memory_in_gbs = var.instance_shape.memory_in_gbs
  }
  source_details {
    source_type = "image"
    source_id   = var.instance_db_image_id
  }
  create_vnic_details {
    subnet_id        = oci_core_subnet.ror_snt_db.id
    assign_public_ip = false
  }
  # [Placeholder for metadata]
}
```

Add an SSH Key to the Compute Instance

To add an SSH key to the authorized keys of the compute instance, you only have to point Terraform to the location of the public key.

16. Assuming that you are in the directory `~/terraform-ror` in your Code Editor terminal, that you have generated an SSH key in Cloud Shell's default location, and that you want to use that key, you can edit the instance's block as follows to add the SSH key.

```
# Database (MySQL) compute instance
resource "oci_core_instance" "instance_db" {
```

```
  compartment_id      = var.compartment_id
  availability_domain = var.ads[0]
  display_name        = var.instance_db_name
  shape               = var.instance_shape.name
  shape_config {
    ocpus         = var.instance_shape.ocpus
    memory_in_gbs = var.instance_shape.memory_in_gbs
  }
  source_details {
    source_type = "image"
    source_id   = var.instance_db_image_id
  }
  create_vnic_details {
    subnet_id        = oci_core_subnet.ror_snt_db.id
    assign_public_ip = false
  }
  metadata = {
    ssh_authorized_keys = file("~/.ssh/id_rsa.pub")
    # [Placeholder for Cloud-init metadata]
  }
}
```

Add the `Cloud-init` Data to the Database Compute Instance

Next, you'll create a template `Cloud-init` script to configure MySQL on the compute instance.

17. Create a file named `configure_mysql.tftpl` in `terraform-ror`.

18. Start the file with a comment declaring that it is a cloud-config file:

```
#cloud-config
```

19. First, you'll use the `write_files` module. This module simply writes the content provided to a file at the path specified. Add an element to this module that will write a SQL script for MySQL:

```
#cloud-config

write_files:
  - path: /home/opc/configure_mysql.sql
    content: |
      ALTER USER 'root'@'localhost' IDENTIFIED BY '${ root_pass }';
      FLUSH PRIVILEGES;
      CREATE USER '${ rails_user }'@'%'
```

```
    IDENTIFIED BY '${ rails_pass }';
  GRANT ALL
    ON *.*
    TO '${ rails_user }'@'%'
    WITH GRANT OPTION;
```

20. Next, you'll use the `runcmd` module. This provides a series of commands to the compute instance to run on the first boot. Append this block to the end of the cloud-config file.

```
runcmd:
  # Install MySQL Community
  - sudo rpm -ivh https://repo.mysql.com/mysql80-community-release-
el8-4.noarch.rpm
  - sudo dnf module disable -y mysql
  - sudo dnf install -y mysql-community-server
  # Start MySQL Server
  - sudo systemctl start mysqld
  - sudo systemctl enable mysqld
  # Get the temporary root password
  - "pass=$(sudo awk -F': ' '/temporary password/ {print $NF}'
/var/log/mysqld.log)"
  # Execute the SQL script to change the root password and create a
new user for Rails
  - mysql -sfu root -p$${pass} --connect-expired-password <
"/home/opc/configure_mysql.sql"
  # Open the firewall for MySQL
  - sudo firewall-offline-cmd --add-service=mysql
  - sudo systemctl restart firewalld
```

21. Next, you can use Terraform's `cloud-init` data source to substitute values into this template and base64 encode it. Back in `db.tf`, add the following block at the bottom:

```
data "cloudinit_config" "mysql_config" {
  gzip          = true
  base64_encode = true
  part {
    content_type = "text/cloud-config"
    content = templatefile("configure_mysql.tftpl", {
      root_pass  = var.mysql_root_pass,
      rails_user = var.mysql_rails_user,
      rails_pass = var.mysql_rails_pass
    })
    filename = "configure_mysql.yaml"
  }
}
```

22. Declare the password for MySQL configuration in `terraform.tfvars`:

```
# MySQL configuration
mysql_root_pass  = "RootPassword123%" # Example password
mysql_rails_user = "rubyonrails" # Example username
mysql_rails_pass = "RailsPassword123%" # Example password
```

23. Because we have declared passwords for MySQL root user along with rails user, add the following block in `variables.tf`:

```
# MySQL configuration
variable "mysql_root_pass" {
  type      = string
  sensitive = true
}
variable "mysql_rails_user" {
  type      = string
  sensitive = true
}
variable "mysql_rails_pass" {
  type      = string
  sensitive = true
}
```

24. Finally, you can pass the rendered `Cloud-init` file as metadata to the compute instance. Add the `user_data` line to your database server resource block, as shown below:

```
# Database (MySQL) compute instance
resource "oci_core_instance" "instance_db" {
  compartment_id      = var.compartment_id
  availability_domain = var.ads[0]
  display_name        = var.instance_db_name
  shape               = var.instance_shape.name
  shape_config {
    ocpus         = var.instance_shape.ocpus
    memory_in_gbs = var.instance_shape.memory_in_gbs
  }
  source_details {
    source_type = "image"
    source_id   = var.instance_db_image_id
  }
  create_vnic_details {
    subnet_id        = oci_core_subnet.ror_snt_db.id
    assign_public_ip = false
  }
  metadata = {
```

```
    ssh_authorized_keys = file("~/.ssh/id_rsa.pub")
    user_data           = data.cloudinit_config.mysql_config.rendered
  }
}
```

25. Your final code for `db.tf` should look like this:

```
# Database (MySQL) compute instance
resource "oci_core_instance" "instance_db" {
  compartment_id      = var.compartment_id
  availability_domain = var.ads[0]
  display_name        = var.instance_db_name
  shape               = var.instance_shape.name
  shape_config {
    ocpus         = var.instance_shape.ocpus
    memory_in_gbs = var.instance_shape.memory_in_gbs
  }
  source_details {
    source_type = "image"
    source_id   = var.instance_db_image_id
  }
  create_vnic_details {
    subnet_id        = oci_core_subnet.ror_snt_db.id
    assign_public_ip = false
  }
  metadata = {
    ssh_authorized_keys = file("~/.ssh/id_rsa.pub")
    user_data           = data.cloudinit_config.mysql_config.rendered
  }
}

data "cloudinit_config" "mysql_config" {
  gzip          = true
  base64_encode = true
  part {
    content_type = "text/cloud-config"
    content = templatefile("configure_mysql.tftpl", {
      root_pass  = var.mysql_root_pass,
      rails_user = var.mysql_rails_user,
      rails_pass = var.mysql_rails_pass
    })
    filename = "configure_mysql.yaml"
  }
}
```

26. Run `terraform plan` to check the configuration.

27. In the terminal, run `terraform apply` to create the MySQL Database. Enter `yes` to confirm.

Test the MySQL Server

28. Open Cloud Shell using the Developer Tools icon in the top-right corner. It may display Code Editor and Cloud Shell as split screen. You can use the **View** drop-down menu on the top-left corner to change these to tabs if you prefer.

29. Recall that the database compute instance is in a private subnet. To access it, use the **Network** drop-down menu to select **Private Network Definition List**. Click **Create Private Network Definition**.
    a. Provide the name `PHX-OP-LAB15-1-PND-ROR`
    b. In the setup panel, select your VCN (`PHX-OP-LAB15-1-VCN-ROR`).
    c. Select the subnet for the app server (`PHX-OP-LAB15-1-SNT-DB`).
    d. Skip the Network Security Group selection. Click Create.
    e. Once the Private network definition is created, click on the 3 vertical dots on the right.
    f. Click **Use as active network**. On the confirmation box, click Use as active network.
    g. Wait for the **Network** drop-down menu to display that you are connected.

30. Connect to the server with the following command:
    ```
    ssh opc@<private_ip>
    ```

31. Check the status of the server with the following command. Note that `Cloud-init` will not actually finish running until several minutes after the instance has finished provisioning. If MySQL appears down, wait for several more minutes.
    ```
    sudo systemctl status mysqld
    ```

32. Check that you can log in to MySQL with the following command. It will ask you to provide the password.
    ```
    mysql -u rubyonrails -p
    ```

33. While connected to MySQL, also check that you can create a test database.
    ```
    CREATE DATABASE testdb1;
    ```

34. Type **`exit`** to step out of the SQL command window.

Provision the Rails Compute Instance

1. Create a file in `terraform-ror` named `app.tf`.

2. You already have variables for availability domain and compartment OCID, so add the following block to `app.tf` to get started:

```
# App (Ruby on Rails) compute instance
resource "oci_core_instance" "instance_app" {
  compartment_id      = var.compartment_id
  availability_domain = var.ads[0]
  # [Placeholder for display name]
  # [Placeholder for shape]
  # [Placeholder for shape config]
  # [Placeholder for source details]
  # [Placeholder for VNIC details]
  # [Placeholder for metadata]
}
```

3. Add a variable in `variables.tf` for a display name:

```
# Display name for the app server
variable "instance_app_name" {
  type = string
}
```

4. Define it in `terraform.tfvars`.

```
instance_app_name = "PHX-OP-LAB15-1-VM-APP"
```

5. Add a line to define the app VM's display name in `app.tf`.

```
# App (Ruby on Rails) compute instance
resource "oci_core_instance" "instance_app" {
  compartment_id      = var.compartment_id
  availability_domain = var.ads[0]
  display_name        = var.instance_app_name
  # [Placeholder for shape]
  # [Placeholder for shape config]
  # [Placeholder for source details]
  # [Placeholder for VNIC details]
  # [Placeholder for metadata]
}
```

Configure Compute Instance Shape

6. Edit the compute instance's resource block to configure the shape:

```
# App (Ruby on Rails) compute instance
resource "oci_core_instance" "instance_app" {
  compartment_id      = var.compartment_id
  availability_domain = var.ads[0]
  display_name        = var.instance_app_name
  shape               = var.instance_shape.name
  shape_config {
    ocpus         = var.instance_shape.ocpus
    memory_in_gbs = var.instance_shape.memory_in_gbs
  }
  # [Placeholder for source details]
  # [Placeholder for VNIC details]
  # [Placeholder for metadata]
}
```

7. Run `terraform plan` in the terminal in Code Editor to test the configuration so far. It should plan to create one resource, the compute instance. Do *not* run `terraform apply` yet (it will fail).

Configure the Compute Instance Image

8. Add a variable in `variables.tf` for the image OCID.

```
# Compute instance image OCID for Ubuntu
variable "instance_app_image_id" {
  type = string
}
```

9. Use the documentation to find the OCID for the latest image of Ubuntu 22.04 for ARM (aarch64) in your region. Set the variable for the image OCID in `terraform.tfvars`.

```
# Compute image OCID
instance_app_image_id = "ocid1.image.oc1.region.exampleid"
```

10. Edit the compute instance's resource block to configure the `source_details` block:

```
# App (Ruby on Rails) compute instance
resource "oci_core_instance" "instance_app" {
  compartment_id      = var.compartment_id
  availability_domain = var.ads[0]
  display_name        = var.instance_app_name
  shape               = var.instance_shape.name
  shape_config {
    ocpus         = var.instance_shape.ocpus
    memory_in_gbs = var.instance_shape.memory_in_gbs
  }
```

```
  source_details {
    source_type = "image"
    source_id   = var.instance_app_image_id
  }
  # [Placeholder for VNIC details]
  # [Placeholder for metadata]
}
```

Configure Compute Instance VNIC

11. Edit the compute instance's resource block to configure the `create_vnic_details` block. Set the subnet OCID and choose not to assign a public IP. This is also where you attach the NSG to the VNIC.

```
# App (Ruby on Rails) compute instance
resource "oci_core_instance" "instance_app" {
  compartment_id      = var.compartment_id
  availability_domain = var.ads[0]
  display_name        = var.instance_app_name
  shape               = var.instance_shape.name
  shape_config {
    ocpus         = var.instance_shape.ocpus
    memory_in_gbs = var.instance_shape.memory_in_gbs
  }
  source_details {
    source_type = "image"
    source_id   = var.instance_app_image_id
  }
  create_vnic_details {
    subnet_id        = oci_core_subnet.ror_snt_app.id
    assign_public_ip = false
    nsg_ids = [
      oci_core_network_security_group.ror_nsg_app.id
    ]
  }
  # [Placeholder for metadata]
}
```

12. Run `terraform plan` to test the configuration.

Add an SSH Key to the Compute Instance

13. Add an SSH key to the metadata field of the app compute instance.

```
# App (Ruby on Rails) compute instance
resource "oci_core_instance" "instance_app" {
  compartment_id      = var.compartment_id
  availability_domain = var.ads[0]
  display_name        = var.instance_app_name
  shape               = var.instance_shape.name
  shape_config {
    ocpus         = var.instance_shape.ocpus
    memory_in_gbs = var.instance_shape.memory_in_gbs
  }
  source_details {
    source_type = "image"
    source_id   = var.instance_app_image_id
  }
  create_vnic_details {
    subnet_id        = oci_core_subnet.ror_snt_app.id
    assign_public_ip = false
    nsg_ids = [
      oci_core_network_security_group.ror_nsg_app.id
    ]
  }
  metadata = {
    ssh_authorized_keys = file("~/.ssh/id_rsa.pub")
    # [Placeholder for Cloud-init metadata]
  }
}
```

Note: If your public key is in a different location, replace `~/.ssh/id_rsa.pub` with that location. If you use a nondefault location, be sure to specify the correct corresponding private key in the SSH command.

Add `Cloud-init` Data to the App Compute Instance

Next, you'll create a template `Cloud-init` script to configure Ruby on Rails on the compute instance.

14. Create a file named `configure_rails.tftpl` in `terraform-ror`.

15. Start the file with a comment declaring that it is a cloud-config file:

```
#cloud-config
```

16. Use the `write_files` module to write a system service configuration for Rails:

```
#cloud-config

write_files:
  - path: /lib/systemd/system/railsd.service
    content: |
      [Unit]
      Description=start rails
      [Service]
      ExecStart=/opt/apps/myapp/start_rails.sh start
      [Install]
      WantedBy=multi-user.target
    owner: root:root
    permissions: '0o644'
    encoding: text/plain

  # [Placeholder for start_rails.sh block]

  # [Placeholder for database.yml block]
```

17. Add another element to this module for a short script that starts Rails.

```
#cloud-config

write_files:
  - path: /lib/systemd/system/railsd.service
    content: |
      [Unit]
      Description=start rails
      [Service]
      ExecStart=/opt/apps/myapp/start_rails.sh start
      [Install]
      WantedBy=multi-user.target
    owner: root:root
    permissions: '0o644'
    encoding: text/plain

  - path: /home/ubuntu/start_rails.sh
    content: |
      #!/bin/bash
      cd /opt/apps/myapp
      sudo rails s  -p 8080 -b 0.0.0.0
    owner: root:root
```

```
    permissions: '0o744'
    encoding: text/plain

  # [Placeholder for database.yml block]
```

18. Add a last element to this module for the database configuration:

```
#cloud-config

write_files:
  - path: /lib/systemd/system/railsd.service
    content: |
      [Unit]
      Description=start rails
      [Service]
      ExecStart=/opt/apps/myapp/start_rails.sh start
      [Install]
      WantedBy=multi-user.target
    owner: root:root
    permissions: '0o644'
    encoding: text/plain

  - path: /home/ubuntu/start_rails.sh
    content: |
      #!/bin/bash
      cd /opt/apps/myapp
      sudo rails s  -p 8080 -b 0.0.0.0
    owner: root:root
    permissions: '0o744'
    encoding: text/plain

  - path: /home/ubuntu/database.yml
    content: |
      default: &default
        adapter: mysql2
        encoding: utf8mb4
        pool: <%= ENV.fetch("RAILS_MAX_THREADS") { 5 } %>
        username: "${ rails_user }"
        password: "${ rails_pass }"
        host: "${ db_ip_addr }"
      development:
        <<: *default
        database: myapp_development
      test:
        <<: *default
        database: myapp_test
      production:
```

```
        <<: *default
        database: myapp_production
        username: "${ rails_user }"
        password: "${ rails_pass }"
    owner: root:root
    permissions: '0o644'
    encoding: text/plain
```

19. Use the `runcmd` module to install and configure Ruby on Rails. Add the following block to the end of the cloud-config file:

```
runcmd:
  # Install ruby manager, mysql-client and other dependencies
  - sudo apt update
  - sudo apt-get install -y build-essential git libsqlite3-dev libssl-
dev libzlcore-dev mysql-client libmysqlclient-dev git-core zlib1g-dev
build-essential libssl-dev libreadline-dev libyaml-dev libsqlite3-dev
sqlite3 libxml2-dev libxslt1-dev libcurl4-openssl-dev software-
properties-common libffi-dev nodejs npm
  # Install yarn
  - curl -sS https://dl.yarnpkg.com/debian/pubkey.gpg | sudo apt-key
add
  - echo "deb https://dl.yarnpkg.com/debian/ stable main" | sudo tee
/etc/apt/sources.list.d/yarn.list
  - sudo apt update
  - sudo apt install -y yarn
  # Install rails and bundler gems
  - sudo apt-get install -y ruby-full
  - sudo gem install rails
  - sudo gem install bundler
  # Create an app
  - sudo mkdir /opt/apps
  - sudo chown ubuntu:ubuntu /opt/apps
  - cd /opt/apps
  - rails new myapp -d mysql
  # Open port 8080 for HTTP
  - sudo iptables -I INPUT 6 -m state --state NEW -p tcp --dport 8080
-j ACCEPT
  # Move database config to the application
  - cd /opt/apps/myapp/
  - mv /home/ubuntu/database.yml /opt/apps/myapp/config/database.yml
  - rake db:create
  - rake db:migrate
  # Add start_rails as a service
  - sudo mv /home/ubuntu/start_rails.sh /opt/apps/myapp/
  - sudo systemctl daemon-reload
```

```
  - sudo systemctl enable railsd.service
  - sudo systemctl start railsd.service
```

20. Next, use Terraform's `cloud-init` data source to substitute values into this template and base64 encode it. Back in `app.tf`, add the following block at the bottom:

```
data "cloudinit_config" "ror_config" {
  gzip          = true
  base64_encode = true
  part {
    content_type = "text/cloud-config"
    content = templatefile("configure_rails.tftpl", {
      rails_user = var.mysql_rails_user,
      rails_pass = var.mysql_rails_pass,
      db_ip_addr = oci_core_instance.instance_db.private_ip
    })
    filename = "configure_rails.yaml"
  }
}
```

21. Finally, pass the rendered `Cloud-init` file as metadata to the compute instance. Add the `user_data` line to your app server resource block, as shown below.

```
# App (Ruby on Rails) compute instance
resource "oci_core_instance" "instance_app" {
  compartment_id      = var.compartment_id
  availability_domain = var.ads[0]
  display_name        = var.instance_app_name
  shape               = var.instance_shape.name
  shape_config {
    ocpus         = var.instance_shape.ocpus
    memory_in_gbs = var.instance_shape.memory_in_gbs
  }
  source_details {
    source_type = "image"
    source_id   = var.instance_app_image_id
  }
  create_vnic_details {
    subnet_id        = oci_core_subnet.ror_snt_app.id
    assign_public_ip = false
    nsg_ids = [
      oci_core_network_security_group.ror_nsg_app.id
    ]
  }
```

```
  metadata = {
    ssh_authorized_keys = file("~/.ssh/id_rsa.pub")
    user_data           = data.cloudinit_config.ror_config.rendered
  }
}
```

Provision the App Compute Instance

22. Run `terraform plan` in the terminal in Code Editor to confirm the configuration.

23. Finally, run `terraform apply` to provision the compute instance. Terraform will summarize the compute instance that it will create. Verify that all the values are correct. Type "yes" to confirm.

Test the Rails Server

24. Open Cloud Shell using the Developer Tools icon in the top-right corner. It may display Code Editor and Cloud Shell as split screen. You can use the **View** drop-down menu on the top-left corner to change these to tabs if you prefer.

25. Recall that the app and database compute instances are in private subnets. To access them, use the **Network** drop-down menu to select **Private Network Definition List.**
    a. Click **Create Private Network Definition**
    b. Provide the name `PHX-OP-LAB15-2-PND-ROR`
    c. Select your VCN (`PHX-OP-LAB15-1-VCN-ROR`).
    d. Select the subnet for the app server (`PHX-OP-LAB15-1-SNT-APP`).
    e. In Network Security Groups, select the app NSG (`IAD-OP-LAB15-1-NSG-APP`)
    f. Click **Create.**
    g. Once the Private network definition is created, click on the 3 vertical dots on the right.
    h. Click **Use as active network**. On the confirmation box, click Use as active network. Click Close.
    i. Wait for the **Network** drop-down menu to display that you are connected.

26. Use curl to retrieve the homepage from the app server. You can find the private IP address on the compute instance table by navigating to **Compute** > **Instances** in the main menu.

```
$ curl http://<private_ip_addr>:8080
```

It should return an HTML file. Note that, like before, `Cloud-init` may take several minutes after the instance finishes provisioning to complete.

Note: In this lab, there was a significant amount of time between the database server and application server provisioning. This allowed `Cloud-init` on the database server to complete before `Cloud-init` on the application server started. However, the Terraform alone did not guarantee that, as Terraform doesn't see into the instances themselves. There are workarounds to allow Terraform to do this, but they are beyond the scope of this lab. It is just important to know that there *is* technically a race condition here.

Add a Public Load Balancer

27. Create a file named `lb.tf` in `terraform-ror`.

28. You'll need to configure a load balancer, a listener, a backend set, and a backend. Open the documentation for each in Terraform.

29. Add the following variables in `variables.tf`:

```
variable "lb_config" {
  type = object({
    display_name              = string
    shape                     = string
    maximum_bandwidth_in_mbps = number
    minimum_bandwidth_in_mbps = number
  })
}

variable "lb_backend_config" {
  type = object({
    display_name = string
    protocol     = string
    port         = number
    policy       = string
    path         = string
  })
}

variable "lb_listener_config" {
  type = object({
    display_name = string
    port         = number
    protocol     = string
  })
}
```

30. Define these values in `terraform.tfvars`:

```
lb_config = {
  display_name              = "PHX-OP-LAB15-1-LB"
  shape                     = "flexible"
  minimum_bandwidth_in_mbps = 10
  maximum_bandwidth_in_mbps = 10
}

lb_backend_config = {
  display_name = "PHX-OP-LAB15-1-LBBS"
```

```
  protocol     = "HTTP"
  port         = 8080
  policy       = "LEAST_CONNECTIONS"
  path         = "/"
}

lb_listener_config = {
  display_name = "PHX-OP-LAB15-1-LBLS"
  protocol     = "HTTP"
  port         = 80
}
```

31. Back in `lb.tf`, add a resource block for the load balancer.

```
# Front-end load balancer
resource "oci_load_balancer_load_balancer" "ror_lb" {
  compartment_id = var.compartment_id
  display_name   = var.lb_config.display_name
  subnet_ids = [
    oci_core_subnet.ror_snt_lb.id
  ]

  shape = var.lb_config.shape
  shape_details {
    maximum_bandwidth_in_mbps =
var.lb_config.maximum_bandwidth_in_mbps
    minimum_bandwidth_in_mbps =
var.lb_config.minimum_bandwidth_in_mbps
  }

  network_security_group_ids = [
    oci_core_network_security_group.ror_nsg_lb.id
  ]
}
```

32. Below the load balancer block, add a block for the backend set and a block for the backend.

```
# Front-end load balancer backend set
resource "oci_load_balancer_backend_set" "ror_lb_backend_set" {
  load_balancer_id = oci_load_balancer_load_balancer.ror_lb.id
  name             = var.lb_backend_config.display_name
  policy           = var.lb_backend_config.policy
  health_checker {
    protocol = var.lb_backend_config.protocol
    port     = var.lb_backend_config.port
    url_path = var.lb_backend_config.path
  }
```

```
}

# Load balancer backend
resource "oci_load_balancer_backend" "ror_lb_backend" {
  load_balancer_id = oci_load_balancer_load_balancer.ror_lb.id
  backendset_name  =
oci_load_balancer_backend_set.ror_lb_backend_set.name
  port             = var.lb_backend_config.port

  ip_address = oci_core_instance.instance_app.private_ip
}
```

33. Finally, add a listener to the load balancer by adding the block below `lb.tf`.

```
# Front-end load balancer listener
resource "oci_load_balancer_listener" "ror_lb_listener" {
  load_balancer_id         = oci_load_balancer_load_balancer.ror_lb.id
  default_backend_set_name =
oci_load_balancer_backend_set.ror_lb_backend_set.name

  name     = var.lb_listener_config.display_name
  port     = var.lb_listener_config.port
  protocol = var.lb_listener_config.protocol
}
```

34. Run `terraform plan` to check the configuration.

35. In the terminal, run `terraform apply` to create the Load balancer. Enter `yes` to confirm.

Test Ruby on Rails Through a Web Browser

36. You should now be able to access the server through a web browser.

37. Enter the public IP address of the load balancer into the URL bar of any web browser.

38. You should see the Rails default home page.