***Chapter 4*: Compute Choices on Oracle Cloud Infrastructure**

One of the core services from **Oracle Cloud Infrastructure** (**OCI**) is the OCI Compute service, which helps you provision and manage compute instances. As it is an on-demand service, you can fulfil your application requirements at any time by launching an instance of your choice. Once you have the instance up and running, you can control it fully. You can access these instances securely and even manage their life cycles, which means you have control over starting, stopping, and deleting these instances.

You can also choose various types of persistence storage that you can attach to these instances, such as block volumes, file storage, object storage, archive storage, and more. Although there are certain high performing NVMe disk-attached instances to help you meet your performance goal, they are lost when you terminate those instances. For those instances, you should add block volumes to back up your data.

In this chapter, we're going to cover the following main topics:

* Introducing the different OCI compute instance types
* Understanding instance operating system images
* Creating similar instances using instance configuration and instance pools
* Connecting to instances using instance console connection

**Introducing the different OCI compute instance types**

OCI is the only public cloud that supports bare metal and VMs using the same set of APIs, hardware, firmware, software stack, and networking infrastructure. You can see the two models in the following diagram:



Figure 4.1 – Compute instance types

**Bare metal** instances are instances where customers get the full server. This is also referred to as a single-tenant model. The advantage here is that there is no performance overhead, no shared agents, and no noisy neighbors. For performance and isolation requirements, this is the best instance type. You can see an example of this in the following diagram:

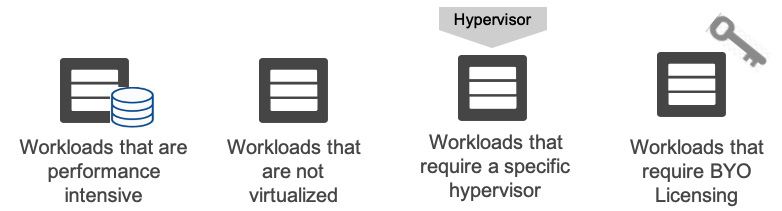


Figure 4.2 – Various workloads for bare metal

On the other spectrum is **virtual machines** (**VMs**), where the underlying host is virtualized to provide smaller VMs. This is also referred to as a multi-tenant model. The advantage here is flexibility in regard to what instance shapes are used. A VM instance is a piece of virtual hardware that shares the physical compute with other VM instances. OS virtualization helps you run multiple virtual machines on top of a physical piece of hardware and share the physical space among them. Logically speaking, these VM instances are isolated.

If you do not have strict or very high performing applications, then this is the best type of instance to use. You can dynamically choose how much CPU and memory to use in these instances and based on your computing power, the network bandwidth available in these instances also varies. Since these instances run on the same type of physical infrastructure, they leverage the same backend infrastructure that created the foundation of Oracle's second-generation cloud.

**Dedicated VM hosts** (**DVH**) is yet another type of OCI compute instance. On DVH, you can run your VMs so that they will be treated as single-tenant VMs. These instances will not be sharing the underlying physical compute with other customers. If you have compliance and regulatory requirements for isolation, then DVH instances are the most suitable. DVH help you be flexible in terms of licensing because you can meet either node-based licensing or host-based licensing using this offering. The following is a logical diagram of a DVH:

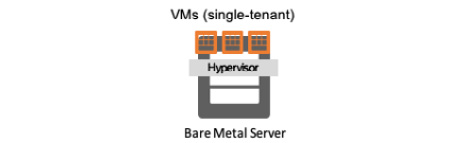


Figure 4.3 – Dedicated VM host

In the next section, we will look at instance shapes.

**Understanding instance shapes**

Based on your application's performance requirements, you have the power to decide on the form of these compute instances. Here, you'll select the amount of CPU cores you wish to use, the amount of memory that you just want for your instances, and so on. Network bandwidth increases linearly with the amount of CPU cores you have per instance. OCI has a good number of compute instance shapes that can help you satisfy your application's performance requirements. Let's observe them.

**Standard shapes**

Standard shapes are for general-purpose workloads. You can meet the requirements of a large range of applications and use cases using these shapes. You can either choose a fixed CPU and memory type for standard shapes or choose a flexible instance type that's powered by an AMD Rome processor.

Standard shapes VMs start from 1 CPU and can go up to 64 CPUs, with the memory starting from 1 GB and going up to 1,024 GB. You can choose either flexible shapes, which are flexible in terms of the amount of processor and memory you can have, or you can go for a fixed amount of CPU and memory. You can also choose instances that are powered by Intel Skylake-based processors. Legacy standard type instances are based on X5 and X7 types of hardware. You can see the AMD Rome-based standard shapes and how you can use the slider to choose the number of resources in the following screenshot.

AMD EPYC processor-powered instances have different use cases. The AMD EPYC bare metal server (64 cores, 512 GB RAM, 2 x 25 Gbps bandwidth, 75 vNICs) is available to customers for $0.03 core/hour (this price is at the time of writing this book). These instances are there to help you cut down on your bill. Also, from a support perspective, it supports **E-Business Suite**, **JD Edwards**, and **PeopleSoft**. You can also run **Cloudera**, **Hortonworks**, **MapR**, and **Transwarp**.

The following screenshot shows various AMD Rome processor-based compute instances on OCI:

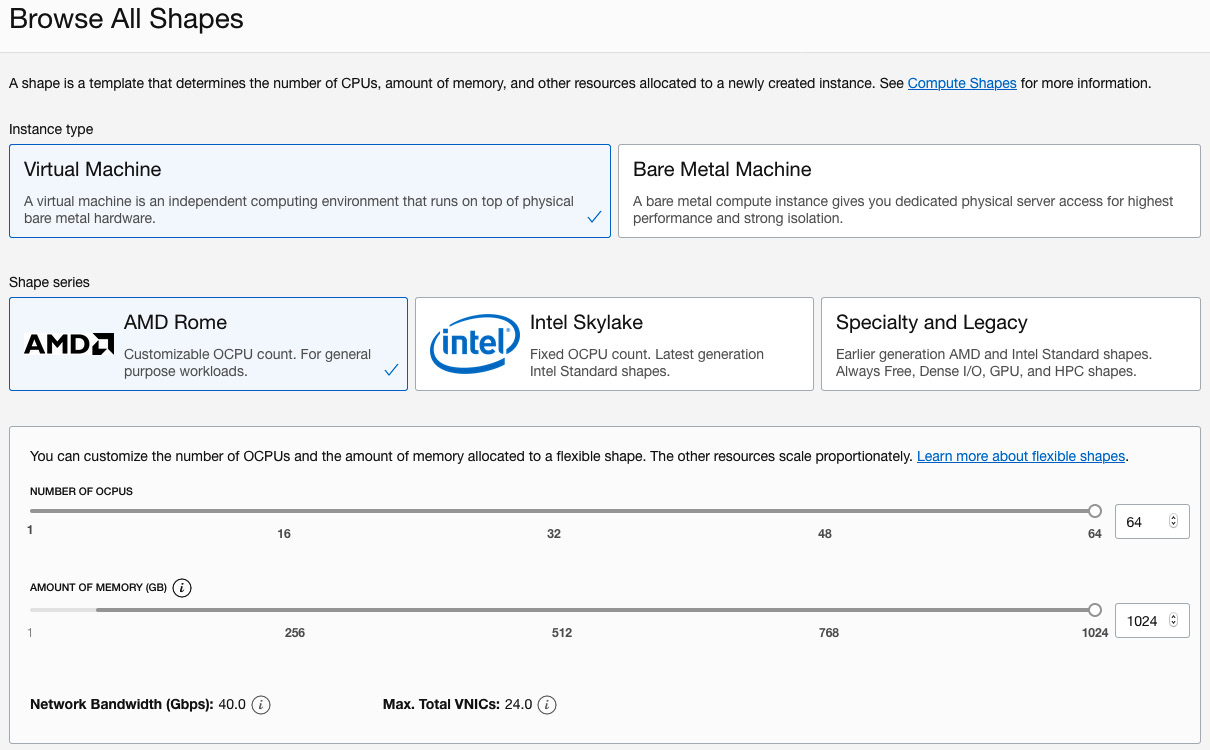


Figure 4.4 – Standard shapes

There are standard types of shapes that are specifically designed for **Always Free Tier**services. This is a micro VM that has just 1 CPU and 1 GB of memory.

Let's create a standard OCI compute instance:

1. Sign into the OCI console.
2. Open the navigation menu, select **Compute**, and then select **Instances**.
3. Click **Create Instance**.
4. Provide a **Name** and select a **Compartment** where you want to deploy it.
5. In the **Availability Domain** section of the **Configure placement and hardware** section, choose where you want to place the AD. Additionally, you can click on the **Choose a Fault Domain for this Instance** checkbox and select a **Fault Domain** from the drop-down menu.
6. Select an **Image** for the operating system that you want to deploy. We will talk about the different types of images in the next section. By default, it will be the latest **Oracle Linux** image.
7. In the **Shape** section, by default, the **VM.Standard.E3.Flex** shape type will be selected, which has 1 core OCPU, 16 GB memory, and 1 Gbps network bandwidth shape. Click on **Change Shape** if you want to change it to a different one. You can either select another type of standard shape or use the slider to change the allocated OCPU and memory for this default instance type. You can see an example of this in the following screenshot:

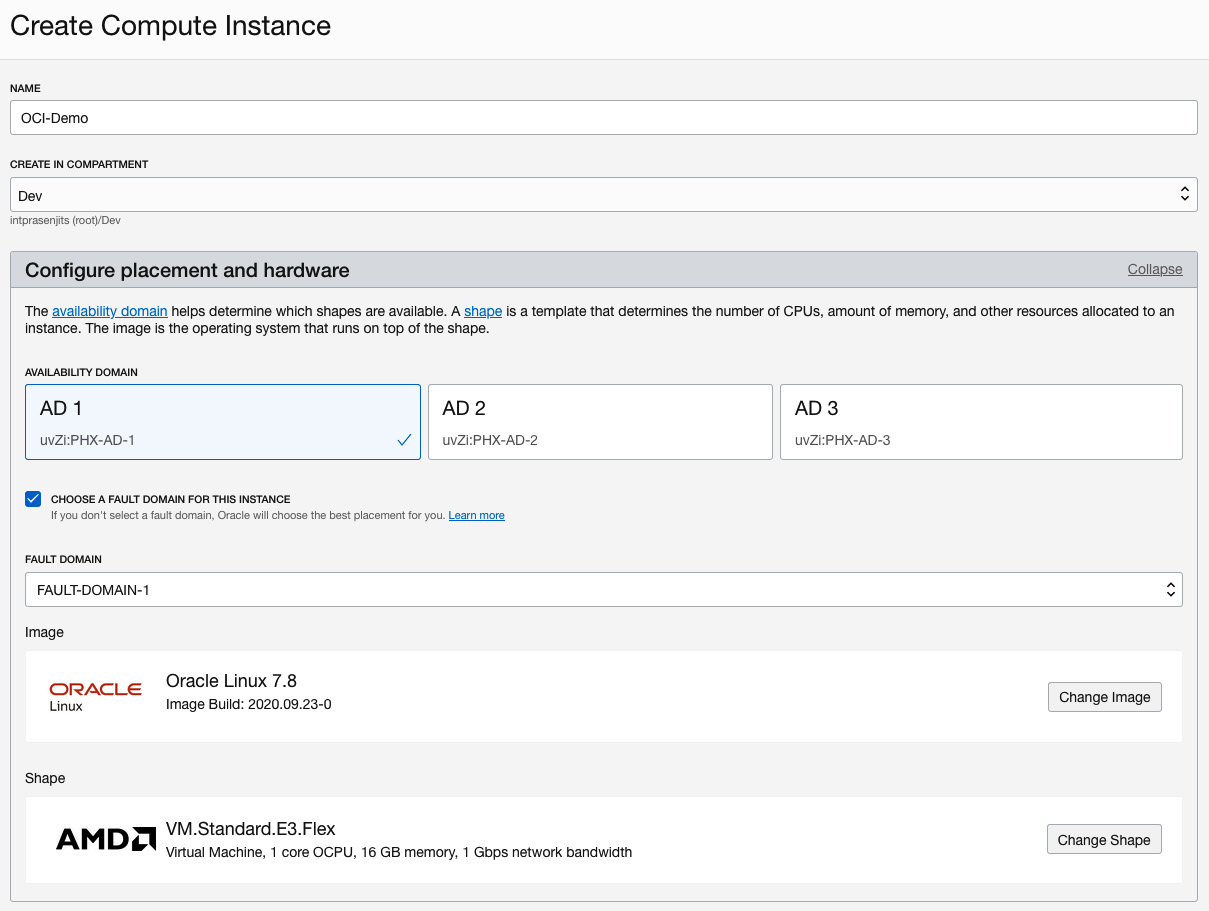


Figure 4.5 – Create compute instance wizard

1. In the **Configure Networking** section, choose the Virtual Cloud Network and the subnet that you want to connect this instance to.
2. Select the **Assign a Public IP Address** radio button to have access to this instance over the public internet.
3. Select **Generate SSH Key Pair** if you're unsure of how to generate an SSH key pair to access this instance.
4. Click on **Save Private Key** and **Save Public Key** so that you can use these keys to connect to this instance. You can use these keys to provision other instances in the future. You can see an example of this in the following screenshot:

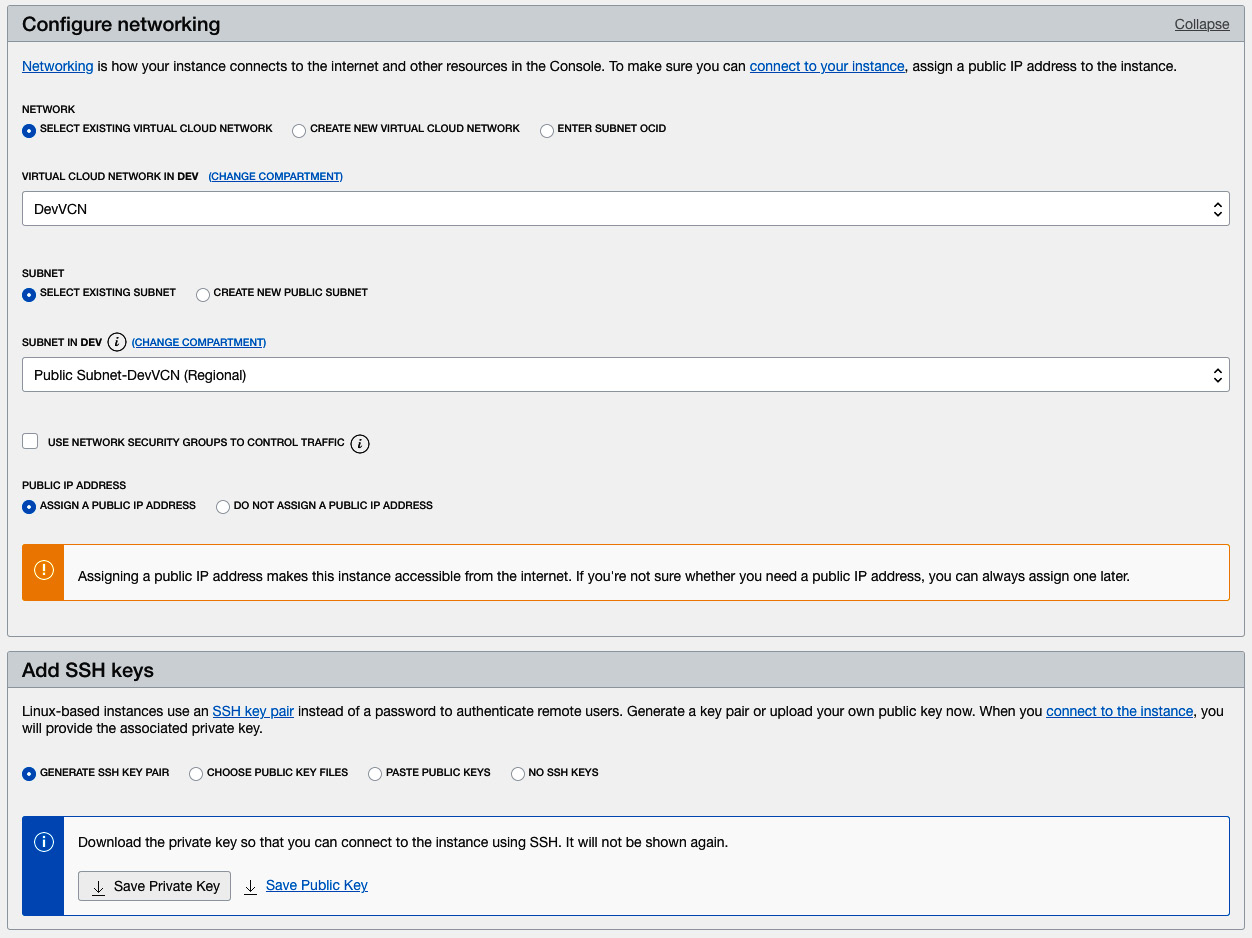


Figure 4.6 – Create compute instance wizard – Configure networking section

1. Optionally, you can **Specify a custom Boot Volume size**. While creating this instance, you can choose the default boot volume size of this instance or a custom boot volume size up to 32 TB.

If you are provisioning a Linux image-based instance, then you must set your custom boot volume size so that it's more than its default volume size, which is 50 GB. If you want to create an instance based on Windows operating system images, then the same rule applies; that is, you must set the custom boot volume size so that it's more than the default boot volume size, which is 256 GB.

The reason behind setting up this default boot volume size is so that you have enough space for Windows patches and a page file.

1. Optionally, you can specify **Use in-transit encryption**. In-transit encryption allows to encrypt the volume when it's being created.
2. Optionally, you can specify **Encrypt this volume with a key that you manage**.
3. Click on **Create**.
4. Once it is in a **Running** state, you must copy the instance's public IP address and connect to it. An example of this can be seen in the following screenshot:

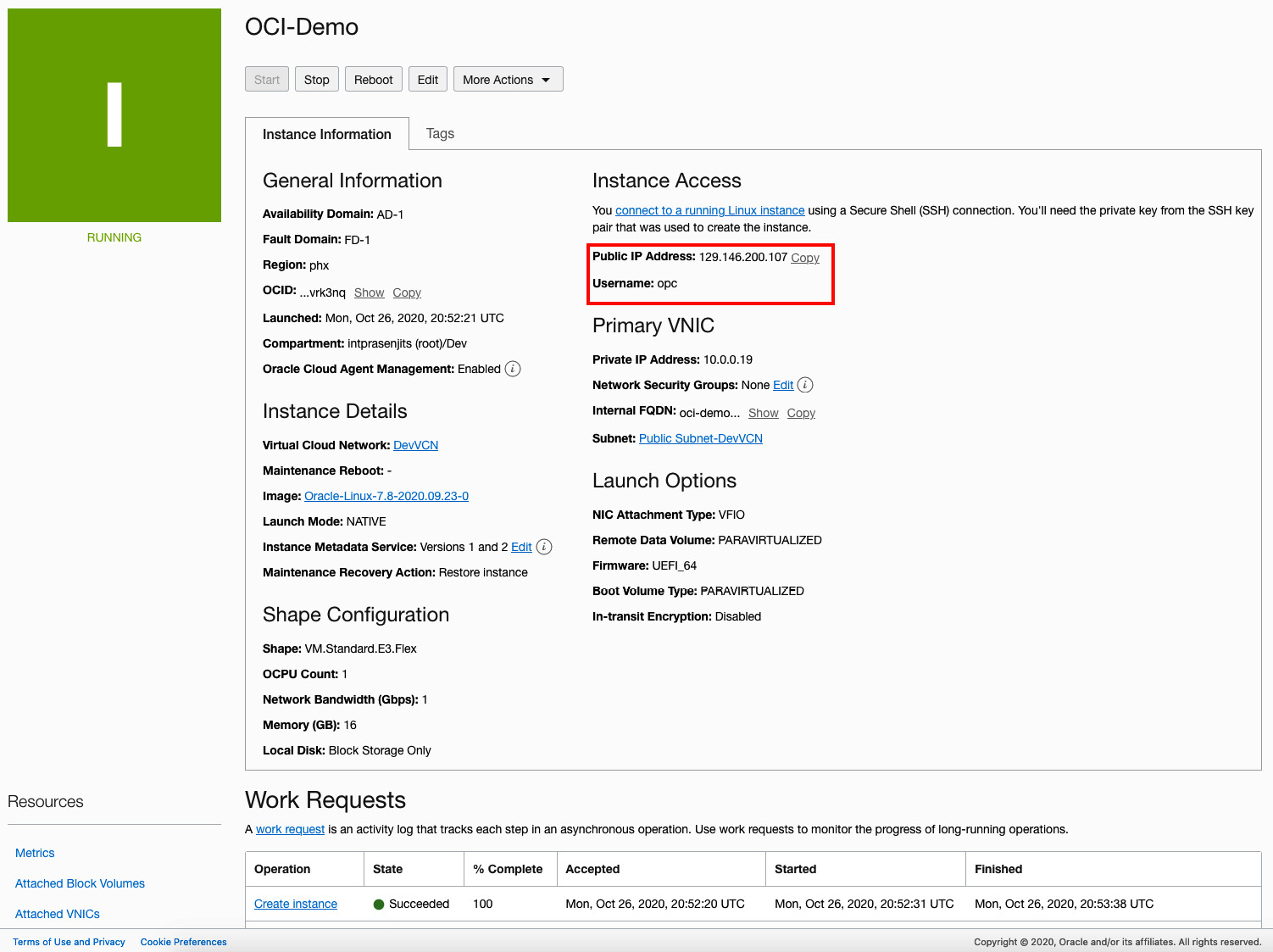


Figure 4.7 – Instance details

With that, you have learned how to create an instance using standard shapes. In the next section, we will talk about DenseIO shapes.

**DenseIO shapes**

When you design instances for giant applications such as large databases, big data applications, or if you are planning to run applications that have strict requirements in terms of high-performance local storage, then you need to go for DenseIO shapes. DenseIO shapes come with NVMe-based SSDs that are locally attached. You can get up to 52 physical CPU cores and 768 GB of memory. It also has 51.2 TB of locally attached SSDs and 2 x 25 Gbps networking bandwidth.

DenseIO shapes come in both bare metal and VM instance types. Let's create an OCI compute instance of the DenseIO type. There are extra steps that need to be performed, including creating a volume out of the available NVMe disk and formatting it with a filesystem before you can use that space:

1. To create an instance of the DenseIO type, follow the same procedure that we described in the *Standard shapes* section, but choose DenseIO as the image type. For this example, we will create an instance of **VM.DenseIO2.8**. You can see an example of this in the following screenshot:

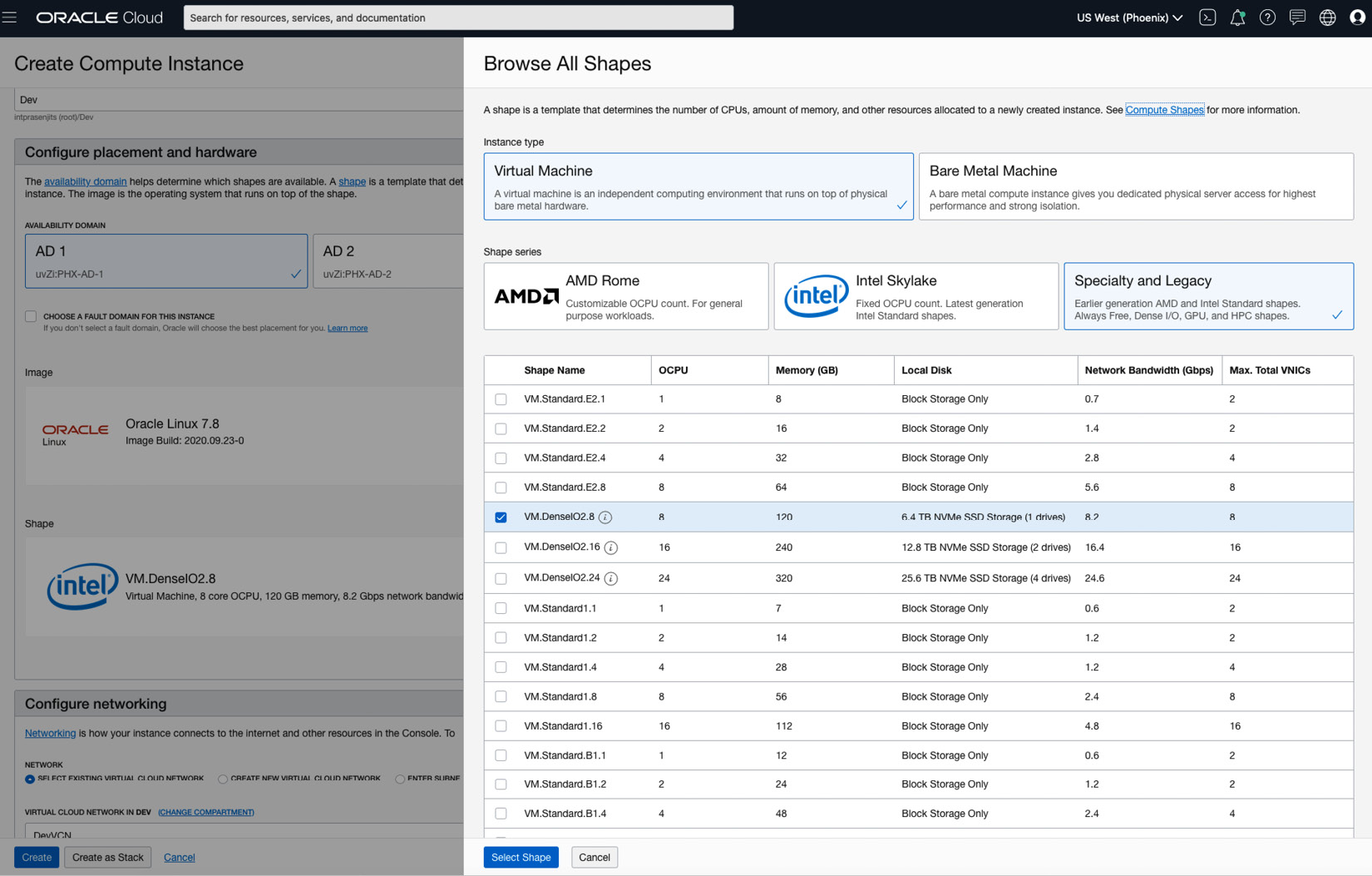


Figure 4.8 – DenseIO type of virtual instance

1. Copy the public IP address of this instance and log into it using **ssh**:

**ssh -i <path-of the-private-key> <username>@<public-ip-address>**

1. Once you've logged into the DenseIO instance, you need to create a filesystem and mount the NVMe disks. Let's do that now:

**sudo fdisk -l**

**sudo lsblk**

**sudo mkdir /mnt/nvme**

**sudo mkfs -t ext4 /dev/nvme0n1**

**sudo mount /dev/nvme0n1 /mnt/nvme**

You can see an example of the output in the following screenshot:

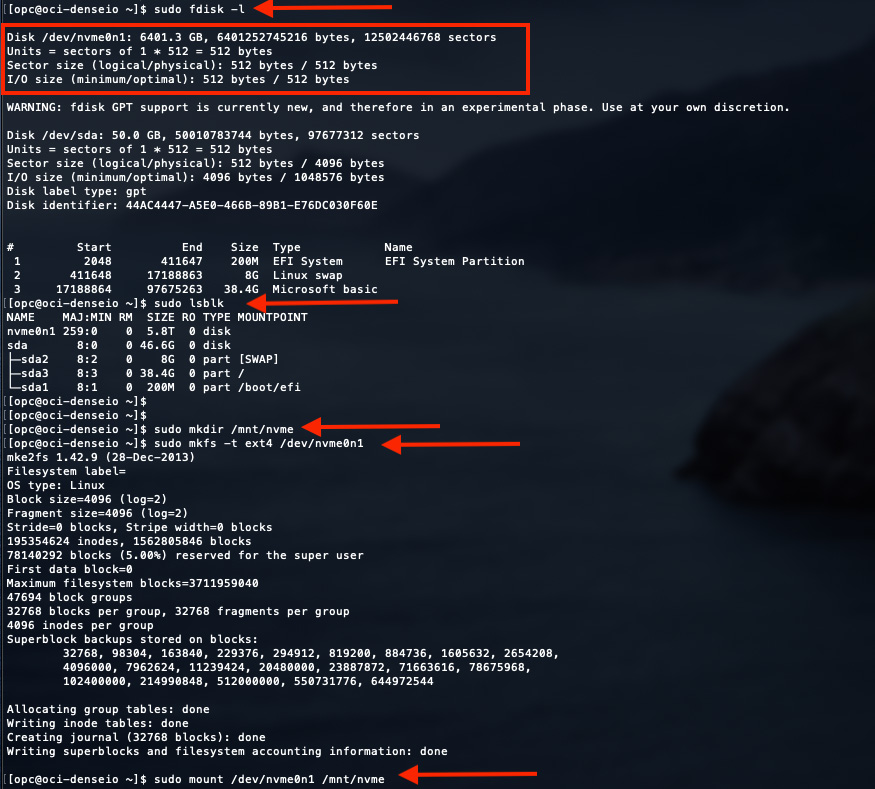


Figure 4.9 – DenseIO NVMe disk mounting

1. At this point, your disk has been formatted and mounted. However, it is not protected from hardware failures. For NVMe disk hardware failures, you should choose to create a software RAID solution on them.

In the next section, we will look at GPU shapes.

**GPU shapes**

GPU shapes are required when you plan to run hardware-accelerated workloads. GPU shapes come with processors from both Intel and AMD and NVIDIA graphics processors.

At the time of writing, there are three different types of GPU shapes available, as follows:

* 2nd-generation GPU shapes that have NVIDIA Tesla P100 GPUs
* 3rd-generation GPU shapes that have NVIDIA V100 GPUs
* 4th-generation GPU shapes that have NVIDIA A100 GPU chipsets

You can either choose a standard operating system and then install the NVIDIA GPU drivers on top of it or you can choose **NVIDIA GPU Cloud** (**NGC**). NGC provides you with GPU-optimized cloud instances that you can use for deep learning and scientific computing.

NVIDIA worked along with OCI engineering in an effort to provide cloud-ready Tesla, Volta, and Pascal GPU shapes. As a result, when you run these instances and then run NGC containers, you get optimum performance for deep learning applications.

Creating a GPU-based instance is same – you just need to choose the OS image and the GPU instance type and then create the instance. An example of this is shown in the following screenshot:

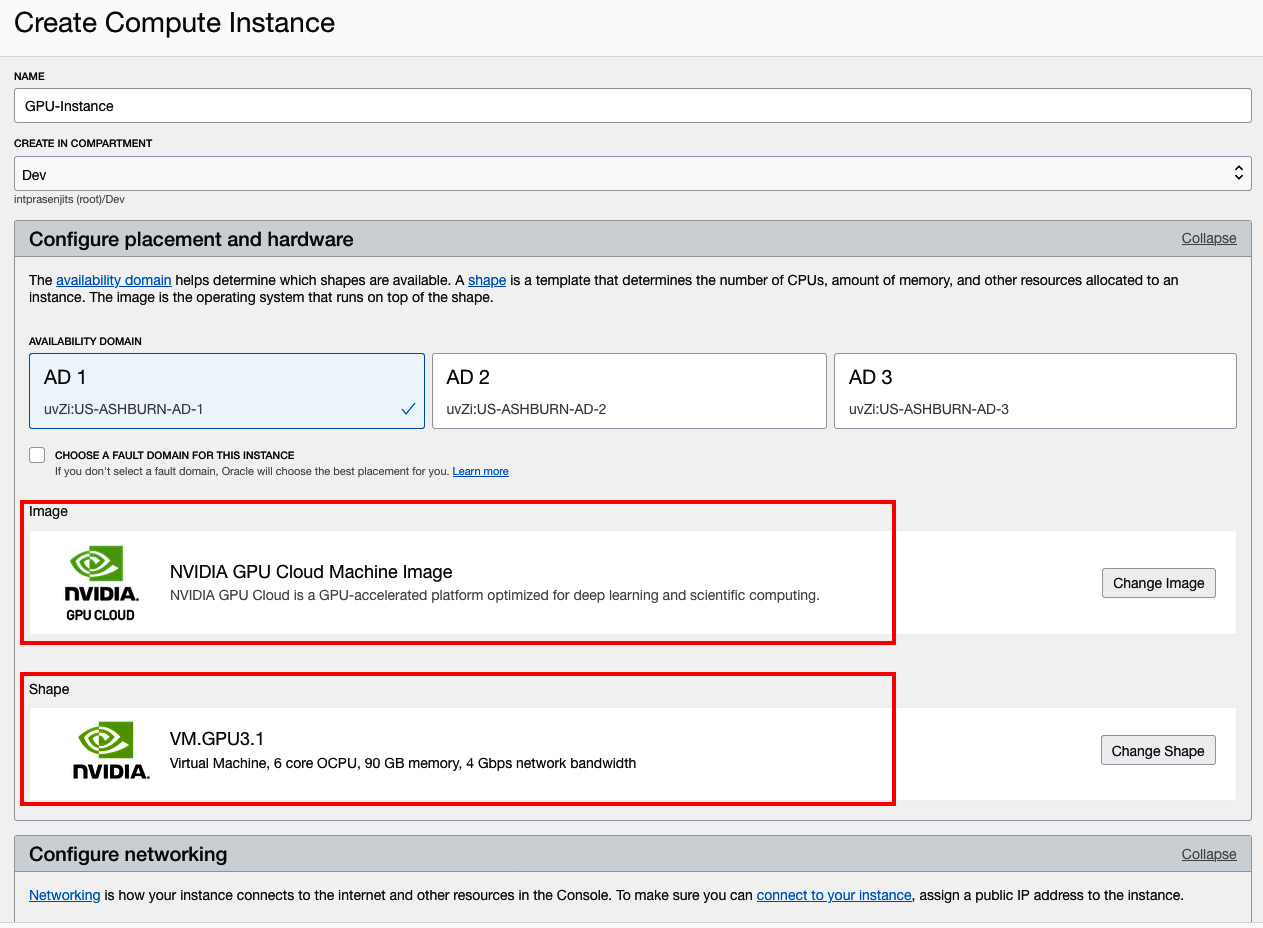


Figure 4.10 – NVIDIA GPU shape VM

In this section, you learned about GPU shapes. Now, let's discuss HPC shapes.

**High performance computing (HPC) shapes**

If you have an application that requires high frequency processor cores and cluster networking since this application will be performing parallel computing, then HPC shapes are the most suitable. You can only choose bare-metal instances for HPC shapes. You can see an example of how to select HPC shapes while creating a bare-metal instance in the following screenshot:

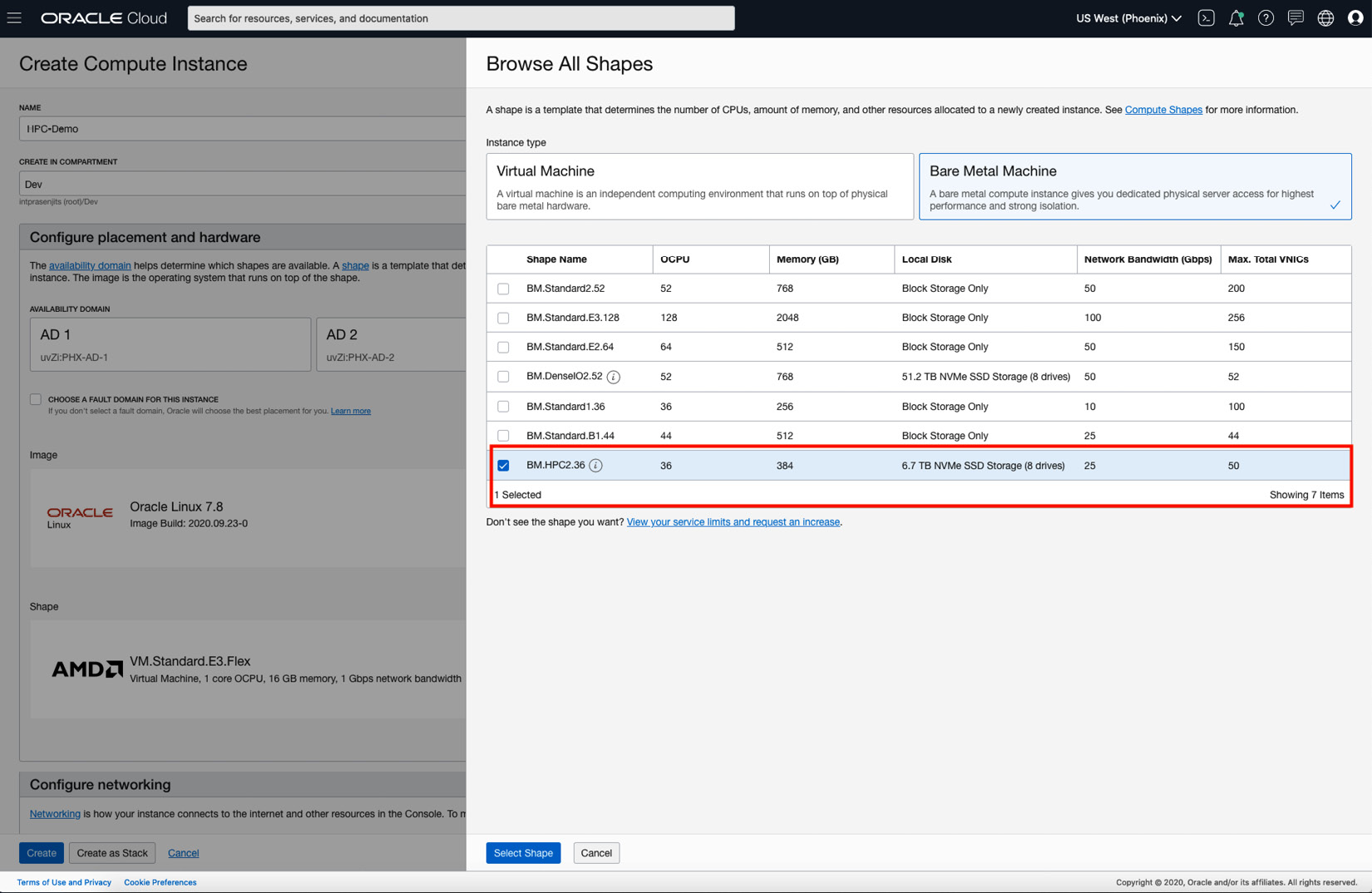


Figure 4.11 – HPC shape bare-metal instance

Most customers will go for some kind of external storage attachment for the compute shapes. In the next section, we will discuss those options.

**Storage for compute instances**

In the previous section, we discussed different types of instances and their different shapes. However, one thing that you must consider is the data persistence of these instances. None of these instances can give you data persistence. So, you need to rely on an externally attached storage type to persist the data. There are four different types of external storage that you can have, and they are as follows:

* **Block volume**: OCI block volumes are backed by iSCSI storage targets that are hosted on top of the same underlying network infrastructure. You can dynamically provision and manage block volumes. Furthermore, you can attach these block volumes to one instance or, for clustering purposes, to more compute instances. OCI also provides paravirtualized attachment for block volumes, which simplifies the configuration steps of block storage by removing the extra commands. However, if you want to provision block volumes for performance, then this is not ideal as it adds to the overhead of virtualization.
* **File storage**: OCI file storage is based on the NFS protocol. It is not only cost-effective and a durable storage solution, but it is highly scalable, secure, and an enterprise-grade network filesystem.
* **Object storage**: For unstructured data storage of any content type, you should choose OCI object storage. This is not only highly scalable but a high-performance storage platform.
* **Archive storage**: If you are looking to archive your data in an unstructured format, then the OCI archive storage platform is the best choice. The archive storage's scope is regional. It is not tied to any specific compute instance.

Although we will cover these specific storage types in the next chapter, let's learn how to create a block volume and then attach it to a compute instance and access the filesystem:

1. Sign into the OCI console.
2. Open the navigation menu, select **Block Storage**, and then select **Block Volumes**.
3. Click **Create Block Volume**.
4. Provide a **Name** and select a **Compartment** where you want to deploy it.
5. Select an **Availability Domain** where you want to deploy it.
6. Under the **Volume Size and Performance** section, select the **Default** option and make sure that the IOPS and throughput is proportionate to the volume's size.
7. Optionally, select **Backup Policy** in **Compartment**.
8. Click on **Create Block Volume**. You can see some example output of this in the following screenshot:



Figure 4.12 – Create Block Volume wizard

1. Once it has been provisioned, you can click on **Attached Instances** from the left-hand side tree.
2. Click on **Attach to Instance**.
3. Here, you can select an **Attachment Type**, which should be either **iSCSI** or **Paravirtualized**.
4. You can select an **Access Type** from here. By default, it is set to **Read/Write**.
5. Click on the **Select Instance** option and then select the instance that will be connected to this block volume from the dropdown.
6. Optionally, you can choose **Require CHAP credentials**.
7. Select a **Device Name** from the drop-down list.
8. Click on **Attach**. You can see some example output in the following screenshot:

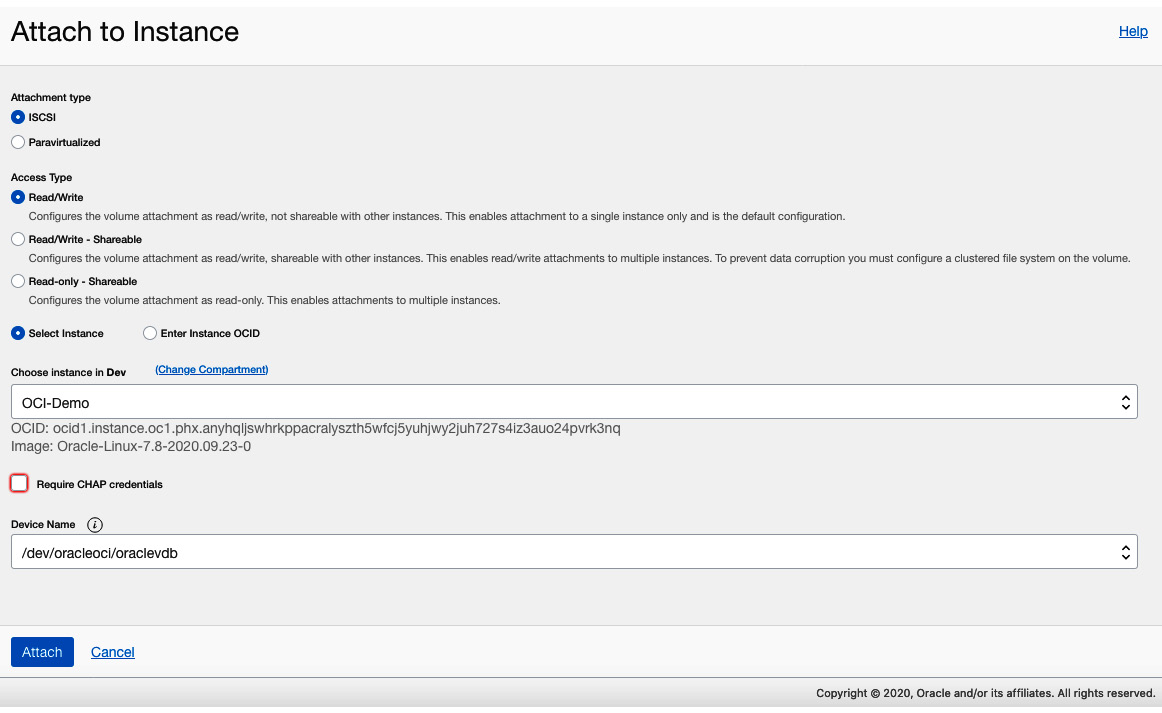


Figure 4.13 – Attach to Instance wizard

1. To attach this block volume to the operating system, you need to copy the commands that OCI provides. To do that, click on the three little dots on the **Attached Instances** page and select **iSCSI Commands & Information**. You can see some example output in the following screenshot:

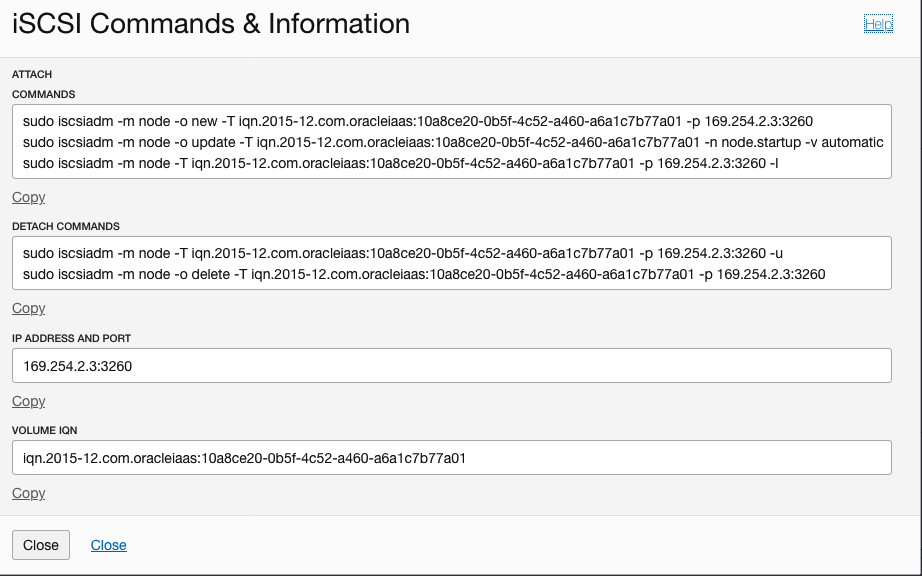


Figure 4.14 – iSCSI commands

1. Log into your instance via SSH if it is a Linux box and then run the commands shown in the previous step. You can see some example output in the following screenshot:

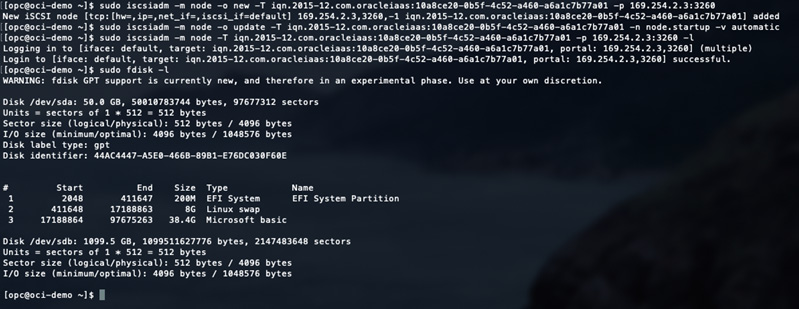


Figure 4.15 – iSCSI attach commands

1. Then, run the following standard commands to format this disk with the **ext4** filesystem type and mount it on a directory:

**sudo mkdir /mnt/block**

**sudo mkfs -t ext4 /dev/sdb**

**sudo mount /dev/sdb /mnt/block**

This way, you can create and attach a block volume to an instance and persist the data on it.

**Instance boot volume**

The instance boot volume comes from the same block storage backend where you launch an instance in any of the regions. You can see the boot volume in the same compartment where you have launched the instance. This boot volume gets associated with this instance. You can use it within this instance as this instance's boot volume or you can terminate this instance and keep this boot volume so that it can be used by another instance. That way, you can preserve the OS and its configuration and other data as well. You can use that in another instance if you wish to.

There are primarily two use cases for boot volumes, as follows:

* **Instance scaling**: In the previous section, we mentioned that you can keep the boot volume while terminating an instance to preserve it. The main use case here is to use this boot volume to create another instance of a different configuration. You can choose to either upgrade the resources that have been assigned to it or cut down on the resources. So, instance scaling is a perfect use case where the boot volume is a key feature. The good news is that you can even switch between VM and bare-metal instances using this boot volume.
* **Troubleshooting**: If you want to troubleshoot cases where instances are not booting up because of some issues in the boot volume itself, you can detach the boot volume and attach it to another instance as a data volume to troubleshoot this. Once you have finished troubleshooting, you can attach it back to the same old instance and boot up.

Let's look at using boot volumes in an instance:

1. Sign into the OCI console.
2. Open the navigation menu, select **Compute**, and then select **Boot Volumes**. Here you can see all the existing boot volumes and their details. You can see an example in the following screenshot:



Figure 4.16 – List of boot volumes

Let's terminate an instance but preserve the boot volume to spin up another instance of a different config.

1. Open the navigation menu, select **Compute**, and then select **Instances**.
2. From the list of instances, select the instance that you want to terminate, click on the three little dots, and select **Terminate**.
3. Do not select the **PERMANANTLY DELETE THE ATTACHED BOOT VOLUME** checkbox. Instead, click on **Terminate Instance**. You can see an example of this in the following screenshot:

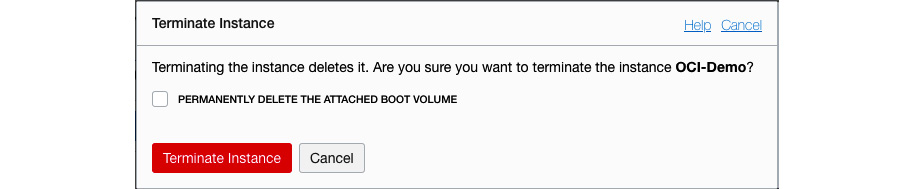


Figure 4.17 – Terminate Instance window

1. Once it has been terminated, you can go back to the boot volumes section (*step 2*) to see that the boot volume does exist.
2. Click on **Create Instance**.
3. Follow the same procedure for everything else, as described in the *Creating an instance* section.
4. In the **Image** section, click on **Change Image**.
5. Click on the **Boot Volumes** tab.
6. Select the boot volume from the list and click on **Select Boot Volume**. You can see an example of this in the following screenshot:

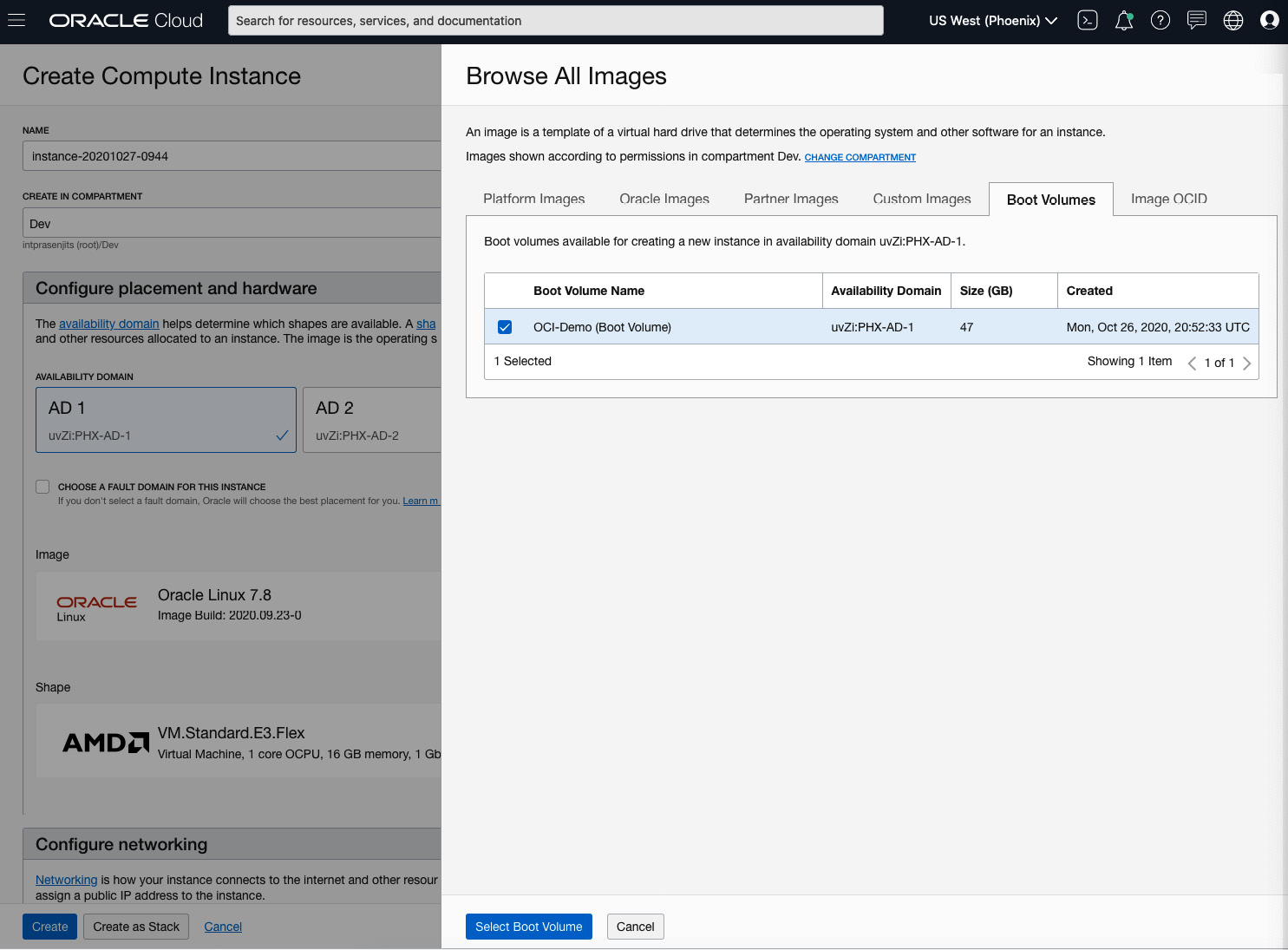


Figure 4.18 – Instance creation using an existing boot volume

1. Choose the shape that is appropriate for your workload and follow the configure of the options before clicking on **Create**.

With that, you have learned how to use the boot volume for various purposes. In the next section, we will talk about instance images.

**Understanding instance operating system images**

An instance image holds the operating system and other tools and software that go with it. Images can be Oracle-provided, Custom, or BYOI. You can choose an image from either Oracle Linux, Microsoft Windows, Ubuntu, or CentOS. As far as security concerns go within those images, Oracle adds rules within those images that restrict anyone else but the root on Linux instances (and administrators on Windows instances) from making outgoing connections to the iSCSI network endpoint (**169.254.0.2:3260**), which serves the instance's boot and block volumes.

For a maximum security posture, OCI's recommendation is not to tamper with any of the default operating system firewall rules as these may expose the risk of non-admins accessing the boot disk and other filesystems. This is also valid when you're creating a custom image from this.

Let's look at some of the characteristics of these images.

The following are characteristics of Oracle-provided Linux images:

* A new user, named **opc**, will be added to the instances that are created from Oracle Linux or CentOS images.
* A new user, named **ubuntu**, will be added to the instances that are created from Ubuntu images.
* By default, these users will get **sudo** privileges and can use SSH v2 to log into these instances.
* By default, an instance firewall will allow SSH access; that is, access to the instance over port **22**.
* Provide a startup script using **cloud-init**.

The following are characteristics of Oracle-provided Windows images:

* A new user, named **opc**, will be added to the Windows instances and for authentication, you will get a **one-time password** (**OTP**).
* These instances will have a Windows Update utility installed so that you can get the latest Windows updates from Microsoft.

Now, let's look at custom images.

**Custom images**

You can also use custom images. Think about a base OS image that you want to use to spin up hundreds of instances that has been tuned to your requirements, installed software, and other configurations. This is one of the main use cases for custom images. This is where you take an instance's boot disk to create and launch many other instances with similar OS configurations.

When you start creating a custom image from an existing instance, that instance goes through life cycle operations; that is, it shuts down on its own and remains unavailable until the ongoing operation is complete. This instance will start up when the process has completed.

However, you cannot create a custom image from the attached block volume. If you want to attach the secondary volumes, you will need to clone or back them up and then attach them to the newly created instance. A custom image cannot exceed 400 GB. For security reasons, you are not allowed to export Windows custom images out of the tenancy. You can see a logical representation of a custom image in the following diagram:

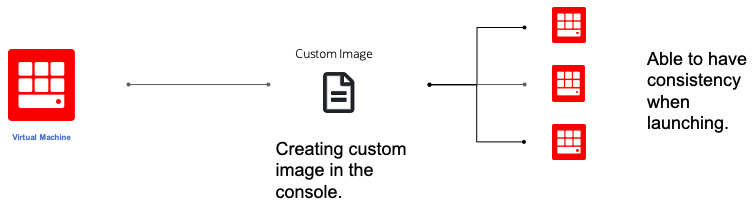


Figure 4.19 – Custom image

Let's create a custom image from an existing instance and use that to spin up another instance:

1. Sign into the OCI console.
2. Open the navigation menu, select **Compute**, and then select **Instances**.
3. Click on the VM that is going to be your base VM image.
4. On the **Instance Details** page, click on the **More Actions** menu and select **Create Custom Image**. You can see an example of this in the following screenshot:

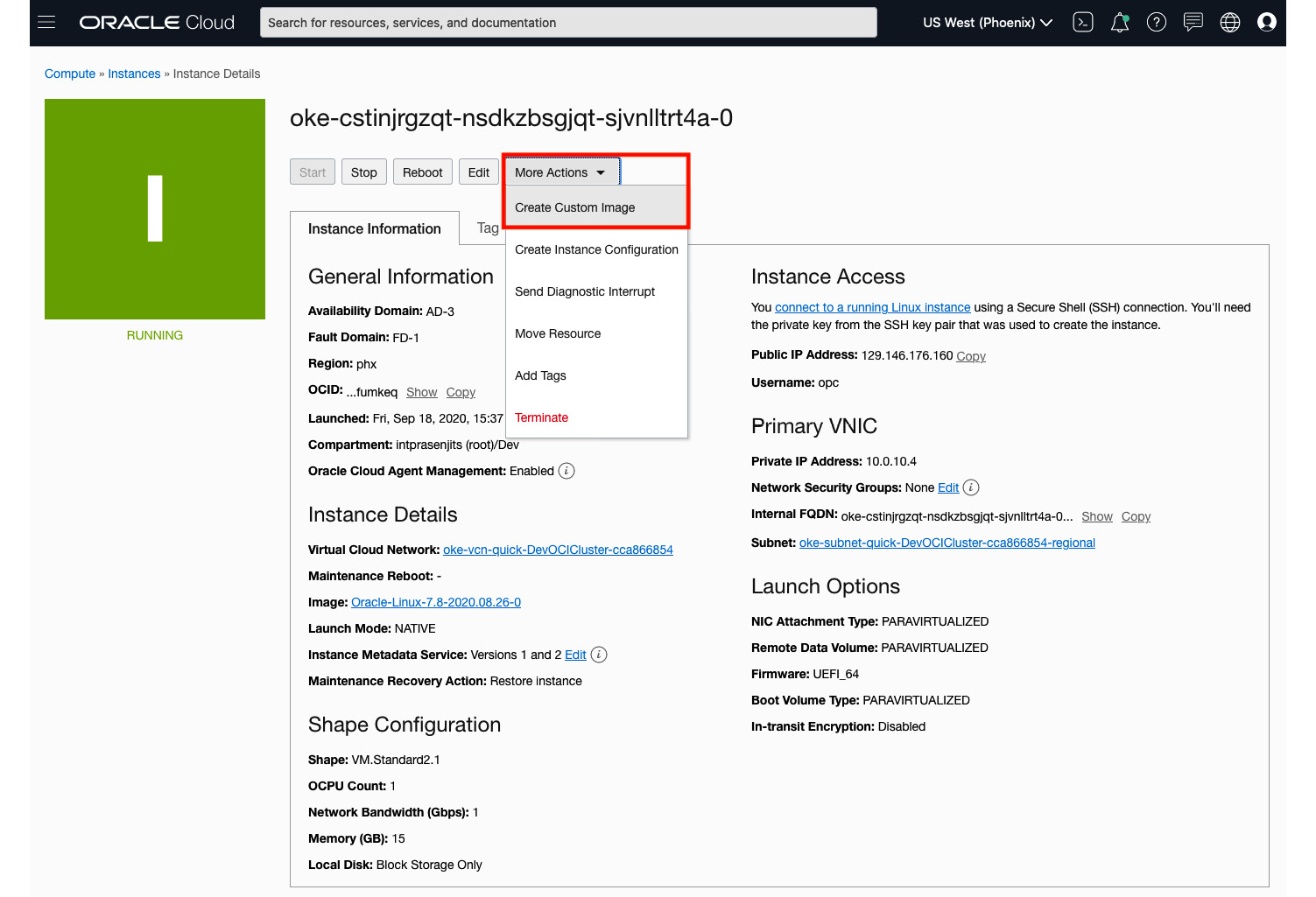


Figure 4.20 – Custom image creation

1. Select the compartment where you want to create this image and provide a **Name**.
2. Click on **Create Custom Image**. This is going to take some time. As described earlier, this VM is going to stop and start in the meantime as well.
3. Once complete, click on **Custom Images** from the **Compute** navigation menu. Here, you can see the custom image listed. You can see an example of this in the following screenshot:

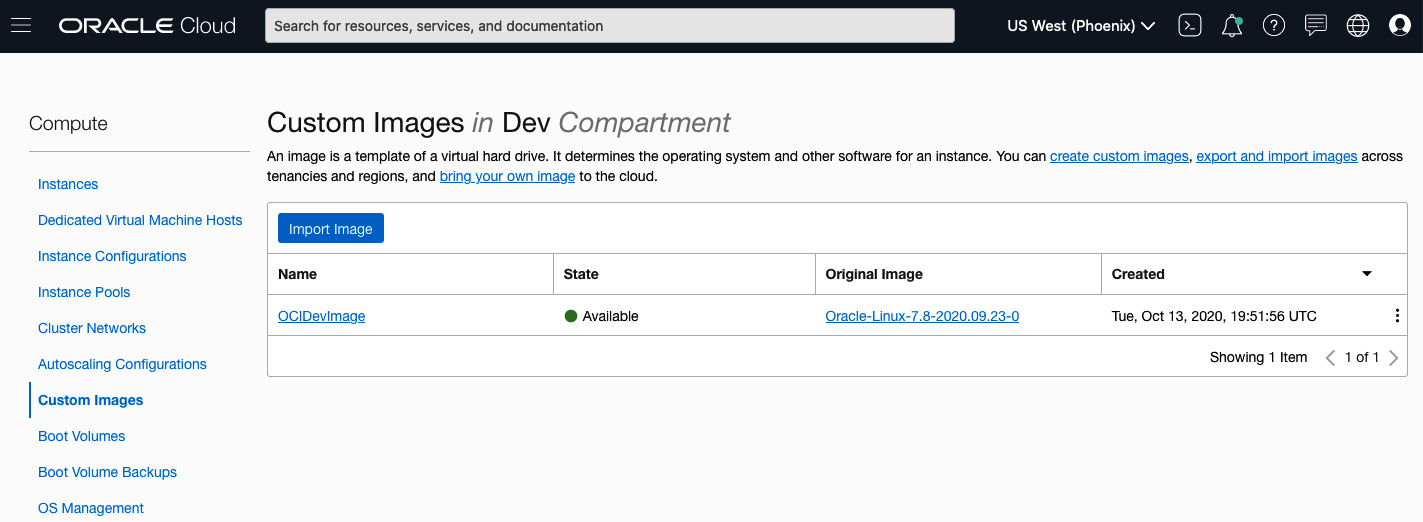


Figure 4.21 – Custom Images list

1. Let's use this image and create an instance. Go to the **Instances** section of the **Compute** navigation menu.
2. Click on **Create Instance**.
3. Follow the same procedure that was described in the *Understanding instance shapes* section.
4. In the **Image** section, click on **Change Image**.
5. Go to the **Custom Images** tab and select the custom image from the list. You can see an example of this in the following screenshot. Click on **Select Image**:

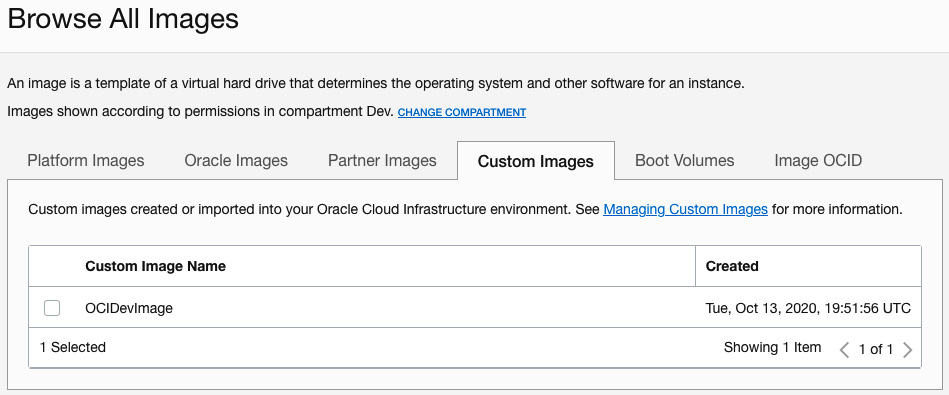


Figure 4.22 – Instance creation using a custom image

1. Click on **Create**.

In this section, you learned how to efficiently create a custom image from an existing VM and use that to instantiate other instances, all while keeping the same configuration.

**Image export and import**

The image export and import capability lets you share custom operating system images across different tenancies and OCI regions. Image import/export uses the OCI Object Storage service.

You can import Linux and Windows operating system. It supports the following mode:

* **Emulation mode**: In emulation mode, VM I/O devices, such as disk, network, and CPU and memory are implemented in software. Emulated VMs can support almost any x86 operating system. However, these VMs are slow.
* **Paravirtualized**: These VMs include a driver specifically designed to enable virtualization.
* **Native mode**: Similar to a **hardware virtualized machine** (**HVM**), it offers the maximum performance for modern operating systems.

Let's export an image using the console:

1. Sign into the OCI console.
2. Open the navigation menu, select **Compute**, and then select **Custom Images**.
3. Click the custom image that you want to export. Then, click **Export**.
4. Select **Export to an Object Storage bucket**.
5. Choose the object storage bucket from the drop-down list. Optionally, you can specify a name for this exported image. You can see an example of this in the following screenshot:

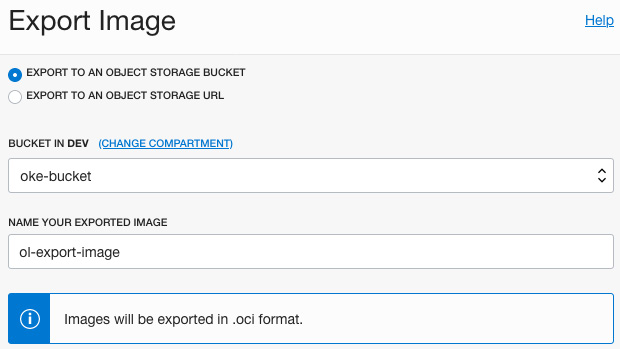


Figure 4.23 – Export Image wizard

1. Click **Export Image**.

With that, you've learned how to quickly export an image and use that in either a different region within the same tenancy or in other tenancies.

**Bring Your Own Image (BYOI)**

If you have an operating system that is not available on OCI, you can use the BYOI feature to get that onto OCI, as long as OCI supports the underlying hardware.

BYOI can help with the following scenarios:

* Lift-and-shift applications
* Support for old and new OSes
* Experimentation using different versions of an OS
* Maximum flexibility of the underlying infrastructure

*NOTE*

*When you bring in your own custom images, you need to comply with the licensing of those OS images on your own.*

The following is a logical diagram of the image export and import process. Here, a customer exports their on-premises image onto a qcow2 image and uploads it to OCI object storage. Then, they can create a custom image out of it and use it to create an instance:

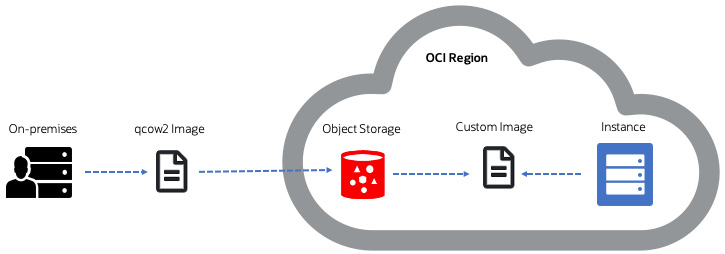


Figure 4.24 – Image export/import process

So, you've learned about exporting a custom image and importing it to OCI. In the next section, we will discuss the process of bringing your own hypervisor.

**Bring Your Own Hypervisor (BYOH)**

In a BYOH scenario, each guest VM can get one or more secondary **virtual NIC cards** (**VNICs**). If SR-IOV **virtual functions** (**VFs**) are used by the hypervisor to provide network access to the guest VMs, each VF can be configured with the VLAN tag and MAC address of a secondary VNIC. The guest VM can have a private IP and a public IP associated with it. However, make sure that you know the limits of the VNICs for each shape type.

So, to illustrate this, we can have multiple virtual network interfaces in different subnets and those VNICs can each be associated with a different guest VM. This can be seen in the following diagram:

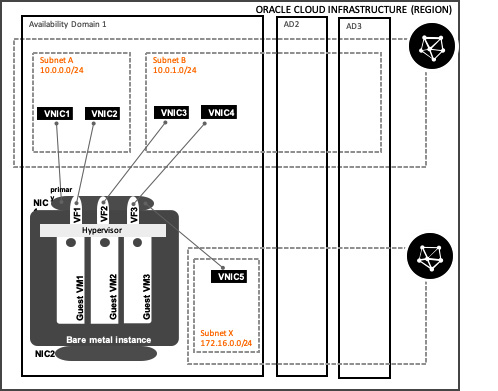


Figure 4.25 – Bring Your Own Hypervisor

In the preceding diagram, you can see how the VNICs are connected to different subnets and how that is translated for the guest VMs inside it.

OCI supports three types of hypervisor that you can bring over to OCI:

* **Oracle Virtual Machine** (**OVM**)
* KVM
* Microsoft Hyper-V

BYOH will give you the flexibility to customize your VM shape while selecting the amount of CPU and memory required for your workload.

Since this is a unique feature of OCI, you can leverage services such as block volume and network interfaces that can be directly attached to the virtual machines you created on top of your hypervisor. This allows them to communicate with other resources and instances provisioned in OCI. There are two benefits of this:

* You can extend the same configuration and the workload that's running on on-premises on OCI.
* Using this feature, you can install legacy OS images using pre-packaged VMs.

To speed up the hypervisor configuration, OCI offers a pre-built image for the KVM hypervisor. This image is part of the Oracle Partner catalog and provides a set of tools you can use to interact with your provisioned virtual machines.

At the time of writing, OCI supports BM Standard and BM. DenseIO shapes are used as part of the deployment.

**Creating similar instances using instance configuration and instance pools**

If you want to create a template configuration that will be used to create multiple OCI instances, then instance configuration is your friend. It is a template for configuration that we can use to create similar OCI instances. You can specify configuration for which OS image to choose, different shapes and their resource allocation, different block volume sizes, and so on.

These templates can be created using an existing OCI instance or using the OCI CLI. When you launch an instance using this instance configuration, OCI creates the resources that are defined within that template config.

You can see a logical representation of this in the following diagram:

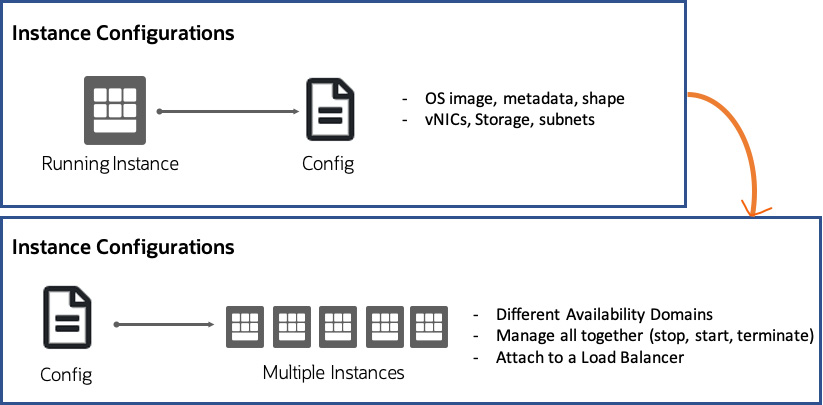


Figure 4.26 – Instance configuration

OCI uses this instance configuration to create and manage identical instances in a logical group. This is known as an instance pool. The main use case for an instance pool is horizontally scaling OCI instances. An instance pool scales these identical instances up and down based on either a schedule or performance metrics.

Let's look at the use cases for both of these options:

* **Instance configurations**:

- Create a configuration file out of an existing instance.

- Create a baseline configuration for instances.

- Use this configuration file to provision OCI instances using the OCI CLI as well.

- Instance creation automation using a base template for the configuration and resources.

* **Instance pools**:

- You can manage and scale a group of OCI instances using a configuration template.

- You can distribute these instances across different availability domains within a region to maintain high availability.

- You can scale out and in of these instances based on a schedule or performance metric.

With that, you've learned how instance configuration and instance pools both work together to create similar instances at scale. However, there are other factors in play here, such as scaling this configuration using metric-based scaling.

**Compute instance metrics**

Compute metrics are critical components of your architecture design. With metrics, you can see how your instances are performing based on CPU, memory, disk, and network usage. You can also take some actions based on the metrics that are displayed for each compute instance.

Each instance will provide information about all the components associated with it, such as network and storage components.

To monitor the overall compute instances, including their health, capacity and performance, you need to use **metrics**, **alarms**, and **notifications**.

Here is a logical representation of how to map the instances and their available metrics:

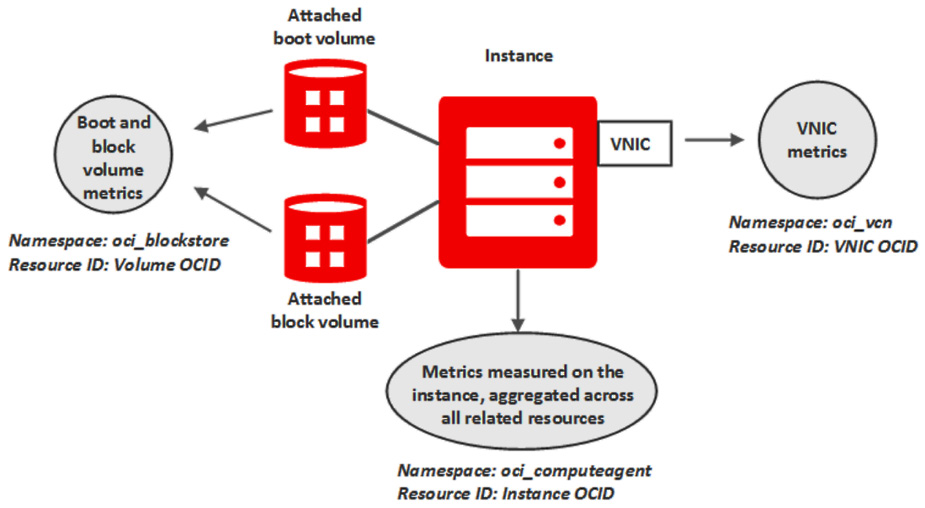


Figure 4.27 – Instance metrics

In this section, you learned how to combine instance configurations and instance pools, and then have a metric to trigger the instance creation automatically using them. In the next section, we will show you how to do this.

**Autoscaling configurations**

In the previous section, we explained how to use instance configurations and instance pools. In this section, we will talk about how you can combine those two to scale an instance pool in and out using autoscaling configuration.

Autoscaling lets you choose schedule or performance metrics such as CPU and memory utilization (this is handled by the OCI monitoring service) to automatically adjust the amount of compute instances in the pool.

There is an order that OCI sets up when the instances scale back in. This is how the extra instances are going to be deleted. First, OCI balances the number of instances across the availability domain and then narrows down to the fault domain. Even within a fault domain, when you need to scale back those instances, it chooses the oldest instance to delete first.

You have the option to choose which performance metric to choose when it comes to the autoscaling configuration. Then, you can decide on a threshold percentage, after which the autoscaler will start adding instances automatically. Once the threshold comes down below the threshold percentage, it will start deleting them.

The OCI monitoring service aggregates these metrics into 1-minute periods and then averages them across the instance pool. Autoscaling doesn't immediately kick in; it waits for the threshold to breach three consecutive times, which is the average of 1 minute's worth of data over 3 minutes. Then, the autoscaling event is triggered:

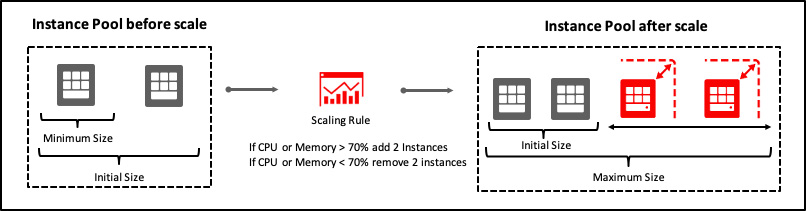


Figure 4.28 – Instance pool

Let's create an instance config and an instance pool and then perform the instance autoscaling configuration:

1. Sign into the OCI console.
2. Open the navigation menu, select **Compute**, and then select **Instances**.
3. Select the instance that you want to create a config from. Click on the **More Actions** menu on the **Instance Details** page and click on **Create Instance Configuration**.
4. Select the compartment where you want to create this config in from the dropdown and provide a **Name**.
5. Click on **Create Instance Configuration**.
6. You will be redirected to the **Instance Configuration Details** page. From here, click on **Create Instance Pool**.
7. Select the compartment a provide a **Name** for it. Select **Instance Configuration** from the dropdown and specify **NUMBER OF INSTANCES** to be provisioned. Click on **Next**. You can see an example of this in the following screenshot:

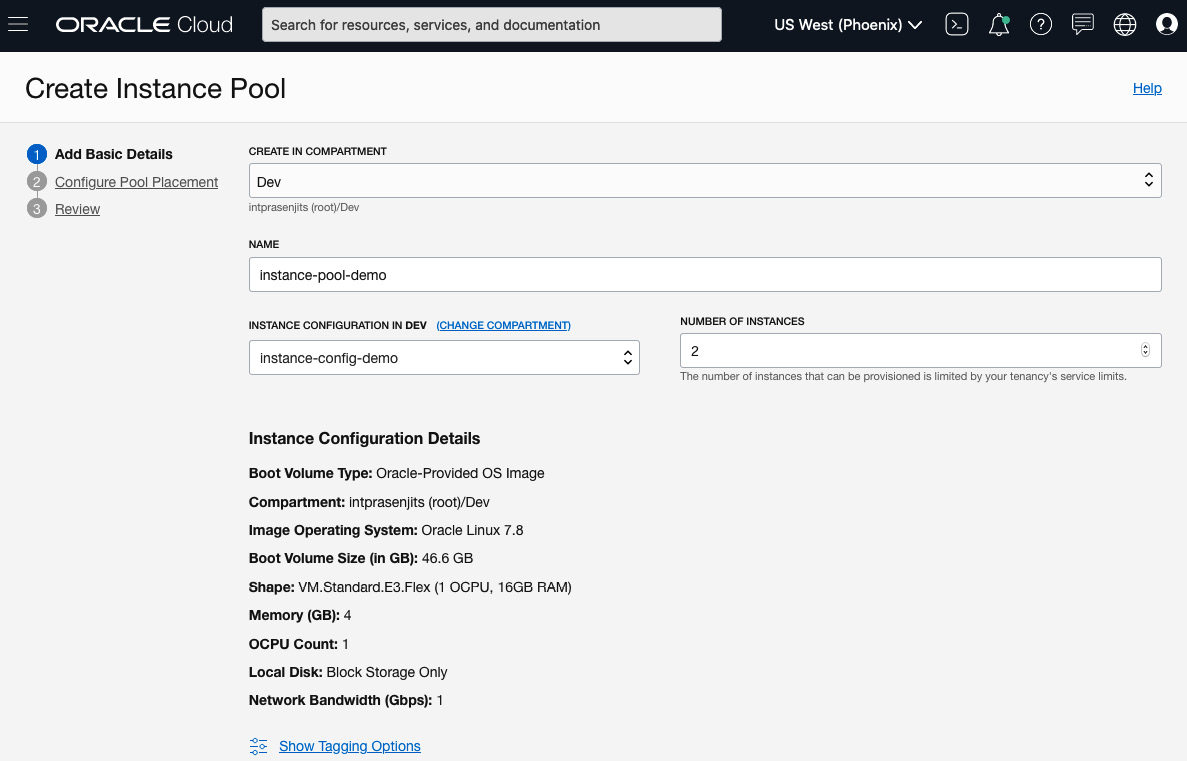


Figure 4.29 – Instance pool – Add Basic Details page

1. On the **Configure Pool Placement** page, select your **AD**, **FDs**, and which **VCN** and **subnet** you want to connect the primary NIC to. Since we have selected two instances, you can see that we are distributing it to two different ADs and two different FDs within them. Click on **Next**. You can see an example of this in the following screenshot:

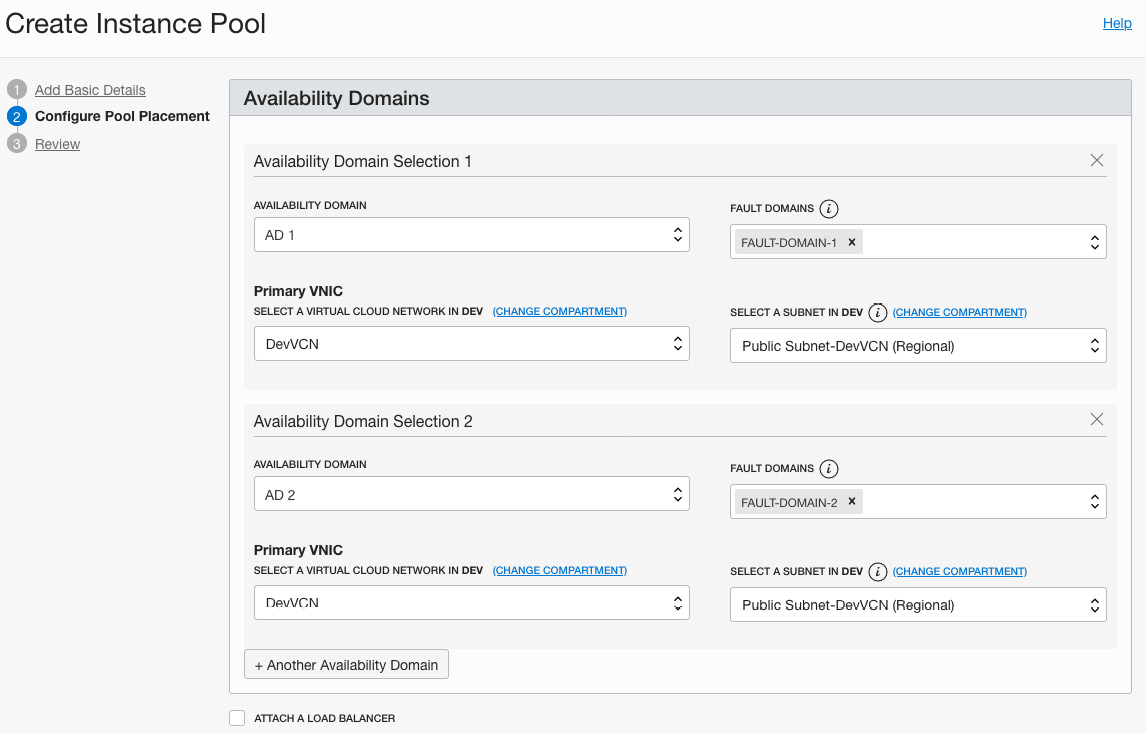


Figure 4.30 – Instance pool – Configure Pool Placement page

1. Review the details and click on **Create**.
2. You will be redirected to the **Instance Pool Details** page. From here, click on the **More Actions** menu and select **Create Autoscaling Configuration**.
3. Select the Compartment and provide a **Name** for it. Then, select the existing **Instance Pool** from the dropdown. Click on **Next**. You can see an example of this in the following screenshot:

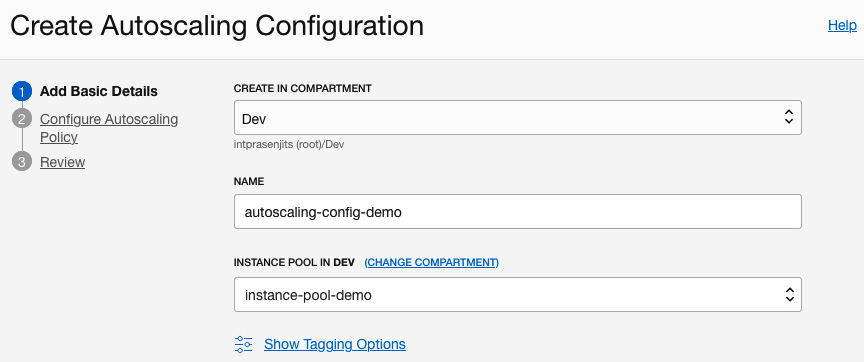


Figure 4.31 – Autoscaling – Add Basic Details page

1. In the **Configure Autoscaling Policy** section, select **Metric-based Autoscaling**.
2. Provide a **Name** and select **COOLDOWN IN SECONDS** (by default, it is **300** seconds).
3. Choose **CPU utilization** for **PERFORMANCE METRIC**.
4. In the **Scale-out rule** section, provide a threshold percentage (for example, 70%) that will be used to trigger the autoscaling function.
5. Provide the number of instances to add. For an example, when autoscaling starts, it will add **1** instance.
6. The same goes to the **Scale-in rule** section. Provide a threshold percentage such as 50%. So, in this case, when the compute instances are only 50% utilized, the instances that were scaled out will be scaled in.
7. Provide the number of instances to remove. For example, when scale-in starts, it will remove **1** instance.
8. In the **Scaling limits** section, provide the minimum number of instances to maintain. In this case, set it to **2**. Provide a maximum number of instances. In this case, we will provide **4** instance limits. Also, provide an initial number of instances. In this case, set it to **2**. You can see an example of this in the following screenshot:

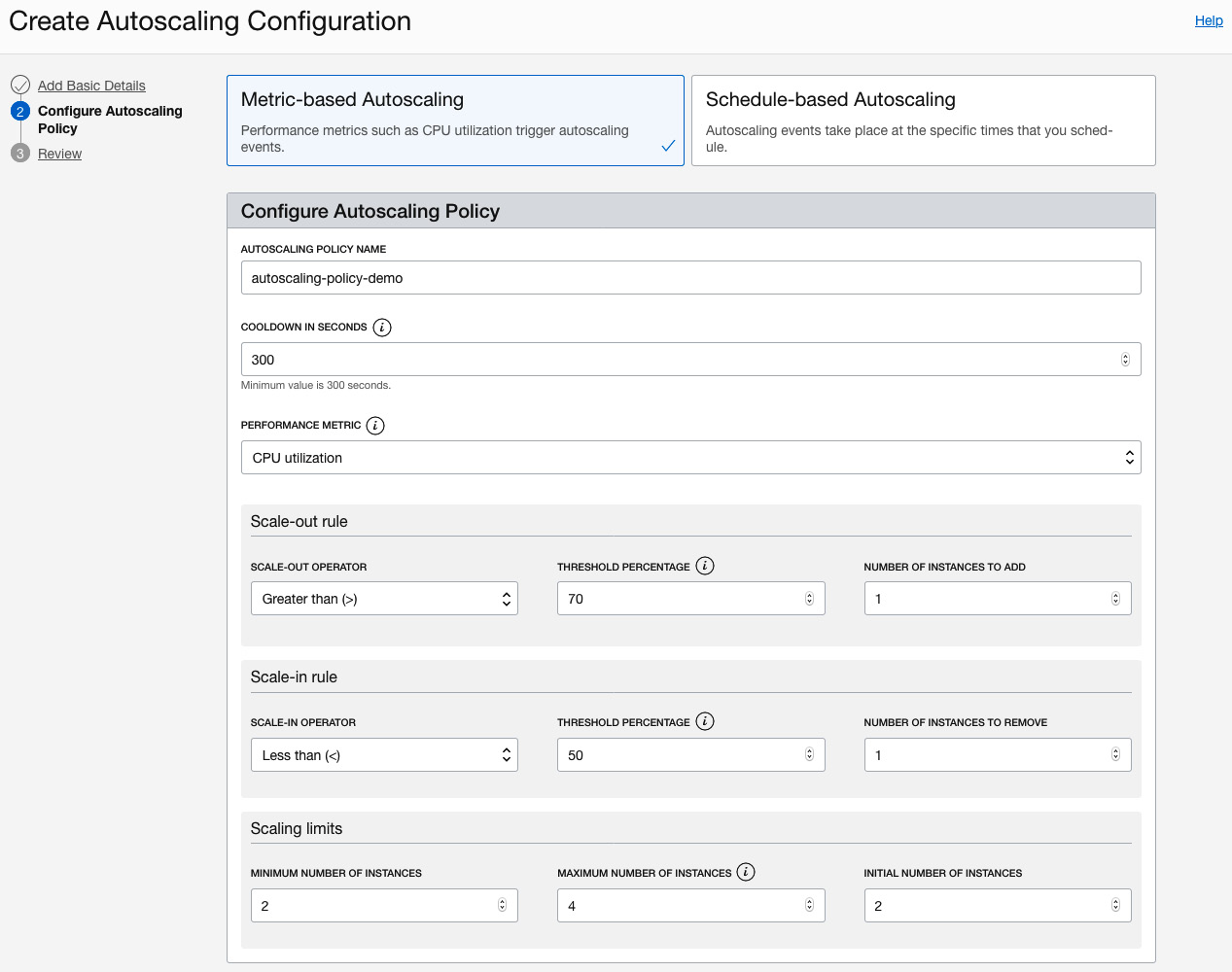


Figure 4.32 – Autoscaling – Configure Autoscaling Policy page

1. Click on **Next**.
2. Review the details and click on **Create**.

With that, you've learned how to create an autoscaling configuration and let OCI handle the horizontal scaling of OCI compute instances.

**Connecting to instances using an instance console connection**

When you're working with deploying compute instances, you can get in situations where the compute has failed to boot because of a misconfiguration in the OS, or there's a problem with a specific driver that could not load during boot. When this happens, your instance will be stuck and you won't have network access.

So, with instance console connections, you can access your instance in the same way you would using a serial console. You will be able to access an instance even if you don't have network connectivity and then gain access to the instance screen. From there, you can, for example, access the root filesystem to fix a misconfigured file or even replace your SSH key.

OCI offers two ways to connect to console connections: using SSH or using VNC.

Using this console connection, you can do the following:

* You can either add an SSH key or reset the SSH keys for the default **opc** user.
* You can edit system configuration files.
* When you import a customized image and that does not boot up, you can SSH to the console of it to troubleshoot.
* If there is an issue with an existing instance no longer responding, you can log into the console.

Let's create a console connection and then log into the instance console. Follow these steps:

1. Open the navigation menu, select **Compute**, and select **Instances**.
2. From the list of instances, click on the instance name that you want to create the console connection for.
3. On the **Instance Details** page, under the **Resources** section, click **Console Connection**, and then click **Create Console Connection**.
4. Specify the public key portion of the SSH key and then click **Create Console Connection**. You can see an example of this in the following screenshot:

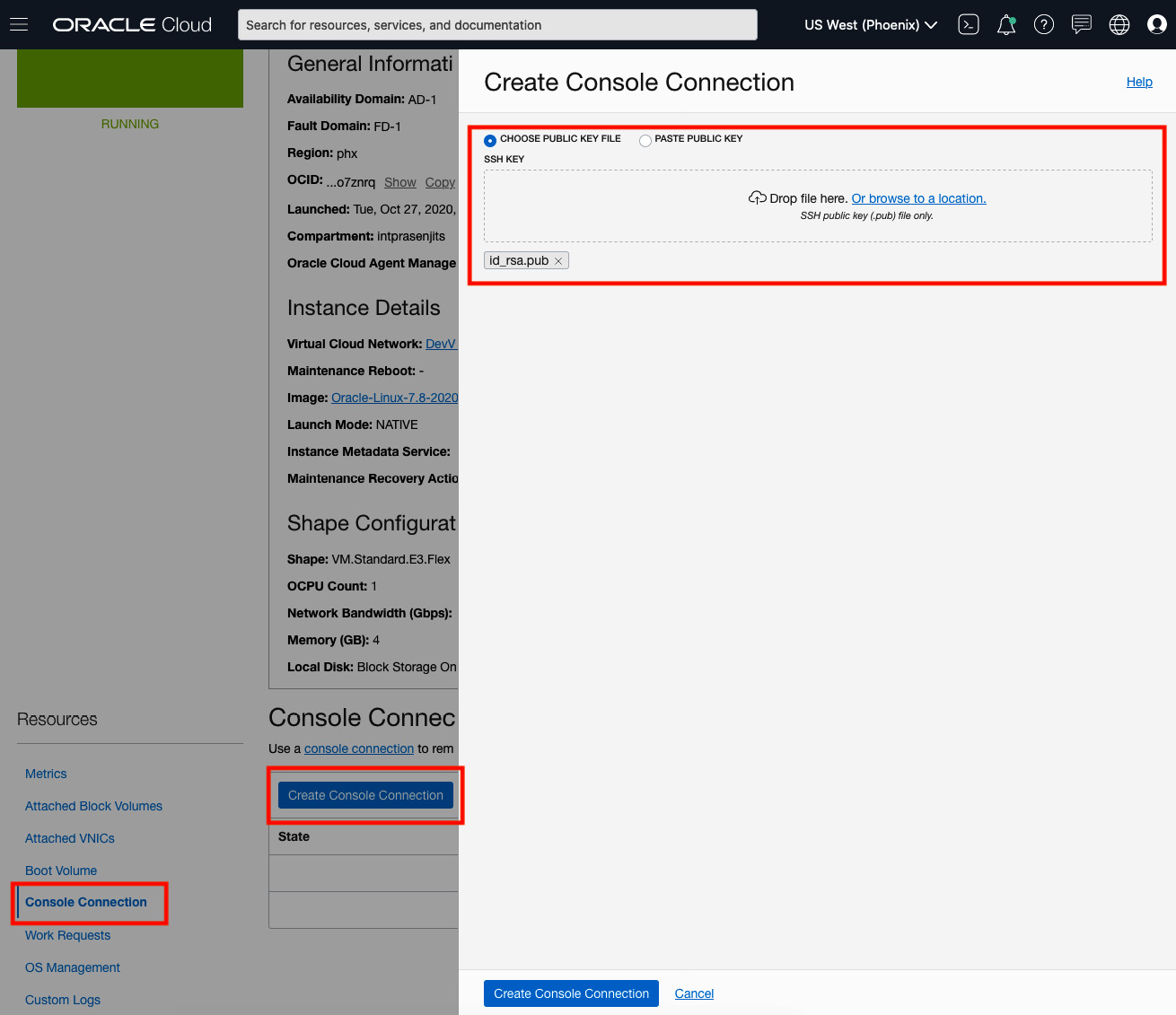


Figure 4.33 – Create Console Connection page

*NOTE*

*This SSH key can be different than the one you used to launch your instance.*

1. Once the console connection job has completed, its status will change to **ACTIVE**.

Now that you have created the connection, let's connect to the instance. We will show you how to do this in the next section.

**Connecting to the serial console from macOS or Linux OS**

You need to use an SSH client to connect to the serial console of the instance. By default, macOS and Linux distribution has an **ssh** client installed:

1. In the console, in the **Resources** section of the **Instances Details** page, click **Console Connections**.
2. Click on the **Actions** icon, and then click **Copy Serial Console Connection for Linux/Mac**.
3. Open a Terminal window and paste in the command that you have copied in *step 2*. Press *Enter*. You can see an example of this in the following screenshot:

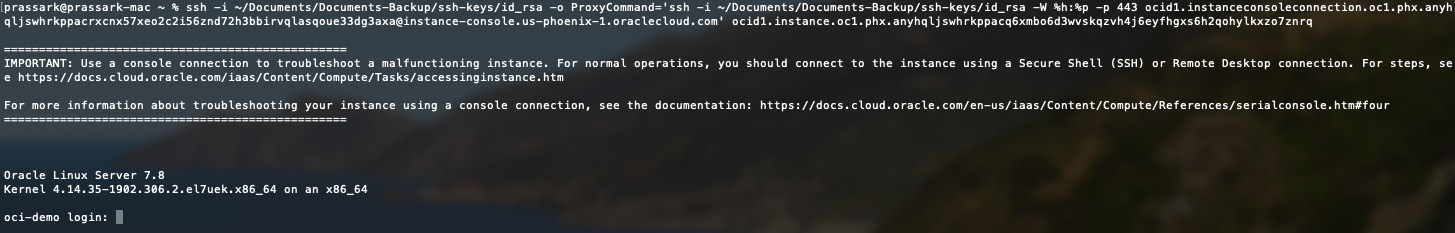


Figure 4.34 – Console connection from Mac

1. If you get a **Permission denied (public key). Key\_exchange\_identification: Connection closed by remote host** error, you should include the identity file flag, **-i**, in the connection string to specify the SSH key to use. You must do this for both the SSH connection and the SSH **ProxyCommand**. This is shown in the following code:

**ssh -i /<path>/<ssh\_key> -o ProxyCommand='ssh -i /<path>/<ssh\_key> -W %h:%p -p 443...**

For security reasons, you should delete the console connection once you have finished troubleshooting. If you do not do that, then OCI will terminate the serial console session after 24 hours.

**Summary**

In this chapter, you learned about the compute choices that you have in OCI. We explained the different types of compute instances that you can use, as well as the different type of shapes they can have. We also described how to use different OS images while creating these instances, such as Oracle-Provided images, custom images, and so on. You also learned how to export/import images for image portability across regions and tenancy. We then discussed instance config, pools, and autoscaling. Finally, you got to grips with console connections and how to use them to troubleshoot instance issues.

In the next chapter, we will cover the various storage choices that you have in OCI and how to leverage them in your use cases.