***Chapter 10*: Interacting with Oracle Cloud Infrastructure Using the CLI/API/SDK**

There are many ways to access **Oracle Cloud Infrastructure** (**OCI**). Although OCI has a console where you can create resources, it also offers an **Application Programming Interface** (**API**)-first approach. Using the OCI APIs you can do almost everything that you can do from the Console itself. The OCI APIs are nothing but **REpresentational State Transfer** (**REST**) API calls over **HyperText Transfer Protocol Secure** (**HTTPS**). But that is not all. If you want to use a **Software Development Kit** (**SDK**), you can do so to access all OCI APIs, with SDKs for Java, Ruby, Python, Golang, and .NET.

Apart from the REST API and the Console, you also have the option to invoke the CLI to perform the same operation. To invoke the OCI CLI, you have the option to either locally install the CLI binary or you can use OCI Cloud Shell to do the same thing. The benefit of using the OCI CLI from Cloud Shell is that you don't need to use security credentials to send the API calls to the OCI API backend.

Apart from the OCI Console, REST API, SDKs, and CLI, you can also control and provision resources on OCI using Oracle Resource Manager, which uses Terraform at the backend and not only that, you can also use Ansible. With Ansible, not only can you provision infrastructure resources; it also gives you the ability to deploy and update software assets.

In this chapter, we will walk you through how to use the CLI, covering both how you install it onto your local workstation and how to use it from OCI Cloud Shell. We will also see how to use SDKs to send API calls to the OCI backend, as well as how to use a REST API client to send the same calls to create or update resources.

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

* Using the OCI CLI to interact with OCI resources
* Using OCI SDKs to automate OCI operations
* Using the OCI API to send REST calls for managing OCI

**Using the OCI CLI to interact with OCI resources**

At the beginning of this chapter, we discussed how you can use other tools to access and create resources on OCI. Here's a quick recap of the fundamentals to show you the different methods to access OCI to create, update, and delete resources. You can see a logical diagram of how these tools are connected to OCI:

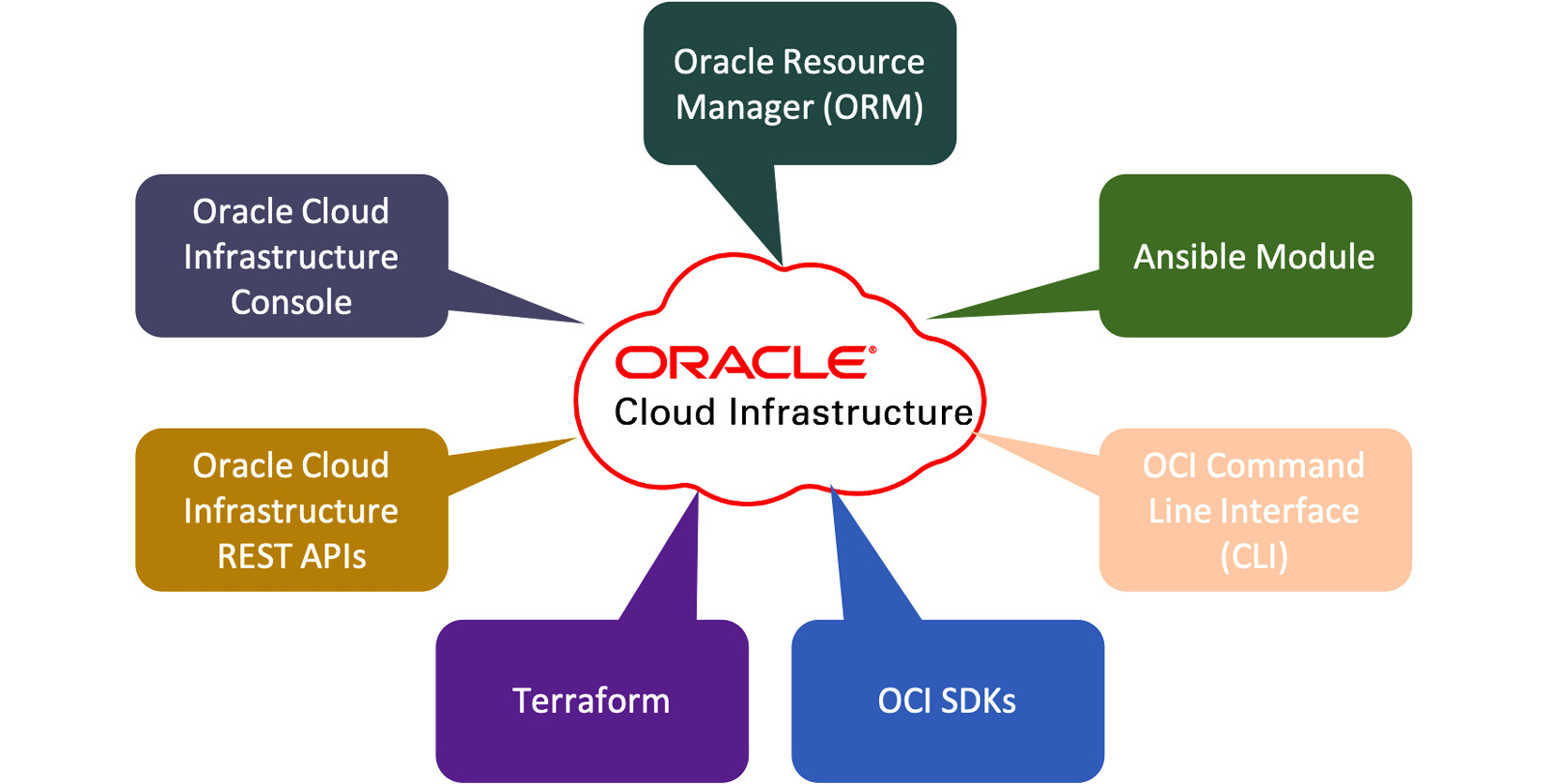


Figure 10.1 – Access methods for OCI

The OCI CLI is a small program that you can either run on a standalone workstation of your choice or you can use OCI Cloud Shell to run the same binary. As stated earlier, the CLI is the easiest way to automate tasks that you can do from the Console.

Let's look at the OCI CLI to see how you can use it to create, update, and delete resources. To install and use the OCI CLI on your local workstation, you need these on the OCI side:

* An OCI account
* A user with the desired permissions
* A key pair for signing API requests with the public key uploaded to OCI

We will use an OCI Linux instance to install the OCI CLI and configure it with permissions to send the calls.

Let's create a standard OCI compute instance:

1. Sign in to 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 availability domain. 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 a 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:

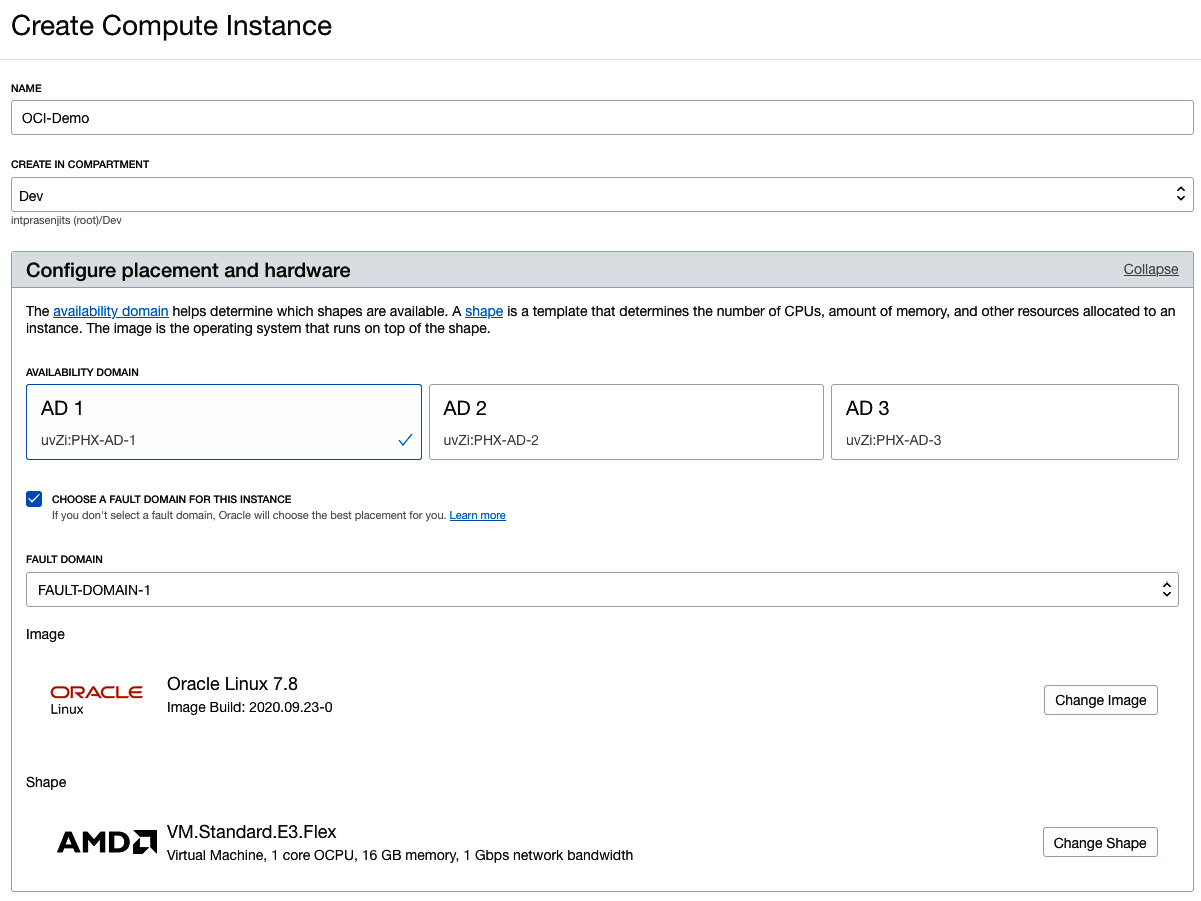


Figure 10.2 – Create Compute Instance wizard

1. In the **Configure networking** section, choose the **Virtual Cloud Network** (**VCN**) 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:

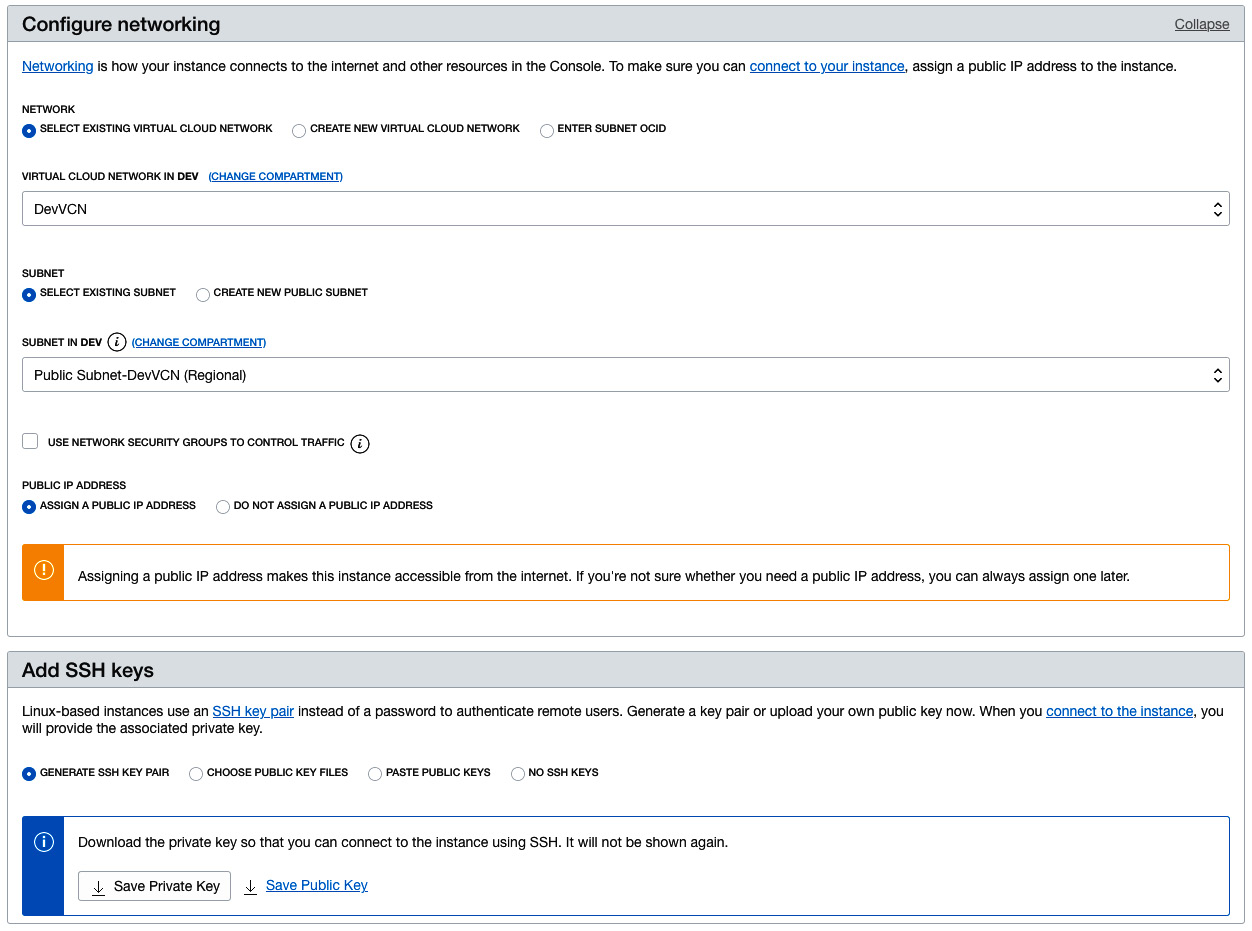


Figure 10.3 – Create Compute Instance wizard – Configure networking section

1. Optionally, you can choose **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 pagefile.

1. Optionally, you can specify **Use in-transit encryption**. In-transit encryption allows you 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 the **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:

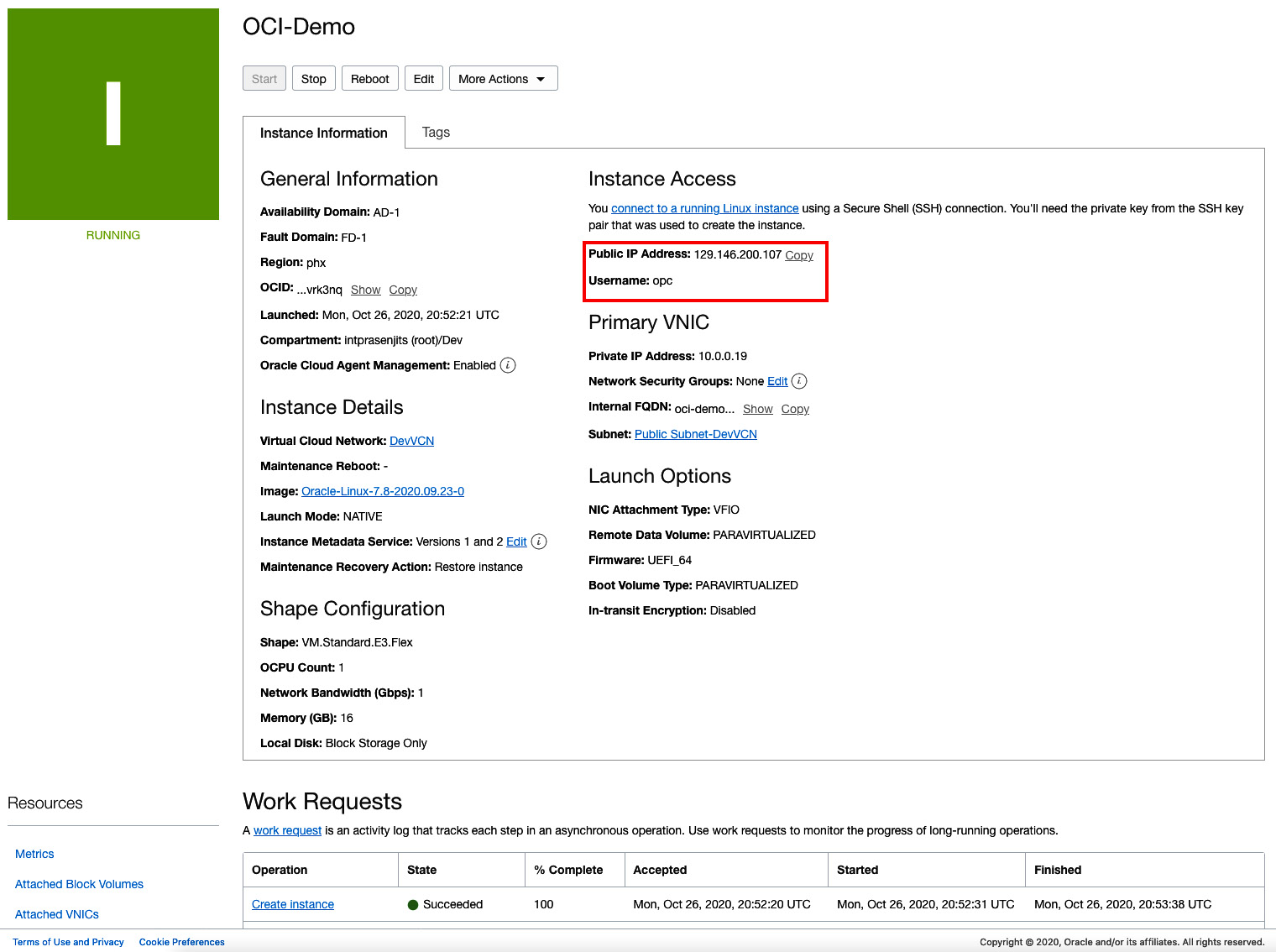


Figure 10.4 – Instance details

As the instance is up and running now, let's log in to it using an SSH terminal of your choice. You need to specify the SSH key to log in to the instance. Run the following command if you're using a macOS or Linux Terminal:

$ ssh -i /path-to-the-ssh-private-key opc@<Public-IP-Address>

Once you log in to the instance, you then install the OCI CLI binary. Let's run the installer script:

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

You need to respond to the installation script prompts. You can see an example of the process in the following screenshot:

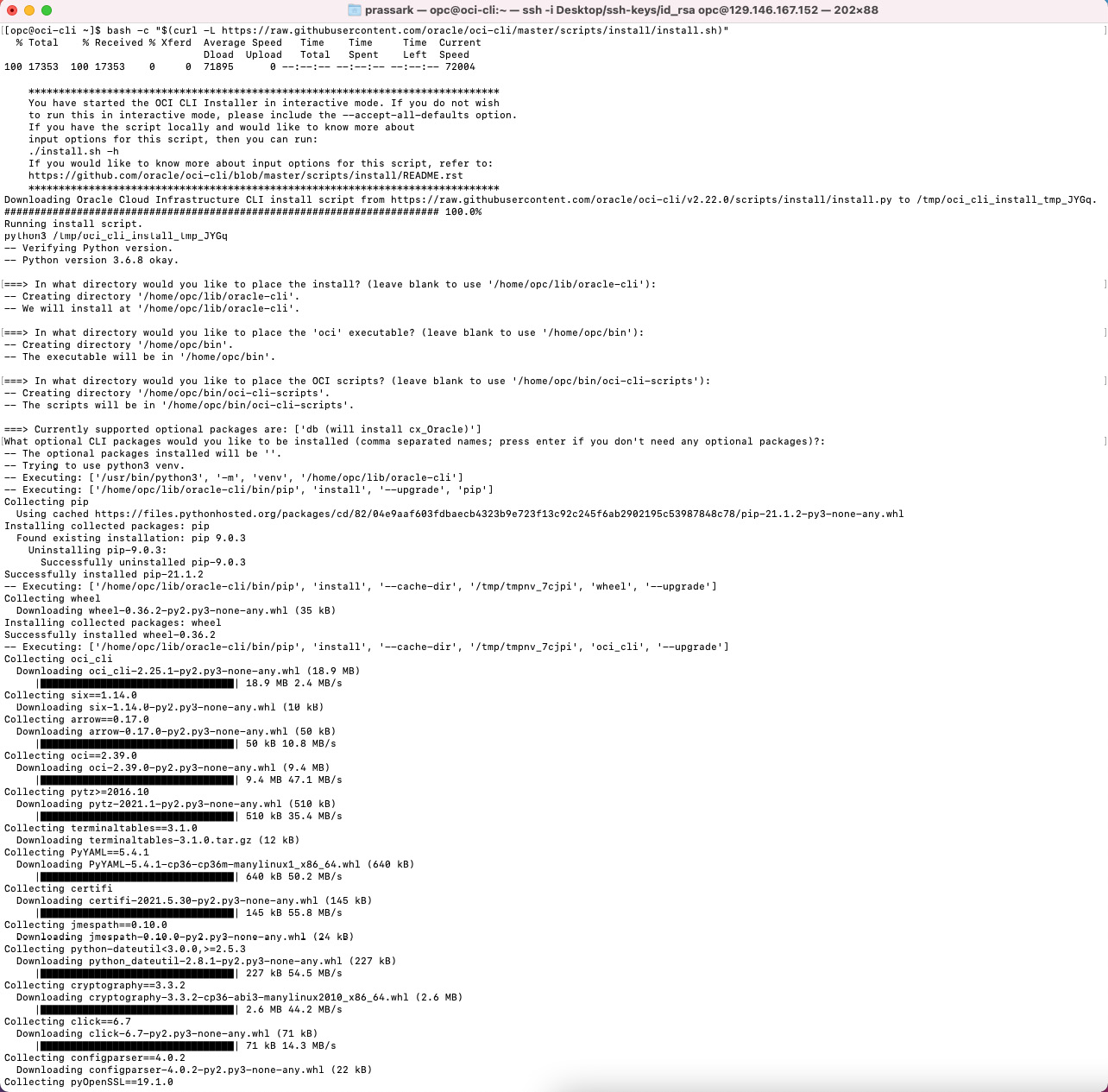


Figure 10.5 – The OCI CLI install script running

Before you can actually use the OCI CLI, you need to set up a config file that will hold information about your tenancy. Run the following command:

$ oci setup config

Follow the prompt and provide details, that is, your user OCID and tenancy OCID, and generate an API signing RSA key pair if you do not have one already. You can see some sample output in the following screenshot:

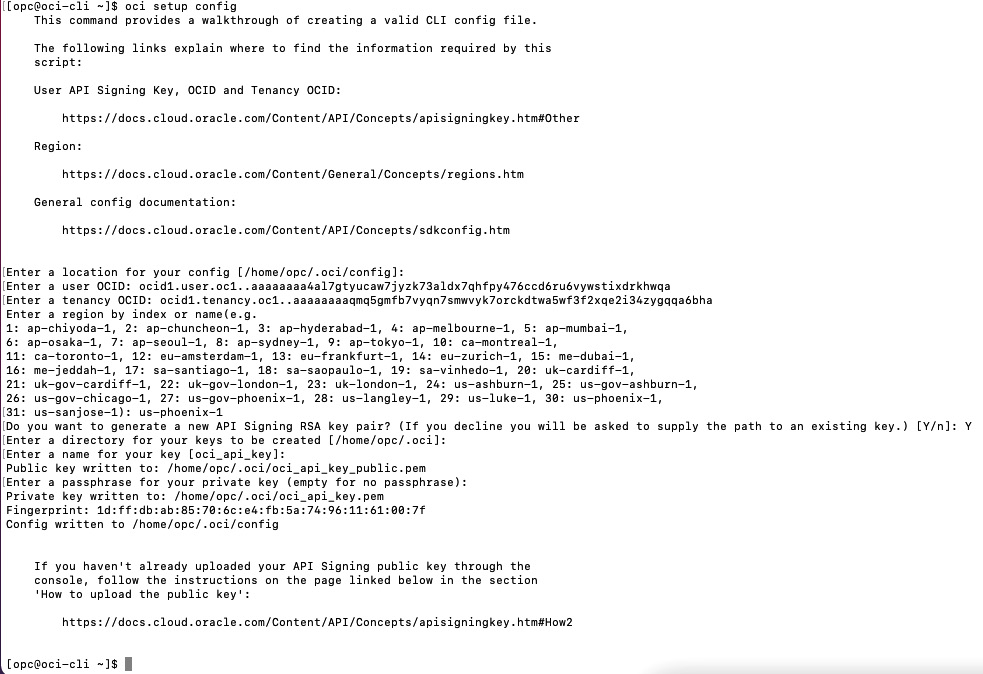


Figure 10.6 – The OCI CLI config file creation

You need to upload the public API signing key to your tenancy so that you can be authenticated for sending OCI CLI calls. Let's upload the public API key to the OCI Console and then try to send a sample OCI CLI command to verify:

1. Sign in to the OCI Console.
2. From the right corner, select **Profile**, and then select **User Settings**.
3. Go to the **API Keys** section.
4. Click on **Add API Key**.
5. Choose **Paste Public Key** and paste the content of the **oci\_api\_key\_public.pem** file from the Linux instance.

Now run a sample command to check the object storage namespace of your tenancy. To do that, run the following command:

$ oci os ns get

{

  "data": "intprasenjits"

}

This validates that the OCI CLI setup is complete. Let's use the CLI to create and remove some resources.

The basic syntax of OCI is as follows:

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

If you want to list the compartments, then run the following command:

$ oci iam compartment list -c <tenancy-ocid>

You can see some sample output in the following screenshot:



Figure 10.7 – The OCI CLI compartment list output

*NOTE*

*Make sure you replace the***<tenancy-ocid>***with the tenancy OCID of your tenancy.*

Let's create an object storage bucket in a given compartment using the OCI CLI. Run the following command:

oci os bucket create --name testbucket --compartment-id <compartment-ocid>

You can see some sample output in the following screenshot:

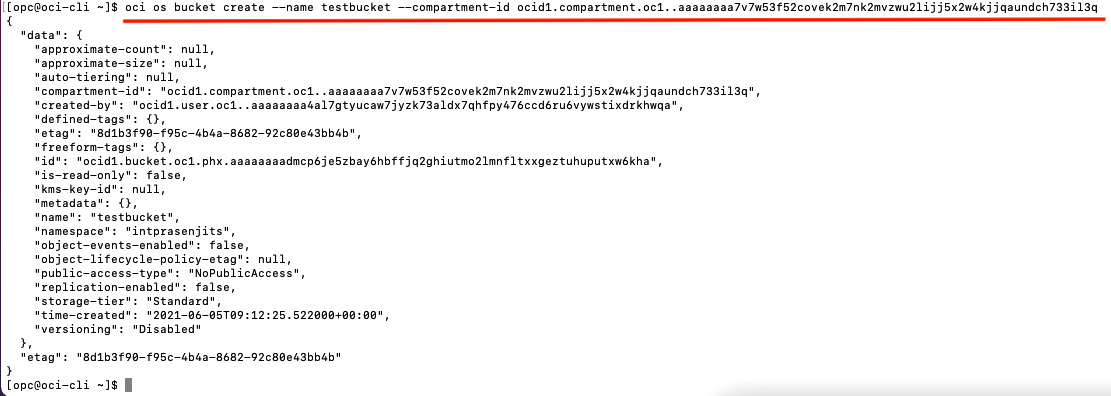


Figure 10.8 – Creating an object storage bucket

So, you can see how easy and interactive it is using the OCI CLI to create, update, or delete resources. OCI allows its customers to use the OCI CLI not only from the local workstation but by using its Cloud Shell as well.

OCI Cloud Shell is a terminal that is accessible from the Oracle Cloud Console. OCI doesn't charge its users to use this but there is a monthly tenancy limit. You can get a Linux shell prompt where the OCI CLI is installed and configured using your tenancy session credentials. So, you need to generate separate keys and configure the OCI CLI to authenticate itself.

Cloud Shell provides the following:

* An ephemeral virtual host that you use for the Linux shell. This host has the latest version of the OCI CLI installed and configured along with other useful tools, including Terraform, Ansible, and Docker.
* You will get 5 GB of storage in your home directory. You can store any useful files here, such as code or scripts.

Let's use OCI Cloud Shell to delete the object storage bucket that you have just created using the OCI CLI from the local workstation:

1. Sign in to the OCI Console.
2. From the right corner, select **Cloud Shell**, as shown in the following screenshot:

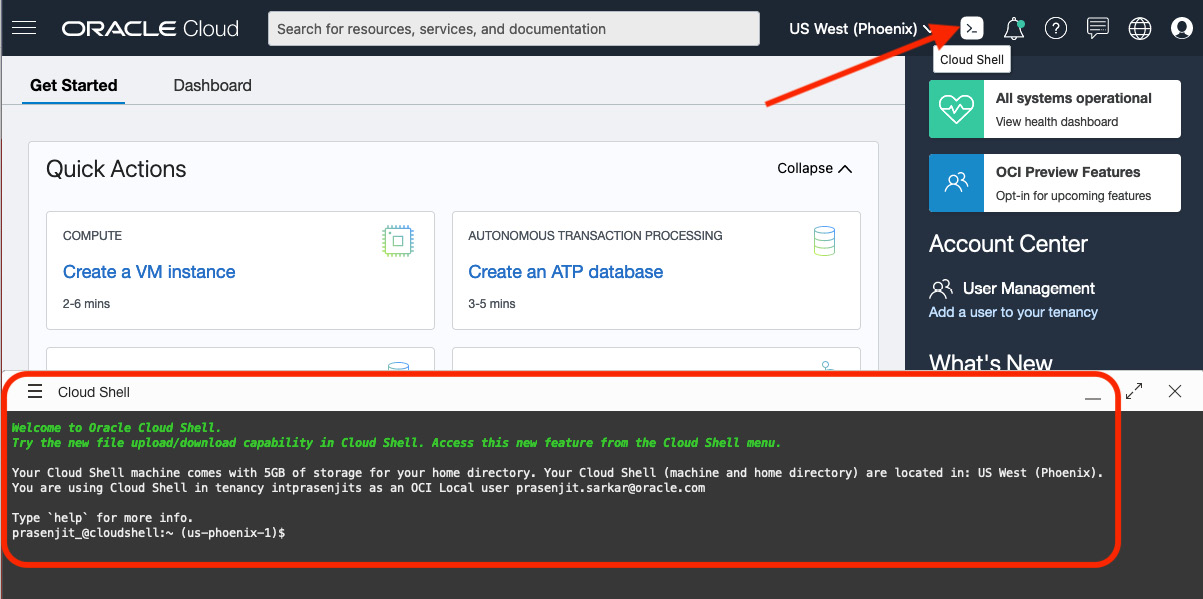


Figure 10.9 – Opening Cloud Shell

1. Let's run the following command to delete the object storage bucket that you have created:

**$ oci os bucket delete --bucket-name testbucket**

**Are you sure you want to delete this resource? [y/N]: y**

So, you see that OCI has a tiny little console that gives us the ability to run important tools, such as the OCI CLI, Terraform, Ansible, Docker, Python, Go, and Java.

In the next section, we will show you how to use the OCI SDKs to do the same job, but by writing code using the SDK.

**Using OCI SDKs to automate OCI operations**

OCI provides several SDKs to develop custom solutions. You can use SDKs for building and deploying applications on OCI, and not only that, but you can also integrate those applications with OCI services as well. You can use the OCI SDK to develop applications for a specific platform, such as Java, Python, Go, Ruby, TypeScript and JavaScript, and .NET.

OCI provides code samples for each language and for each specific API call. So, it is very easy for you to consume these samples, as well as referring to the rich documentation to help you build something for your organization. OCI has open sourced these SDKs, so you can contribute to them as well.

The SDKs available are the following:

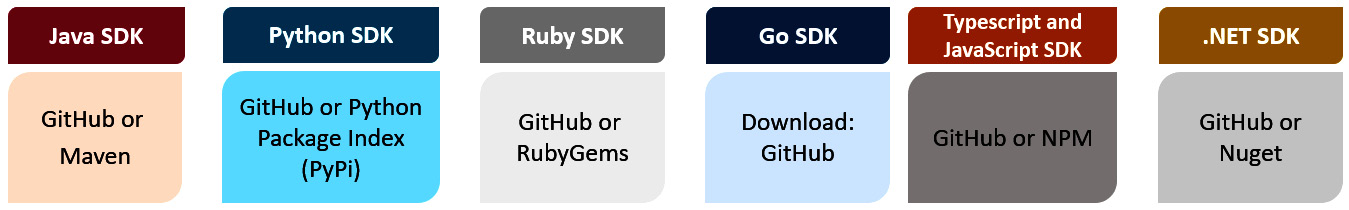


Figure 10.10 – Supported SDKs for OCI

Let's install and configure the Python SDK to list some of the OCI resources:

1. Log in using SSH to the OCI instance that you have created in the *Using the OCI CLI to interact with OCI resources* section. We will use the following topology to run the Python code:

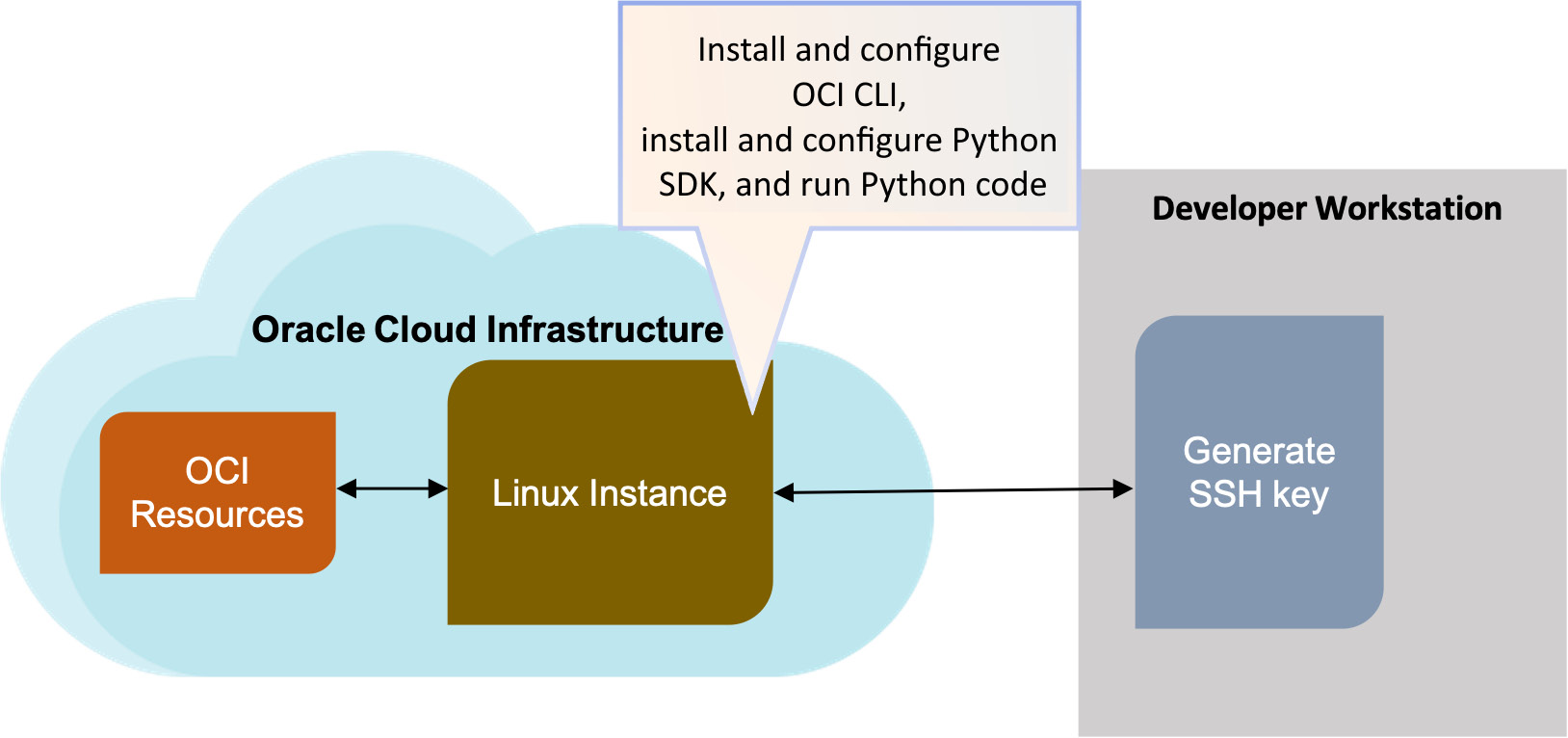


Figure 10.11 – SDK topology

1. Run the following command to install the OCI SDK:

**$ python3.6 -m pip install oci**

You can see a sample output of the command in the following screenshot:

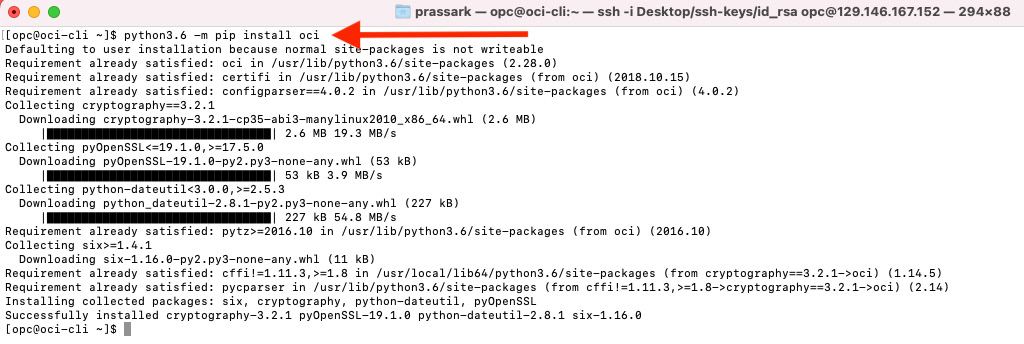


Figure 10.12 – Installing the OCI SDK

1. At this point, you are ready write your first Python code to do any operations that you want using the SDK.
2. Create a file by running **vi user.py**. Paste the following code and save the file by pressing *Esc* and then type **:wq!** and press *Enter*:

import oci

config = oci.config.from\_file()

identity = oci.identity.IdentityClient (config)

user = identity.get\_user(config["user"]).data

print(user)

1. Let's run the program and check the output and then we will explain what it does in each line.
2. Run it by typing **python3 user.py**. You can see a sample output of the code in the following screenshot:



Figure 10.13 – Running Python code using the OCI SDK

So, you can see that the code fetches details about the current user and its details.

Let's explain what the code does in each line:

* The **import** statements are used to import the required packages that are used later in the program:

import oci

* To set up the config, by default OCI CLI refers to the **~/.oci/config** file from the following line of code. This is what you set up while configuring the OCI CLI:

config = oci.config.from\_file()

* When the communication of OCI is set up, the following API code creates a service client to OCI. Refer to <https://oracle-cloud-infrastructure-python-sdk.readthedocs.io/en/latest/api/landing.html> for more OCI APIs:

identity = oci.identity.IdentityClient (config)

* The following line of code uses a service client to get the current user with all its details:

user = identity.get\_user(config["user"]).data

* The following line of code prints the current user details to the terminal after execution of the code:

print(user)

So, you see that OCI provides not only a CLI for interaction, but also lets you use your choice of programming language to automate the management of OCI resources. You can choose between Python, Java, Ruby, Go, .NET, TypeScript, and JavaScript to automate things.

In the next section, we will dive deep into the actual REST API implementation and will see how you can use a simple shell script to send REST API calls to the OCI API endpoint.

**Using the OCI API to send REST calls for managing OCI**

OCI has taken an API-first strategy, which means that before they develop their UIs, they develop the API that will interact with the backend resources. The OCI APIs are typical REST APIs, meaning they use the standard HTTPS requests and responses.

Every OCI service has its own API endpoint. To check the current endpoint related to the specific region you want to send the call to, refer to <https://docs.oracle.com/en-us/iaas/api/>.

OCI maintains its own API versioning as well. If you look carefully at the endpoint, then you will find the desired API version from the base path of the API endpoint. For example, as of now, most of the APIs are versioned as 20160918.

Here's an example for a **GET** request to list users in the Phoenix region:

GET https://identity.us-phoenix1.oraclecloud.com/20160918/users

For tighter security, all OCI API requests must be signed for authentication purposes. For more details on the signature generation for API signing, refer to the following link, as Oracle may change the version of the signature in the future: <https://docs.oracle.com/en-us/iaas/Content/API/Concepts/signingrequests.htm>.

Let's use a bash script to send a REST API call to list the VCNs in a given compartment:

1. Log in using SSH to the OCI instance that you created in the *Using the OCI CLI to interact with OCI resources* section. We will use the following topology to run the Python code.
2. Type **vi vcns.sh** and paste the following code. Change the values of the tenancy ID, user OCID, fingerprint, and compartment OCID:

#!/bin/bash

########################## Fill these in with your values ##########################

#OCID of the tenancy calls are being made in to

tenancy\_ocid="ocid1.tenancy.oc1..<unique-id>"

# OCID of the user making the rest call

user\_ocid="ocid1.user.oc1..<unique-id>"

# path to the private PEM format key for this user

privateKeyPath="/home/opc/.oci/oci\_api\_key.pem"

# fingerprint of the private key for this user

fingerprint="1d:ff:……"

# The REST api you want to call, with any required paramters.

rest\_api="/20160918/vcns?compartmentId=ocid1.compartment.oc1..<unique-id>"

# The host you want to make the call against

host="iaas.us-phoenix-1.oraclecloud.com"

####################################################################################

date=`date -u "+%a, %d %h %Y %H:%M:%S GMT"`

date\_header="date: $date"

host\_header="host: $host"

request\_target="(request-target): get $rest\_api"

# note the order of items. The order in the signing\_string matches the order in the headers

signing\_string="$request\_target\n$date\_header\n$host\_header"

headers="(request-target) date host"

echo "====================================================================================================="

printf '%b' "signing string is $signing\_string \n"

signature=`printf '%b' "$signing\_string" | openssl dgst -sha256 -sign $privateKeyPath | openssl enc -e -base64 | tr -d '\n'`

printf '%b' "Signed Request is  \n$signature\n"

echo "====================================================================================================="

set -x

curl -v -X GET -sS https://$host$rest\_api -H "date: $date" -H "Authorization: Signature version=\"1\",keyId=\"$tenancy\_ocid/$user\_ocid/$fingerprint\",algorithm=\"rsa-sha256\",headers=\"$headers\",signature=\"$signature\""

1. Save the file by pressing *Esc*, then type **wq!** and press *Enter*.
2. Change the permission of this file and provide execute permission by typing **chmod +x vcns.sh** and press *Enter*.
3. Run the script and observe the output. You will see output like that shown in the following screenshot, which shows the standard **200 OK** HTTP response and then the details of the VCNs in the provided compartment:



Figure 10.14 – Running a shell script to list the VCNs using the OCI REST API

Let's explain what this script does on each line:

1. First of all, the first four lines are standard, and these variables will hold your tenancy-specific information. So, provide your tenancy OCID, user OCID, private key path, and fingerprint.
2. This line holds the API version and path-specific information. In this case, it holds the path to list the VCNs in a given compartment. You need to change the compartment OCID to your specific details:

**rest\_api="/20160918/vcns?compartmentId=ocid1.compartment.oc1..<unique-id>"**

1. This line holds the API endpoint. Every service has its own API endpoint. To find out your specific API endpoint, refer to <https://docs.oracle.com/en-us/iaas/api/>:

**host="iaas.us-phoenix-1.oraclecloud.com"**

1. The following code section defines the headers that you need to create the signing string. You need to define these headers in an appropriate format that will be used in conjunction with your signing string. Headers are dependent on your request type (that is, **GET** or **PUT**).

For a **GET** request, you need the request target, host, and date:

**date=`date -u "+%a, %d %h %Y %H:%M:%S GMT"`**

**date\_header="date: $date"**

**host\_header="host: $host"**

**request\_target="(request-target): get $rest\_api"**

**# Note the order of items. The order in the signing\_string matches the order in the headers.**

**signing\_string="$request\_target\n$date\_header\n$host\_header"**

**headers="(request-target) date host"**

1. The next code block is the signature generation, created from your signing string:

**echo "====================================================================================================="**

**printf '%b' "signing string is $signing\_string \n"**

**signature=`printf '%b' "$signing\_string" | openssl dgst -sha256 -sign $privateKeyPath | openssl enc -e -base64 | tr -d '\n'`**

**printf '%b' "Signed Request is  \n$signature\n"**

1. The last section is the actual use of the signature and all of the other details that you set and send to the API endpoint to list the VCNs:

**curl -v -X GET -sS https://$host$rest\_api -H "date: $date" -H "Authorization: Signature version=\"1\",keyId=\"$tenancy\_ocid/$user\_ocid/$fingerprint\",algorithm=\"rsa-sha256\",headers=\"$headers\",signature=\"$signature\""**

Let's break it down piece by piece:

1. The first part is as follows:

**curl -v -X GET -sS**

The **-v** option is to generate verbose output. **-X GET** is to request a **GET** operation. The **-sS** is for silent mode, but still shows errors.

1. Let's look at the second part, as follows:

**https://$host$rest\_api**

The first section is the endpoint that you stored in the variable, and the second part is the API version stored in the variable as well.

1. For all types of requests, that is, for **GET**, **DELETE**, **PUT**, or **POST**, you need the date header. You need this date in the correct HTTP header format. You have already computed this:

**-H "date: $date"**

1. The entire header is then passed in the **Authorization** header. You can see that as follows:

**-H "Authorization: Signature version=\"1\"**

If you look closely, you will notice that the first header field is **Signature version**. The OCI documentation says it is version **1** at the time of writing.

1. Next, you need to pass **keyId**, which we have taken as input into the variable. **keyId** is required for all request types. The syntax is **keyId="<TENANCY-OCID>/<USER-OCID>/<KEY-FINGERPRINT>"**:

**keyId=\"$tenancy\_ocid/$user\_ocid/$fingerprint\"**

1. Next, you need to pass the signing algorithm, which must be **rsa-sha256**:

**algorithm=\"rsa-sha256\"**

1. Lastly, put down the header and the signature that you generated earlier in the code:

**headers=\"$headers\",signature=\"$signature\""**

So, you can see how you can use a simple bash script to send OCI REST API requests to manage OCI resources.

**Summary**

In this chapter, you learned the various way of managing resources in OCI other than using the Console. OCI's API-first strategy is excellent in that all the other developer tools can utilize its benefits and produce various other ways to interact with resources such as the OCI CLI, Cloud Shell, SDKs, and REST API. You learned how to use the OCI CLI to create and delete resources not only from your local workstation, but also using Cloud Shell. The various programming languages on offer give you the ability to choose your preferred option and automate operations using code. Finally, the REST API offers an intuitive, easy, robust and at the same time secure way of interacting with the API endpoint.

In the next chapter, you will see how you can use a VMware solution and craft a hybrid cloud solution.