***Chapter 9*: Managing Infrastructure as Code on Oracle Cloud Infrastructure**

**Infrastructure as code** (**IaC**) is a term used to reference the ability to turn a script or a template into functioning infrastructure resources. Some tools such as ad hoc scripts require you to specify all necessary commands in an appropriate sequence in order to create the resources for those scripts. In this scenario, the user would be responsible for managing dependencies and the sequence of all command executions. In addition, they would need to create a similar script to remove all the resources.

Configuration management tools provide both IaC and **configuration as code** (**CaC**) capabilities. This often allows for a somewhat limited set of infrastructure provisioning capabilities, while yielding significant configuration management capabilities. The concept of idempotency is also important, as tools such as Chef and Puppet will enforce the defined configuration, reverting potentially unwanted manual changes.

Docker, Packer, and Vagrant are all server templating tools. These are gaining popularity as an alternative to Bootstrap and Config tools, and work around snapshotting and delivering templated systems. These tools help in creating immutable infrastructure. Immutable infrastructure states that once you deploy a server, you don't change it. Changes should be via destroy and create operations rather than reconfiguring. Netflix has done a lot around this with **continuous integration** (**CI**), Bakery, Aminator, and Spinnaker. Everything Netflix deploys goes through CI, gets baked into an **Amazon Machine Image** (**AMI**) cloud image, and goes through a test loop. Any changes require a destroy operation and redeployment of the infrastructure. No time is spent curating an image once it has been deployed.

Infrastructure and provisioning tools' or orchestrators' jobs are not only to create servers, but almost anything on a cloud infrastructure, such as databases, networking configurations, and load balancers. Besides Terraform, CloudFormation and OpenStack Heat are typical examples of infrastructure automation tools. These tools can provision servers and do some configuration management, but are mainly there to provision resources that a given infrastructure provider provides.

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

* Understanding the need for IaC
* Understanding the use cases of **Oracle Resource Manager** (**ORM**)
* Learning to generate IaC from an existing setup
* Learning to integrate ORM with **source code management** (**SCM**)

IaC has taken center stage in the DevOps realm. Using IaC, you treat your infrastructure the same way as you would your application code. As an example, you will check your IaC code into version control, you will write tests for it, and then you'll make sure that it doesn't diverge from what you have across multiple environments.

By the end of this chapter, you will get to know not only the basics of IaC and Terraform, but also the Managed Service offering of Terraform on **Oracle Cloud Infrastructure** (**OCI**) and how you can integrate it with a source code repository, generate code from an existing deployed infrastructure, and so on.

**Understanding the need for IaC**

IaC automates infrastructure deployment and updates with software. It enables agile development and DevOps. Terraform is an open source engine that processes IaC written in **HashiCorp Configuration Language** (**HCL**).

So, let's take a look at why you need IaC. Developers or DevOps engineers use a fast and reusable process to deploy and update an infrastructure. Thus, it is clear that infrastructure should be automatically provisioned and managed from code, not manually. That's why you use IaC.

In IaC, you define the end state of an infrastructure and let tools manage it for you. IaC is literally a self-documenting infrastructure, which is consistent and achieves repeatable results. It increases efficiency while reducing risk. Here, you can see a high-level diagram of how you can write code to define the end state of an infrastructure and let a tool such as Terraform handle that for you:

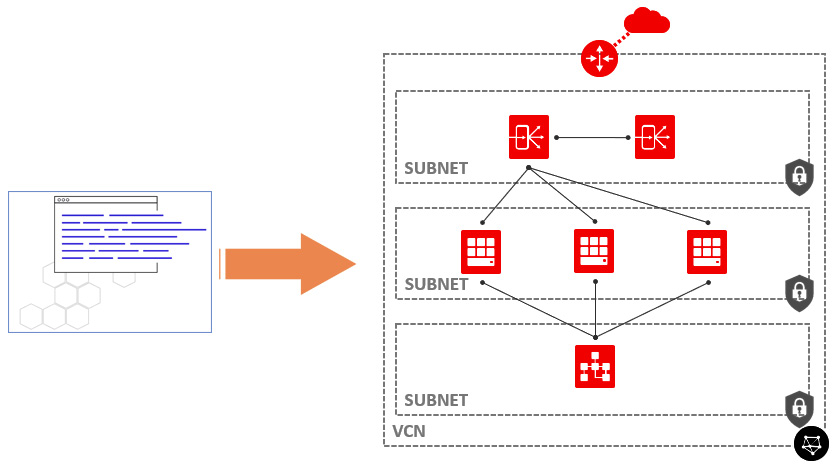


Figure 9.1 – Use case of IaC

Terraform is the front runner in the DevOps toolchain. It can build and change an infrastructure very efficiently and manage this infrastructure state as code using versions as well. Terraform has a rich built-in provider support that you can use to deploy infrastructure on a variety of platforms. Terraform has become a part of the cloud orchestration tools that handle the infrastructure and application life cycle.

IaC's goal is to create and manage cloud infrastructure and deployments predictably and repeatedly. It makes use of templates and automation for just about everything. You can see the benefits of using IaC in the following diagram:

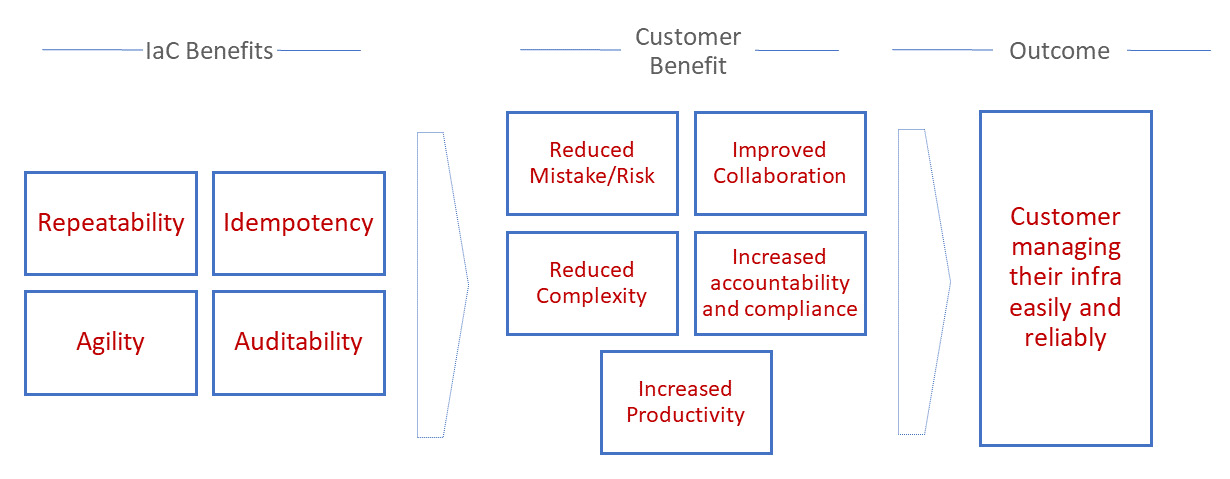


Figure 9.2 – Benefits of IaC

As part of your automation tools, Terraform can be integrated with configuration management tools such as Chef, Puppet, and Ansible.

Let's look at couple of tools from HashiCorp, along with their purpose. HashiCorp builds and maintains a variety of cloud operations tools, and their goal is *"Any application on any infrastructure."* This means that using these tools, you can define and deploy any application on any cloud.

For provisioning, they built **Vagrant**, which is a popular tool for automating development environments whereby you can define the environments as a template. By way of an example, install **Linux, Apache, MySQL, PHP/Perl/Python** (**LAMP**) and dependencies, set permissions, enable logging, create a database, create users and permissions, and then fire up something to work on—for example, WordPress—and do all of this with a **vagrant up** command.

**Packer** is a very useful tool for custom creation of operating system images. One way is to use configuration and deployment tools such as Chef, Puppet, Ansible, and SaltStack, and another way is to continually bake ready-to-go images that require minimal or no Bootstrap configuration.

For security, HashiCorp built **Vault**. This is capable of storing keys, certificates, passwords, and other sensitive information. Vault is a **Go** application with a **REpresentational state transfer command-line interface** (**REST CLI**) that you can use to store secrets.

For running distributed containers, they created **Nomad**, which is a cluster management system for distributed, highly available, data center-aware container schedulers. Nomad *only provides* cluster management and scheduling.

While other orchestration tools provide much more than just cluster management and scheduling, Nomad keeps it simple with a single binary for clients and servers. It is often compared to Kubernetes, but only focuses on scheduling (jobs and tasks).

For the storing of key values, HashiCorp created **Consul**. Consul's mission statement is to automatically know and control what applications and infrastructure are doing at any given time and detect failures, deal with new servers, and update configurations, database names, bucket names, and so on.

So, we have learned about the use cases and benefits of using IaC. Let's learn about the use cases of Oracle's managed IaC service.

**Understanding the use cases of ORM**

Oracle provides a managed service of Terraform called **ORM**, which manages infrastructure using HashiCorp Terraform. It uses templates to define configurations, and you can reuse those templates as needed. It's a free service and you need to pay only for infrastructure, not for service. ORM is deeply integrated with OCI services, such as identity, security, metering, monitoring, and tagging.

Oracle built this ORM service on unmodified open source software, meaning there is no lock-in and you get simple migrations from or to any private and third-party clouds. Here, you can see a high-level diagram of the ORM stack. This workflow depicts how you can use ORM to deploy a web application to different environments on OCI:

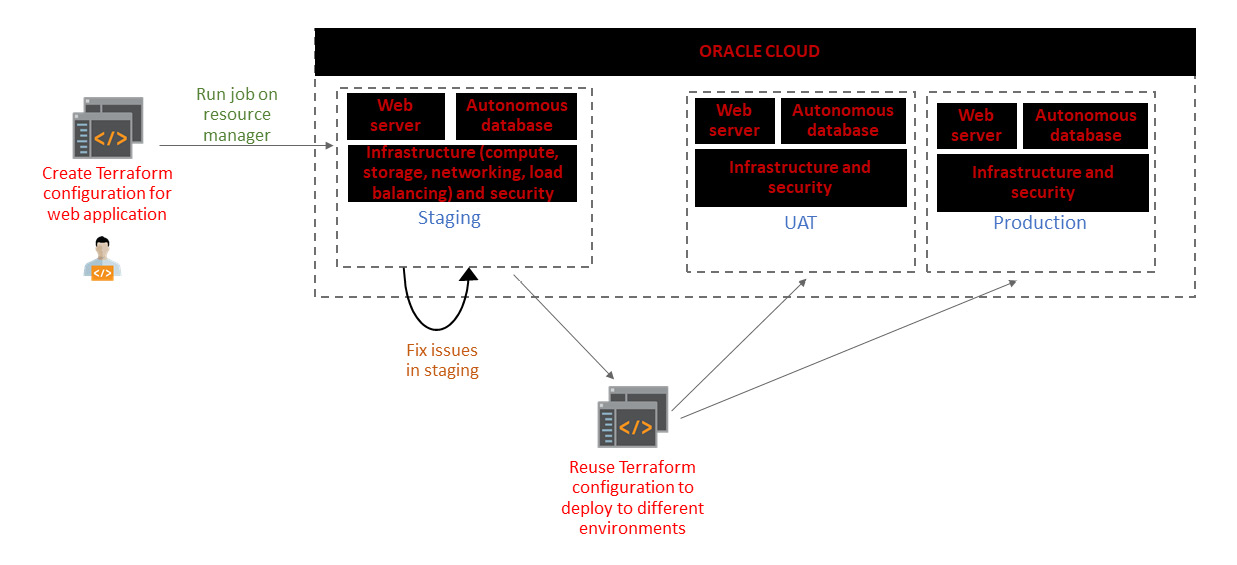


Figure 9.3 – Use case of IaC

Here are the benefits of using ORM:

* As with Terraform, you can use ORM to unlock infrastructure automation capabilities that can help you to standardize your infrastructure, and you can easily replicate your environments into multiple replicas.
* ORM is deeply integrated with the OCI platform, and as a developer, you can leverage the OCI **application programming interface** (**API**) catalog.
* Built on top of Terraform's principal, you can use ORM to share and manage infrastructure configurations as well as Terraform state files across multiple teams and platforms, improving collaboration.
* You don't need to install the Terraform executable on your local developer workstation to invoke the infrastructure creation. You can use the ORM **user interface** (**UI**) to just invoke the code that you have written, or you can use it to push the code to GitHub and GitLab as well. Both of them are integrated with the ORM stack.
* ORM is a free service whereby you only pay for the resources that you create and consume.

So, you have learned about the various different use cases of ORM, why ORM is important, and where you can apply it. Let's now look at the components that make up this service.

**ORM components**

ORM is nothing but Terraform-as-a-service for OCI resources. You need to write your own code that will be used to create an OCI infrastructure. This is called a **Terraform file**. Then, you need to adjust the variables, and that's it. You can use ORM to start building stacks and executing jobs.

In ORM, a stack represents a set of OCI resources you want to create in a compartment. Each stack holds the Terraform configuration files, which are **.tf** files where you specify the resources you want to create using ORM.

Once you upload the configuration files, you run a job to create a stack. In ORM, a job is an action on a stack. ORM provides three jobs, and they are **plan**, **apply**, and **destroy**.

You can see a high-level diagram of life cycle operations on the ORM stack in the following diagram:

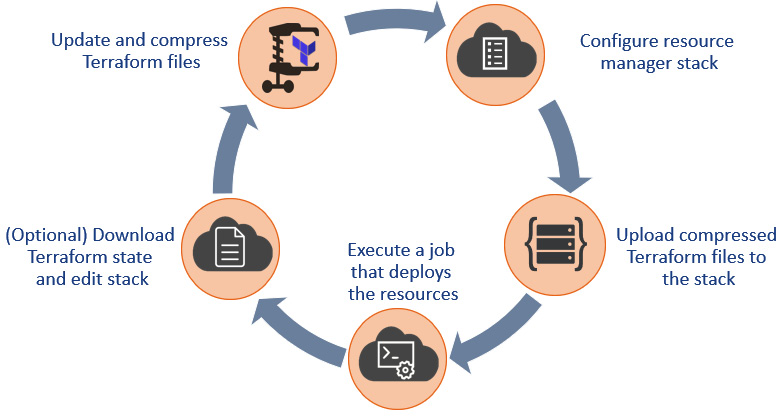


Figure 9.4 – Cycle of ORM components

A plan job, when run on an uploaded stack, parses the Terraform configuration file and creates an execution plan. You can check the logs to see which resources are being added or destroyed.

If you feel that the intended job is just the way you have designed it to be, then you need to apply the job. Once you perform an apply job, then only the stack gets fully provisioned.

Essential information about the state of your resource's configuration is maintained in a state file, which is a **JavaScript Object Notation** (**JSON**) file. It is ORM's job to create this state file and maintain it. It's stored internally in the OCI object store, and you can download this file.

ORM supports state locking on a stack when you apply a job. This is done by only allowing one job at a time to run on a given stack. You can see the stages of ORM jobs in the following diagram:

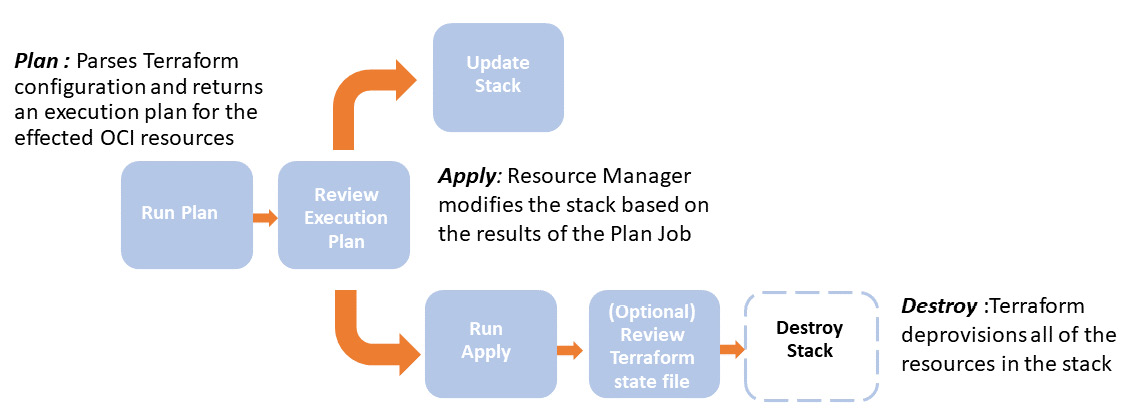


Figure 9.5 – ORM workflow stages

Let's create a sample ORM stack and see how easily you can use ORM to deploy your infrastructure. For this task, you need to have a **.zip** file of the Terraform code. We have supplied sample code that will create an Oracle container engine for a Kubernetes cluster on OCI using ORM. You need to download the sample code from this link: <https://github.com/stretchcloud/oke-terraform-ci-cd/blob/master/OKE-TF.zip>. Then, follow these next steps:

1. Sign in to the OCI console.
2. Open the navigation menu, select **Resource Manager**, and then **Stacks**.
3. Click on **Create Stack**.
4. Select **My Configuration (default)**, and then select **.Zip file**.
5. Upload the **.zip** file that you downloaded from the link provided earlier. You can see a sample screenshot here:

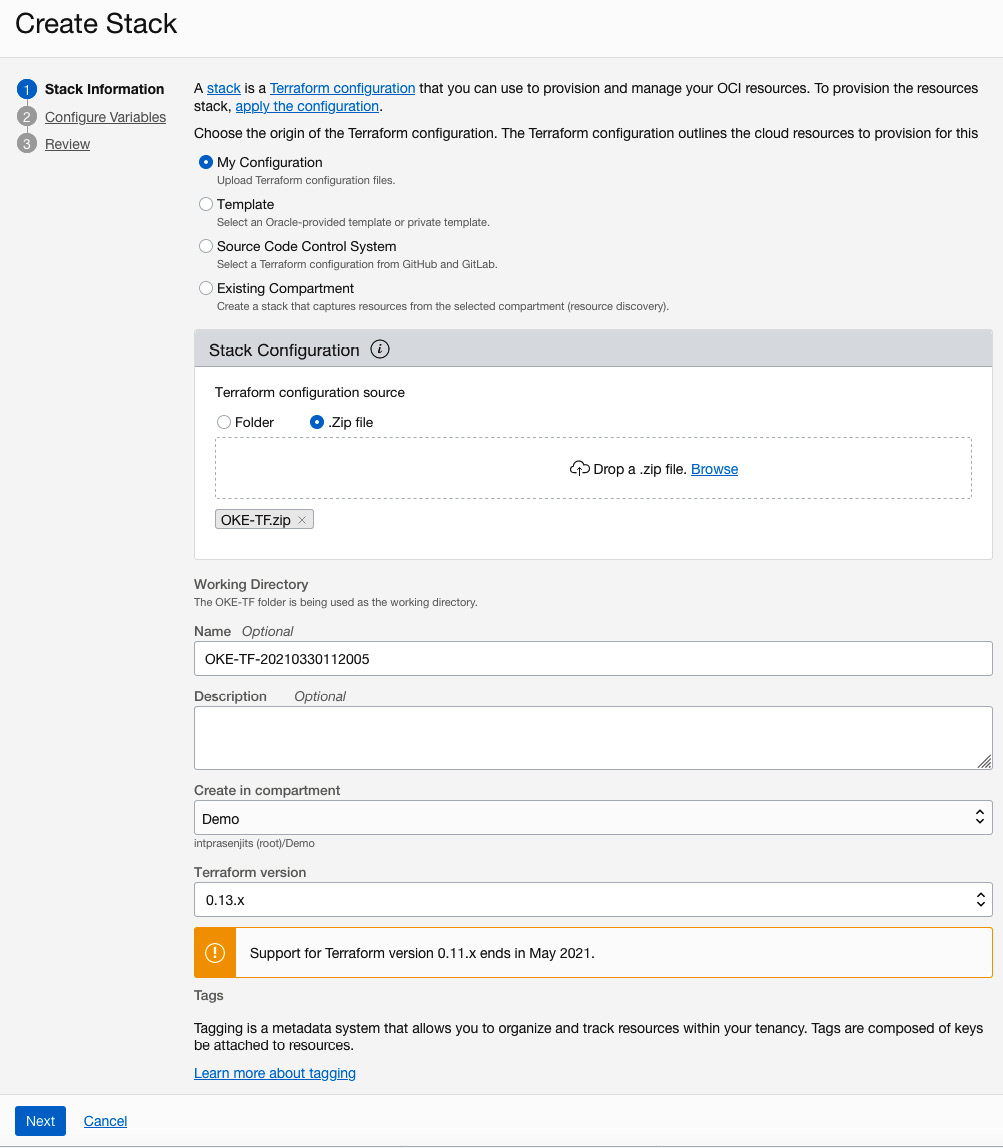


Figure 9.6 – Creating an ORM stack

1. Click on **Next**.
2. In the **Configure Variables** page, it will automatically pop in the **Oracle Cloud ID** (**OCID**) compartment and region based on your selection.
3. Provide a node pool **Secure Shell** (**SSH**) public key and click on **Next**. You can see a sample screenshot here:

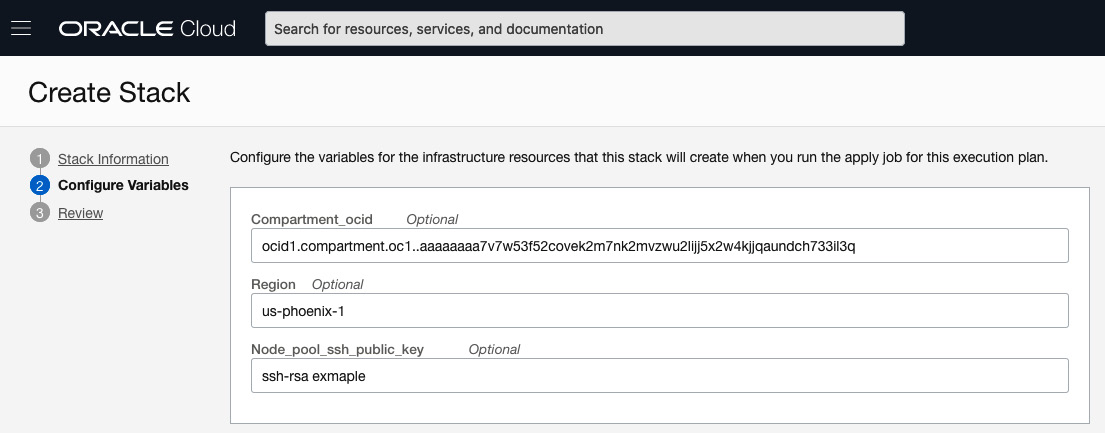


Figure 9.7 – Configuring variables for an ORM stack

1. Click on **Create**.

Once a stack is created, you need to run plan and apply jobs on it. Let's do that now, as follows:

1. From the **Stack Details** screen, select **Jobs** in the **Resources** section.
2. Click on **Terraform Actions** and then select **Plan**. You can see a sample screenshot here:

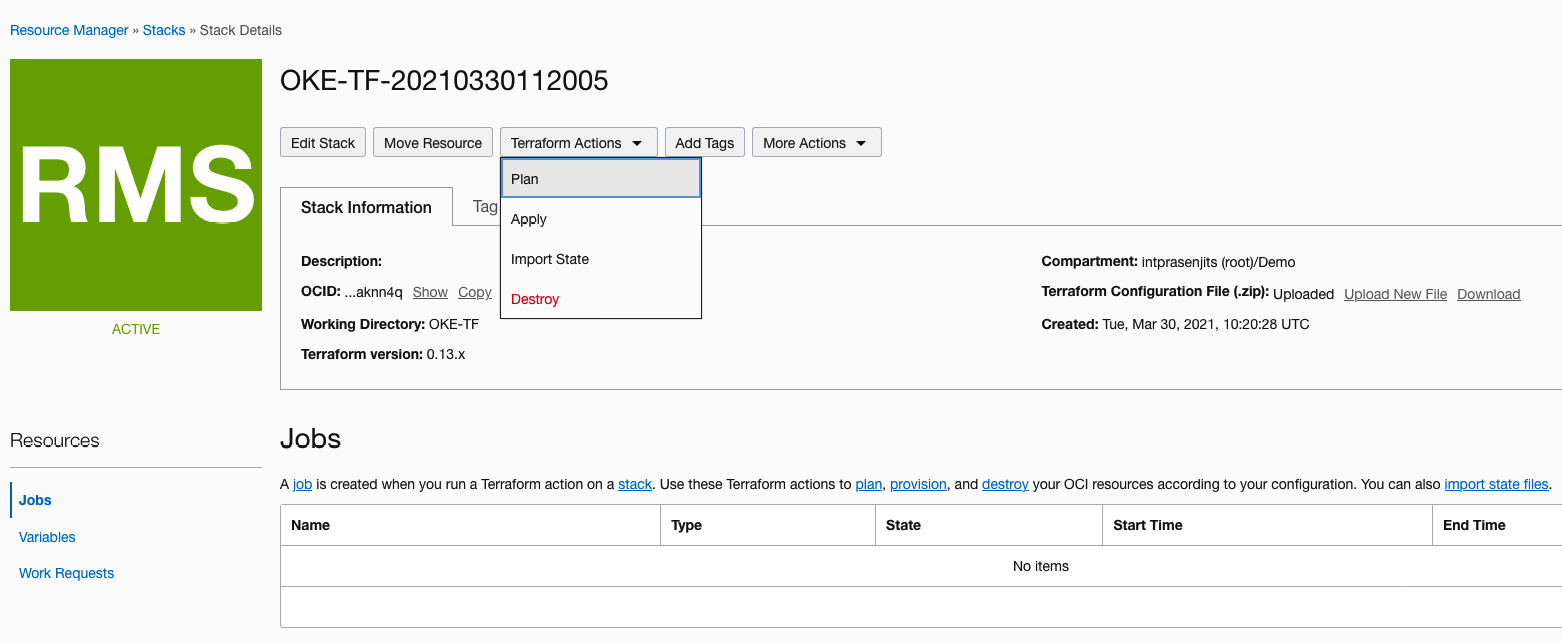


Figure 9.8 – Plan job for an ORM stack

1. Provide a name for the plan job (optional) and click on **Plan**.
2. Once the plan job is finished, you will see the output of it and will be able to verify how many resources are going to be created. You can see a sample screenshot here:

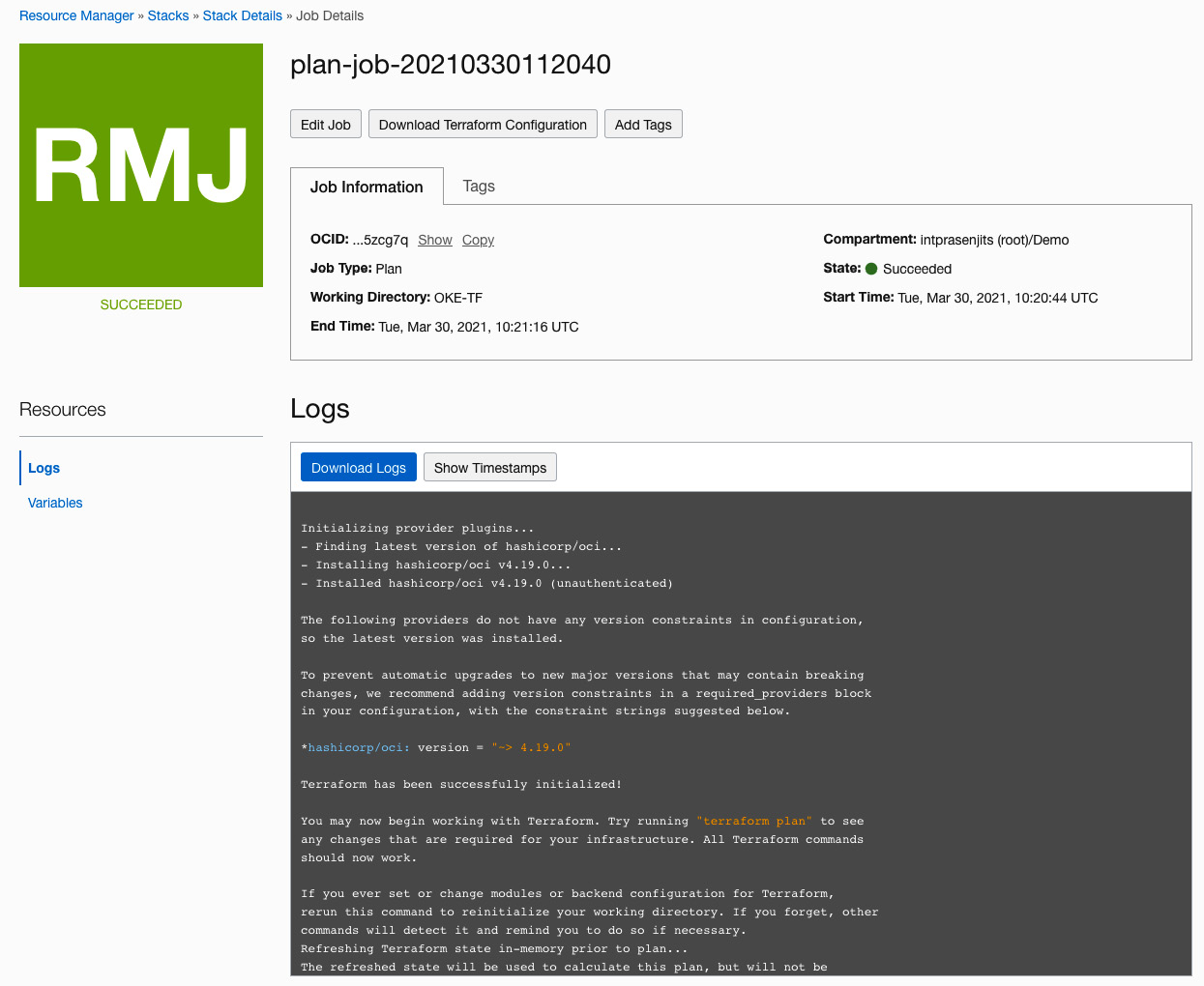


Figure 9.9 – Plan job output for an ORM stack

Once the plan job is finished running, you need to run an apply job to finally create the infrastructure. Let's do that now, as follows:

1. From the **Stack Details** screen, select **Jobs** in the **Resources** section.
2. Click on **Terraform Actions** and select **Apply**, which will take you to the following screen:

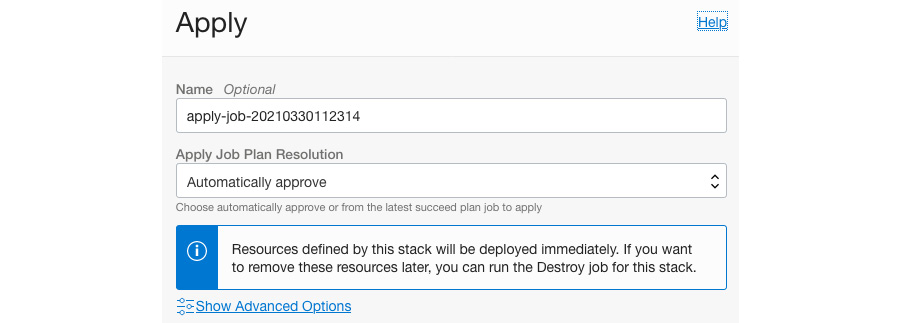


Figure 9.10 – Apply job for an ORM stack

1. Choose the default option and click on **Apply**.
2. Once the apply job is finished, you will see the output of it. You can see a sample screenshot here:

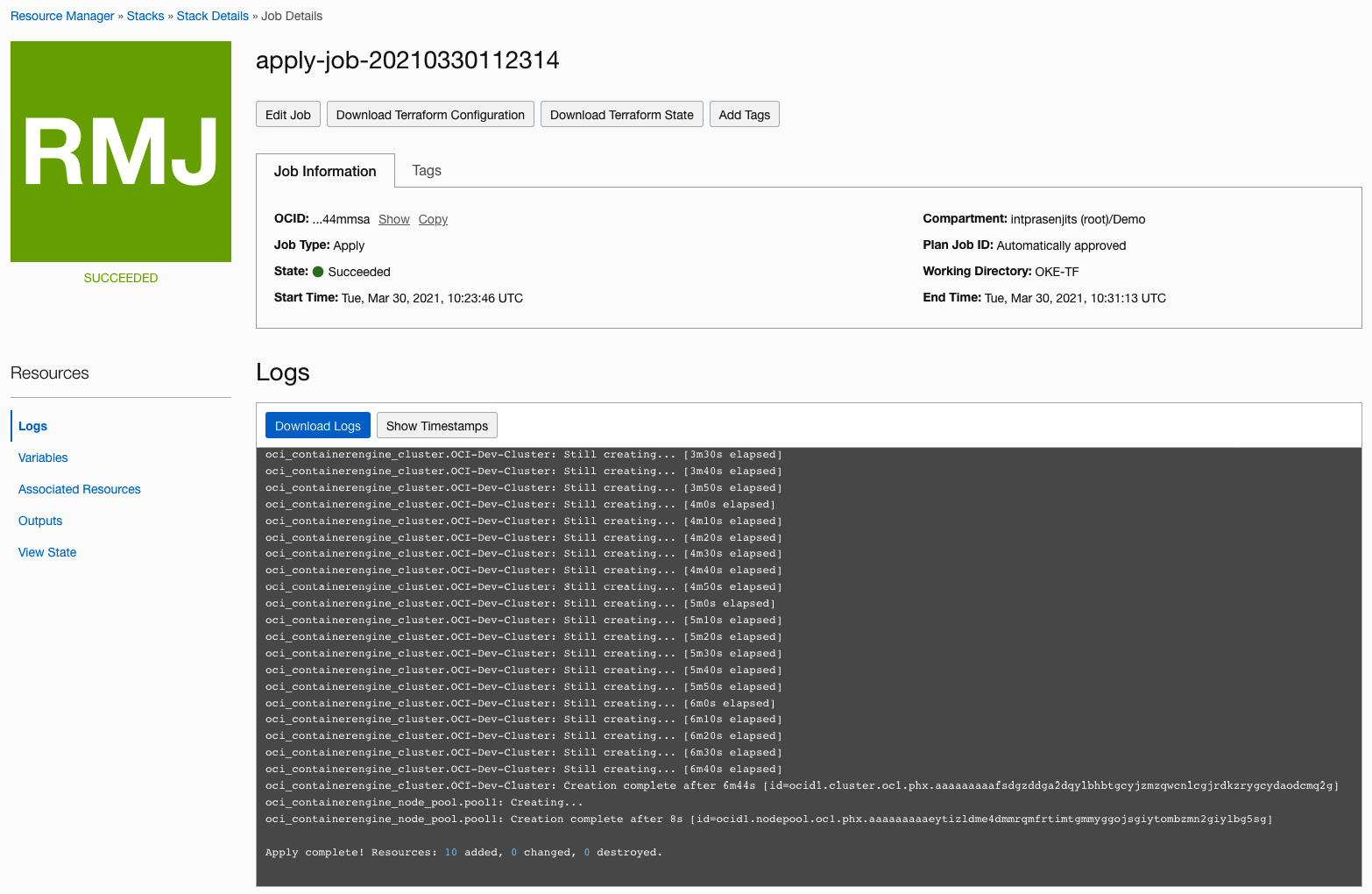


Figure 9.11 – Apply job output for an ORM stack

So, you can see how easy it is to create an infrastructure using ORM on OCI. At this point, you can check the associated resources and also download the Terraform state if you want to.

1. From the **Apply** job screen, select **Associated Resources** in the **Resources** section.
2. Here, you will see the resources that have been created. You can see a sample screenshot here:

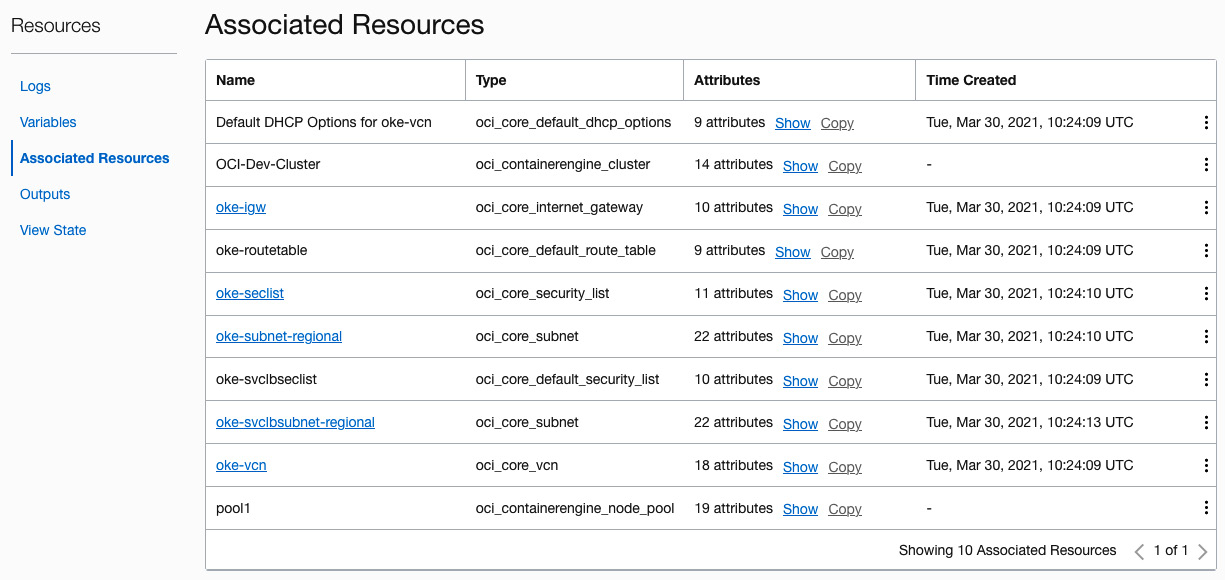


Figure 9.12 – Associated resources created by an ORM stack

If you want to destroy the infrastructure that you have just created, run a destroy job now.

1. From the **Stack Details** screen, select **Jobs** in the **Resources** section.
2. Click on **Terraform Actions** and select **Destroy**.
3. Provide a name (optional) and select **Destroy**. You can see a sample screenshot here:

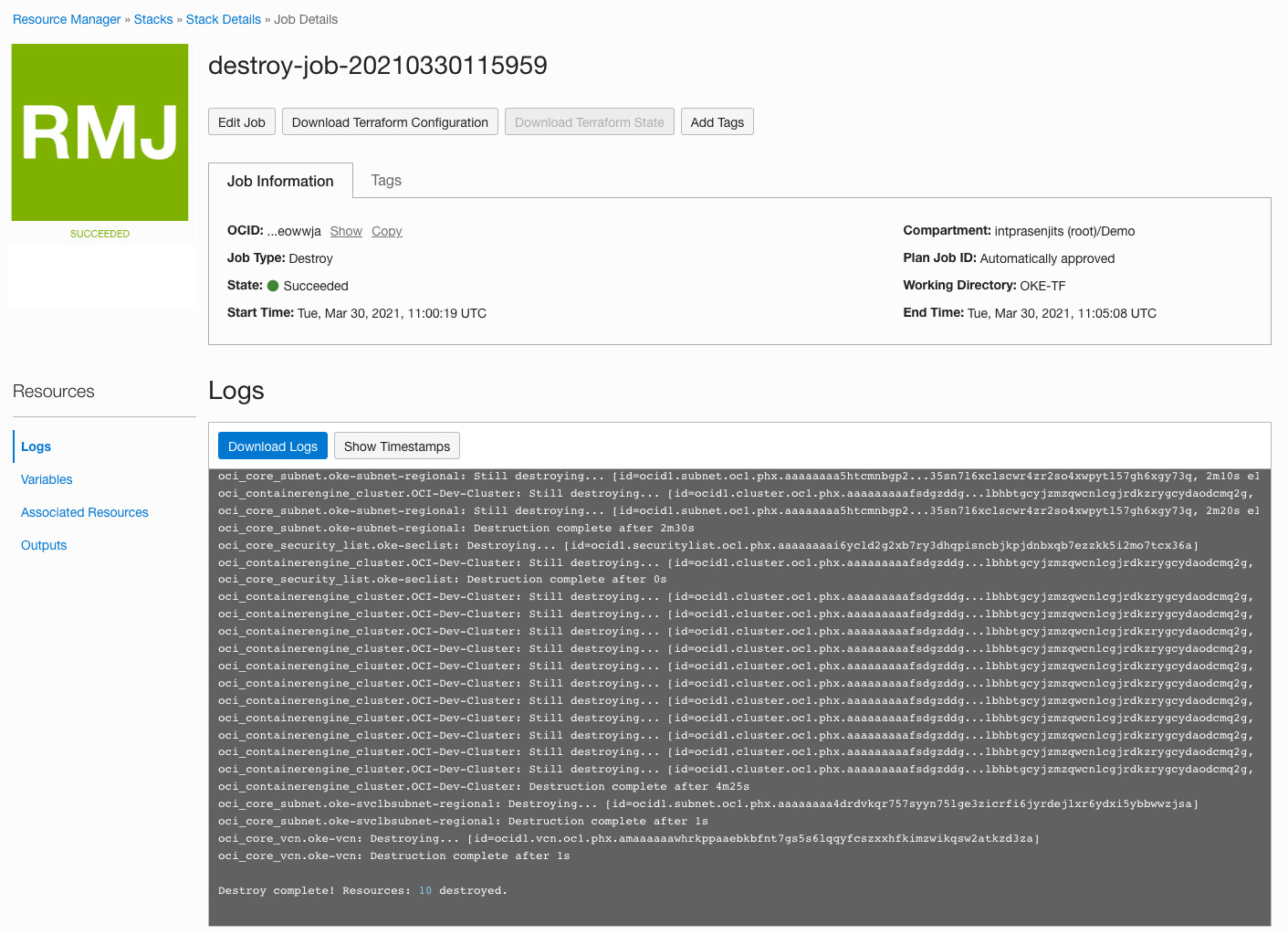


Figure 9.13 – Destroying an ORM stack

Up to now, you have learned how to create an ORM stack and apply jobs to create your infrastructure on OCI. You have also learned how to destroy resources. In the next section, we will discuss the benefits of using ORM to create sample Terraform code from an existing compartment and its associated infrastructure.

**Learning to generate IaC from an existing setup**

Terraform **Resource Discovery** has been created to help you to discover deployed resources within a compartment and then export all of them to Terraform configuration and state files.

The goal of this feature is to generate Terraform configurations for an OCI customer's compartment. The idea is that a customer infrastructure that has been created in one compartment (mainly via the console) can be replicated in other compartments and/or tenancies.

The generated resource configurations should be able to duplicate the user infrastructure across compartments, tenancies, and regions.

This will solve the pain points of customers when they start creating resources using a console but, at a later stage, think of moving to code-based infrastructure deployment, and will help them to significantly reduce the learning curve of writing HCL from scratch.

To discover resources using ORM, you need to go through the **Create Stack** workflow, as follows:

1. Sign in to the OCI console.
2. Open the navigation menu, select **Resource Manager**, and then **Stacks**.
3. Click on **Create Stack**.
4. Choose **Existing Compartment**.
5. Under the **Stack Configuration** section, select the compartment from where you want to discover resources.
6. Choose a region where you want to discover the resources from.
7. You can either select **All** from **Terraform Provider Services** or select a particular type of resource by choosing **Selected** and selecting an option from the **Services** dropdown. Examples of these services include **apigateway**, **auto\_scaling**, **availability\_domain**, **containerengine**, core, database, functions, and **load\_balancer**, to name a few. You can see a sample screenshot here:

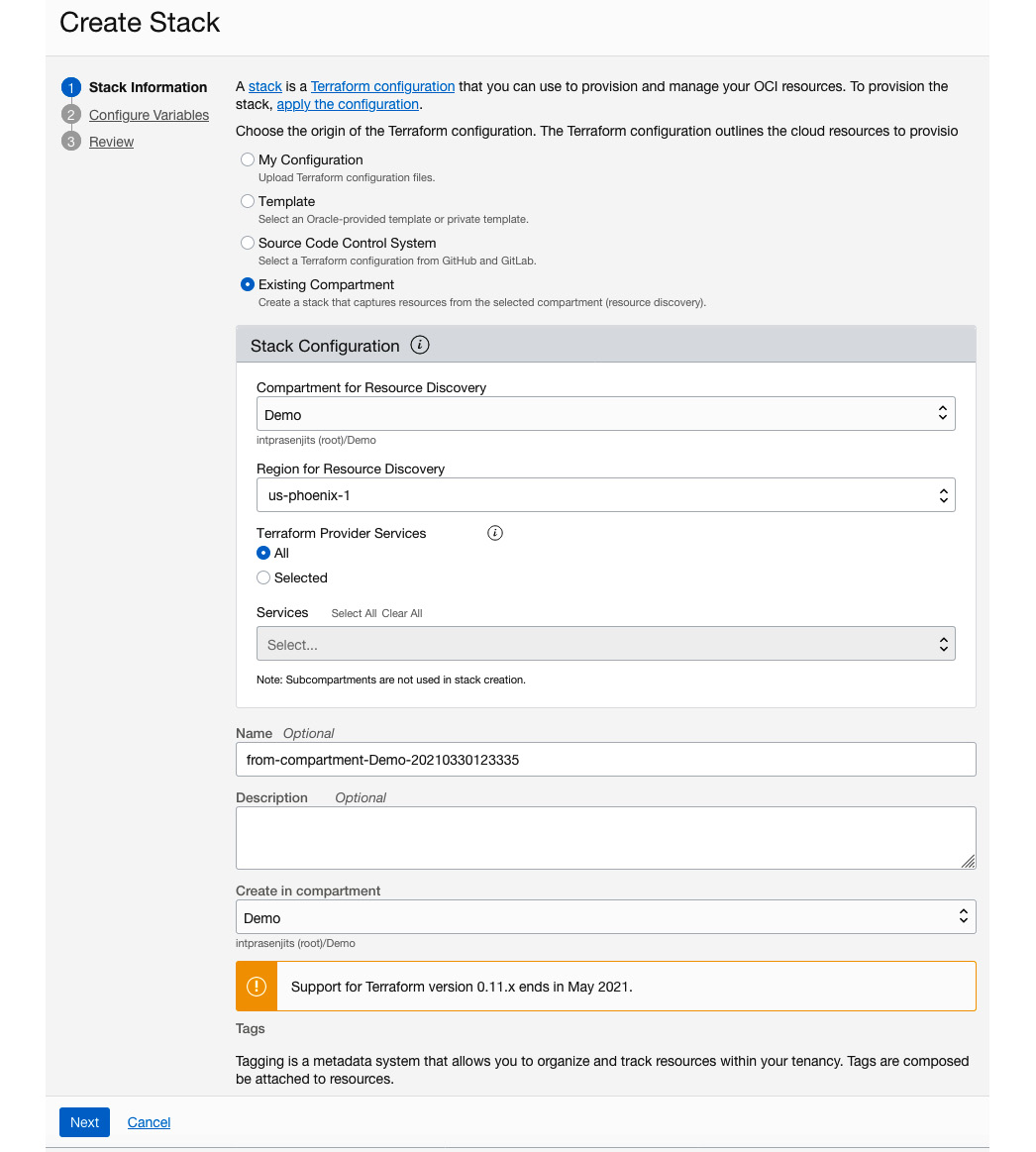


Figure 9.14 – Discovering resources using ORM

1. Click on **Next**.
2. Click on **Next**.
3. Click on **Create**.
4. Once the job is finished, you can download the **Terraform Configuration File (.zip)** from the **Stack Details** page. You can see a sample screenshot here:

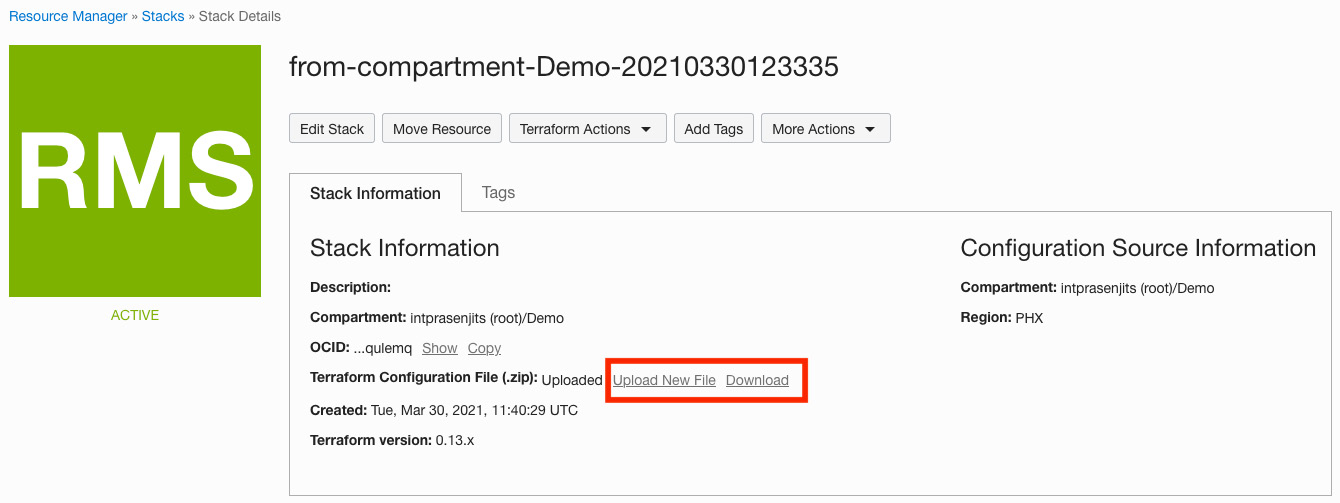


Figure 9.15 – Downloading a Terraform configuration file using Resource Discovery

So, you see how easy it is to adopt IaC by using ORM and its **Resource Discovery** service. In the next section, we will show you how to use an SCM system such as GitHub to integrate the code check-in and discover resources on the ORM stack to create an infrastructure.

**Learning to integrate ORM with SCM**

In a DevOps world, keeping a source Terraform configuration file on a local laptop is highly unlikely. All service teams typically store their Terraform code in an SCM tool such as GitHub or GitLab.

ORM provides a way to integrate two of the most popular SCM providers within ORM so that you can use SCM to store the Terraform configuration files to create an infrastructure. Those providers are GitHub and GitLab.

In this section, we will show you how to configure an SCM provider and then how to use it to create an infrastructure. Follow these next steps:

1. Sign in to the OCI console.
2. Open the navigation menu, select **Resource Manager**, and then **Configuration Source Providers**.
3. Click on **Create Configuration Source Provider**.
4. Provide a name for the provider.
5. Select **GitHub** for the **Type** option.
6. Provide the **Server URL**. As we are using free GitHub user accounts, we are using [https://github.com](https://github.com/).
7. If you do not have a **personal access token** (**PAT**), then go to <https://docs.github.com/en/github/authenticating-to-github/creating-a-personal-access-token> to create a PAT for your GitHub account. You can see a sample screenshot here:

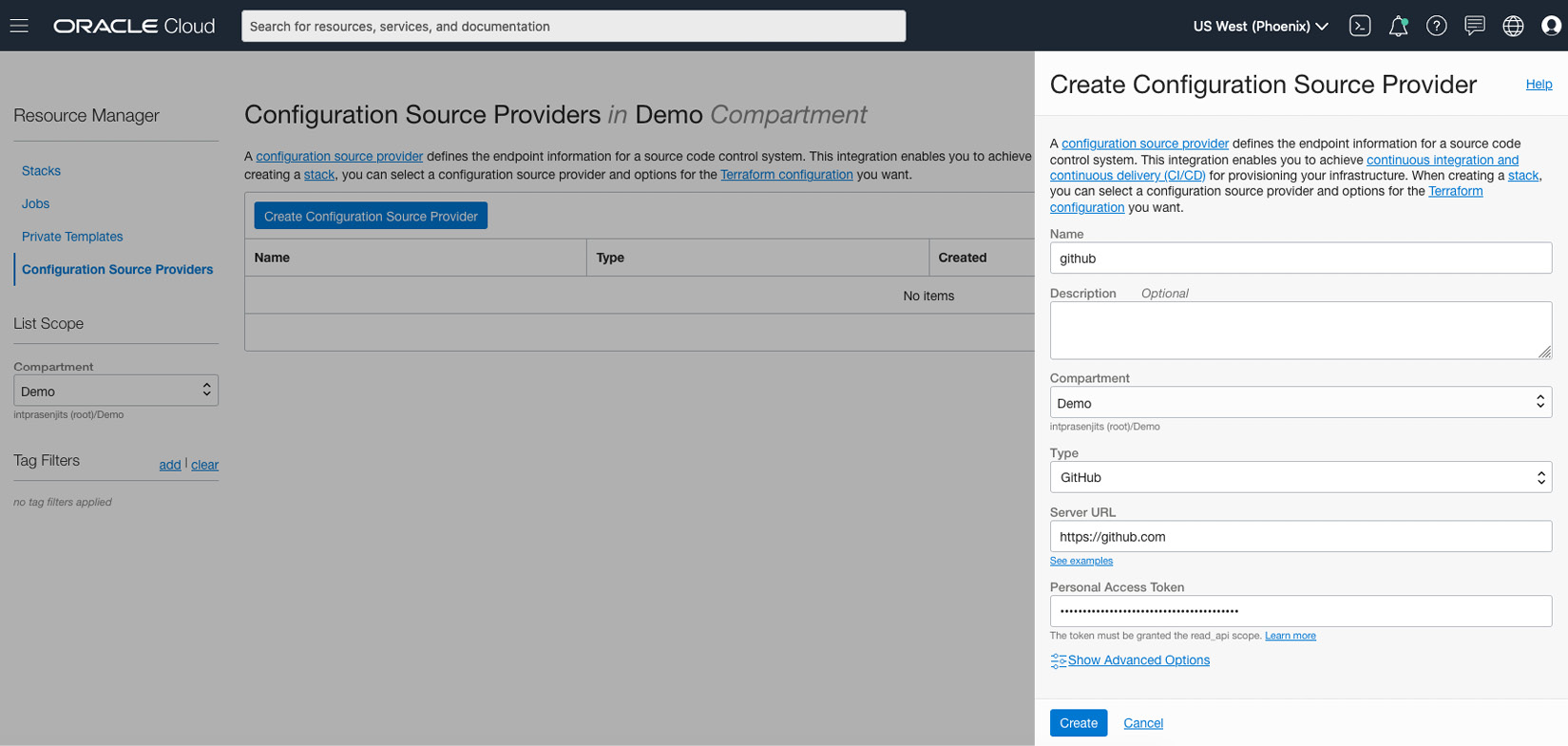


Figure 9.16 – Creating a configuration provider

1. Paste the token into the **Personal Access Token** field and click on **Create**.

Once the configuration provider is created, you can create a stack from the code that is present in any one of the repositories that you created. Let's go through these steps now, as follows:

1. From the **Resource Manager** screen, select **Stacks**.
2. Click on **Create Stack**.
3. Select **Source Code Control System**.
4. Choose an option from the **Configuration Source Provider** dropdown.
5. Select the **Repository** and **Branch** where you kept your code. You can see a sample screenshot here:

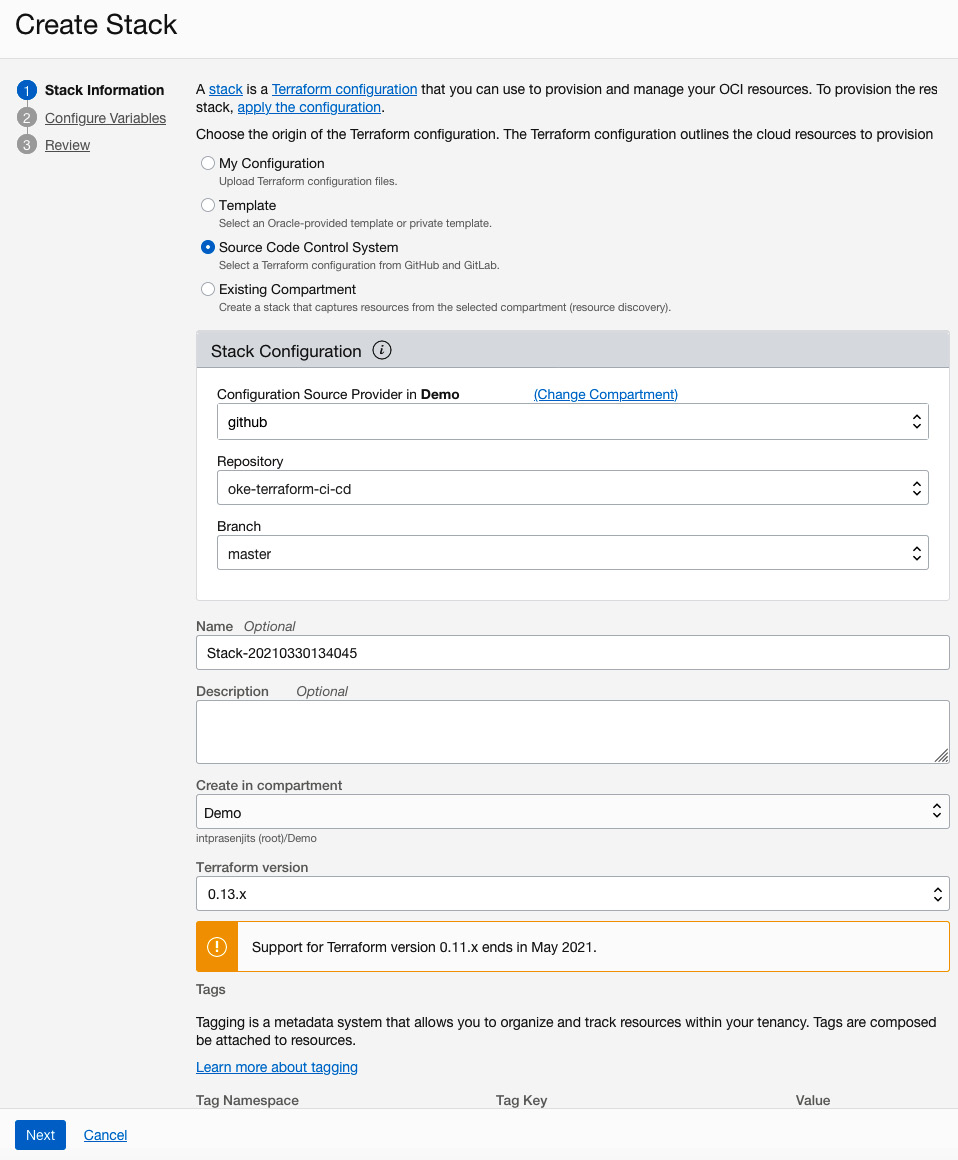


Figure 9.17 – Creating a stack from SCM

1. Click on **Next** and if there is any variable, provide the value of it in this page and click on **Next**.
2. Click on **Create** to create this stack.

After you have created a stack, you will go through the usual steps to create an infrastructure, which involves planning and applying jobs to this stack.

**Summary**

In this chapter, you have learned about the need and the use cases of IaC tools such as Terraform. You have also learned how the managed Terraform service works on OCI by creating ORM stacks and applying jobs on top of them (in other words, the stacks). You have further learned how a customer can quickly get started with **Resource Discovery** and Terraform configuration. Not only that, but you have also learned how you can use development tools such as SCM to check in your code and use that code to spin up infrastructure. This is required for adopting a DevOps culture within the software development life cycle.

In the next chapter, you will see how you can use the OCI CLI, API, and **software development kit** (**SDK**) to automate processes that you have been carrying out so far using the console.