**Automating the Deployment of Infrastructure Using Deployment Manager**

Toggle Lab Panel



**Automating the Deployment of Infrastructure Using Deployment Manager**

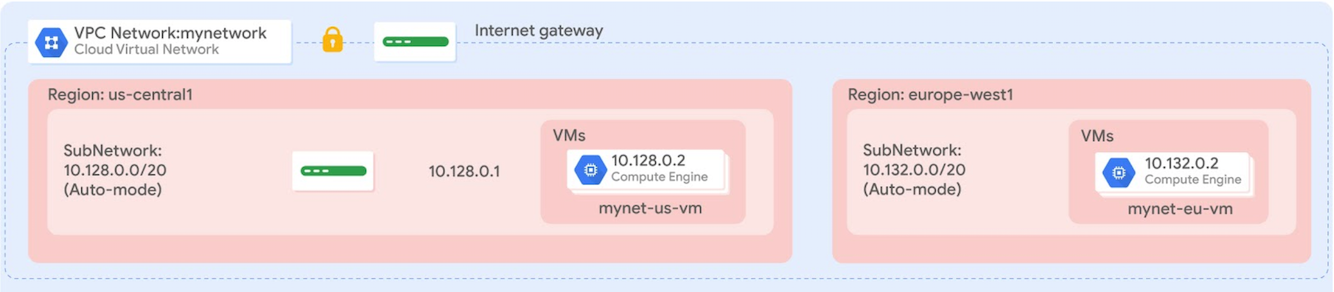
1 hour Free

Rate Lab

**Overview**

Deployment Manager is an infrastructure deployment service that automates the creation and management of Google Cloud resources. Write flexible template and configuration files and use them to create deployments that have a variety of Cloud Platform services, such as Cloud Storage, Compute Engine, and Cloud SQL, configured to work together.

In this lab, you create a Deployment Manager configuration with a template to automate the deployment of Google Cloud infrastructure. Specifically, you deploy one auto mode network with a firewall rule and two VM instances, as shown in this diagram:



**Objectives**

In this lab, you learn how to perform the following tasks:

* Create a configuration for an auto mode network
* Create a configuration for a firewall rule
* Create a template for VM instances
* Create and deploy a configuration
* Verify the deployment of a configuration

**Qwiklabs setup**

For each lab, you get a new GCP project and set of resources for a fixed time at no cost.

1. Make sure you signed into Qwiklabs using an **incognito window**.
2. Note the lab's access time (for example, img/time.pngand make sure you can finish in that time block.

There is no pause feature. You can restart if needed, but you have to start at the beginning.

1. When ready, click img/start_lab.png.
2. Note your lab credentials. You will use them to sign in to Cloud Platform Console. 
3. Click **Open Google Console**.
4. Click **Use another account** and copy/paste credentials for **this** lab into the prompts.

If you use other credentials, you'll get errors or **incur charges**.

1. Accept the terms and skip the recovery resource page.

Do not click **End Lab** unless you are finished with the lab or want to restart it. This clears your work and removes the project.

**Task 1. Configure the network**

A configuration describes all the resources you want for a single deployment.

**Verify that the Deployment Manager API is enabled**

1. In the Cloud Console, on the **Navigation menu** (Navigation menu), click **APIs & services** > **Library**.
2. In the search bar, type **Deployment Manager**, and click the result for **Google Cloud Deployment Manager V2 API**.
3. Verify that the API is enabled.

**Start the Cloud Shell Editor**

To write the configuration and the template, you use the Cloud Shell Editor.

1. In the Cloud Console, click **Activate Cloud Shell** (Cloud Shell).
2. If prompted, click **Continue**.
3. Run the following commands:

mkdir dminfra

cd dminfra

1. In Cloud Shell, click **Launch code editor** (Cloud Shell Editor).
2. In the left pane of the code editor, expand the **dminfra** folder.

**Create the auto mode network configuration**

A configuration is a file written in YAML syntax that lists each of the resources you want to create and their respective resource properties. A configuration must contain a resources: section followed by the list of resources to create. Start the configuration with the **mynetwork** resource.

1. To create a new file, click **File** > **New File**.
2. Name the new file **config.yaml**, and then open it.
3. Copy the following base code into config.yaml:

resources:

# Create the auto-mode network

- name: [RESOURCE\_NAME]

type: [RESOURCE\_TYPE]

properties:

#RESOURCE properties go here

This base configuration is a great starting point for any Google Cloud resource. The **name** field allows you to name the resource, and the **type** field allows you to specify the Google Cloud resource that you want to create. You can also define properties, but these are optional for some resources.

1. In config.yaml, replace [RESOURCE\_NAME] with mynetwork
2. To get a list of all available network resource types in Google Cloud, run the following command in **Cloud Shell**:

gcloud deployment-manager types list | grep network

The output should look like this (**do not copy; this is example output**):

compute.beta.subnetwork

compute.alpha.subnetwork

compute.v1.subnetwork

compute.beta.network

compute.v1.network

compute.alpha.network

Alternatively, you can find the full list of available resource types [here](https://cloud.google.com/deployment-manager/docs/configuration/supported-resource-types).

1. Locate compute.v1.network, which is the type needed to create a VPC network using Deployment Manager.
2. In config.yaml, replace [RESOURCE\_TYPE] with compute.v1.network
3. Add the following property to config.yaml:

autoCreateSubnetworks: true

By definition, an auto mode network automatically creates a subnetwork in each region. Therefore, you are setting **autoCreateSubnetworks** to **true**.

1. Verify that config.yaml looks like this, including the spacing/indentation:

resources:

# Create the auto-mode network

- name: mynetwork

type: compute.v1.network

properties:

autoCreateSubnetworks: true

1. To save config.yaml, click **File** > **Save**.

**Task 2. Configure the firewall rule**

In order to allow ingress traffic instances in **mynetwork**, you need to create a firewall rule.

**Add the firewall rule to the configuration**

Add a firewall rule that allows HTTP, SSH, RDP, and ICMP traffic on **mynetwork**.

1. Copy the following base code into config.yaml (below the **mynetwork** resource):

# Create the firewall rule

- name: mynetwork-allow-http-ssh-rdp-icmp

type: [RESOURCE\_TYPE]

properties:

#RESOURCE properties go here

1. To get a list of all available firewall rule resource types in Google Cloud, run the following command in **Cloud Shell**:

gcloud deployment-manager types list | grep firewall

The output should look like this (**do not copy; this is example output**):

compute.v1.firewall

compute.alpha.firewall

compute.beta.firewall

1. Locate compute.v1.firewall, which is the type needed to create a firewall rule using Deployment Manager.
2. In config.yaml, replace [RESOURCE\_TYPE] with compute.v1.firewall
3. In config.yaml, add the following properties for the firewall rule:

network: $(ref.mynetwork.selfLink)

sourceRanges: ["0.0.0.0/0"]

allowed:

- IPProtocol: TCP

ports: [22, 80, 3389]

- IPProtocol: ICMP

These properties define:

* **network:** Network that the firewall rule applies to
* **sourceRange:** Source IP ranges that traffic is allowed from
* **IPProtocol:** Specific protocol that the rule applies to
* **ports:** Specific ports of that protocol

Because firewall rules depend on their network, you are using the **$(ref.mynetwork.selfLink)** reference to instruct Deployment Manager to resolve these resources in a dependent order. Specifically, the network is created before the firewall rule. By default, Deployment Manager creates all resources in parallel, so there is no guarantee that dependent resources are created in the correct order unless you use references.

1. Verify that config.yaml looks like this, including the spacing/indentation:

resources:

# Create the auto-mode network

- name: mynetwork

type: compute.v1.network

properties:

autoCreateSubnetworks: true

# Create the firewall rule

- name: mynetwork-allow-http-ssh-rdp-icmp

type: compute.v1.firewall

properties:

network: $(ref.mynetwork.selfLink)

sourceRanges: ["0.0.0.0/0"]

allowed:

- IPProtocol: TCP

ports: [22, 80, 3389]

- IPProtocol: ICMP

1. To save config.yaml, click **File** > **Save**.

**Task 3. Create a template for VM instances**

Deployment Manager allows you to use Python or Jinja2 templates to parameterize your configuration. This allows you to reuse common deployment paradigms such as networks, firewall rules, and VM instances.

**Create the VM instance template**

Because you will be creating two similar VM instances, create a VM instance template.

1. To create a new file, click **File** > **New File**.
2. Name the new file **instance-template.jinja**, and then open it.
3. Copy the following base code into instance-template.jinja:

resources:

- name: [RESOURCE\_NAME]

type: [RESOURCE\_TYPE]

properties:

#RESOURCE properties go here

1. In instance-template.jinja, replace [RESOURCE\_NAME] with {{ env["name"] }}. Make sure to include the double brackets {{}}.

This is an environment variable that will be provided by the configuration so that you can reuse the template for multiple instances.

1. To get a list of all available instance resource types in Google Cloud, run the following command in **Cloud Shell**:

gcloud deployment-manager types list | grep instance

The output should look like this (**do not copy; this is example output**):

...

compute.v1.instance

compute.alpha.instance

1. Locate compute.v1.instance, which is the type needed to create a VM instance using Deployment Manager.
2. In instance-template.jinja, replace [RESOURCE\_TYPE] with compute.v1.instance

**Specify the VM instance properties**

To create a VM instance in the correct zone and network, you need to define these as properties.

1. In instance-template.jinja, add the following properties for the VM instance:

machineType: zones/{{ properties["zone"] }}/machineTypes/{{ properties["machineType"] }}

zone: {{ properties["zone"] }}

networkInterfaces:

- network: {{ properties["network"] }}

subnetwork: {{ properties["subnetwork"] }}

accessConfigs:

- name: External NAT

type: ONE\_TO\_ONE\_NAT

disks:

- deviceName: {{ env["name"] }}

type: PERSISTENT

boot: true

autoDelete: true

initializeParams:

sourceImage: https://www.googleapis.com/compute/v1/projects/debian-cloud/global/images/family/debian-9

These properties define:

* **machineType:** Machine type and zone
* **zone:** Instance zone
* **networkInterfaces:** Network and subnetwork that VM is attached to
* **accessConfigs**: Required to give the instance a public IP address (required in this lab). To create instances with only an internal IP address, remove the **accessConfigs** section.
* **disks:** The boot disk, its name and image

Most properties are defined as template properties, which you will provide values for from the top-level configuration (config.yaml).

1. Verify that instance-template.jinja looks like this, including the spacing/indentation:

resources:

- name: {{ env["name"] }}

type: compute.v1.instance

properties:

machineType: zones/{{ properties["zone"] }}/machineTypes/{{ properties["machineType"] }}

zone: {{ properties["zone"] }}

networkInterfaces:

- network: {{ properties["network"] }}

subnetwork: {{ properties["subnetwork"] }}

accessConfigs:

- name: External NAT

type: ONE\_TO\_ONE\_NAT

disks:

- deviceName: {{ env["name"] }}

type: PERSISTENT

boot: true

autoDelete: true

initializeParams:

sourceImage: https://www.googleapis.com/compute/v1/projects/debian-cloud/global/images/family/debian-9

1. To save instance-template.jinja, click **File** > **Save**.

**Task 4. Deploy the configuration**

Configure the instances and deploy the configuration.

**Import the template**

Templates are included in the \*.yaml configuration using import:.

1. Copy the following into config.yaml (before resources):

imports:

- path: instance-template.jinja

The **import** statement defines the template that you want to use in your configuration. You can import multiple templates in one configuration. For example, you could create templates for the firewall rule and networks if you want to reuse those.

**Configure VM instances in each network**

Create the **mynet-us-vm** and **mynet-eu-vm** instances.

1. Add the mynet-us-vm instance to config.yaml (within the resources block):

# Create the mynet-us-vm instance

- name: mynet-us-vm

type: instance-template.jinja

properties:

zone: us-central1-a

machineType: n1-standard-1

network: $(ref.mynetwork.selfLink)

subnetwork: regions/us-central1/subnetworks/mynetwork

1. Add the mynet-eu-vm instance to config.yaml (within the resources block):

# Create the mynet-eu-vm instance

- name: mynet-eu-vm

type: instance-template.jinja

properties:

zone: europe-west1-d

machineType: n1-standard-1

network: $(ref.mynetwork.selfLink)

subnetwork: regions/europe-west1/subnetworks/mynetwork

The **zone**, **machineType**, and **subnetwork** are passed from this configuration to the template. This allows you to create only one template for multiple VM instances. Also, references to the network are used for the VM instances. This ensures that the network is created before the VM that is attached to that network.

1. To save config.yaml, click **File** > **Save**.

**Deploy the configuration**

It's time to deploy your configuration from Cloud Shell.

1. In Cloud Shell, run the following command:

gcloud deployment-manager deployments create dminfra --config=config.yaml --preview

The output should look like this (**do not copy; this is example output**):

NAME TYPE STATE

mynet-eu-vm compute.v1.instance IN\_PREVIEW

mynet-us-vm compute.v1.instance IN\_PREVIEW

mynetwork compute.v1.network IN\_PREVIEW

mynetwork-allow-http-ssh-rdp-icmp compute.v1.firewall IN\_PREVIEW

The **--preview** flag gives you a preview of how your configuration is applied before creating it. Previewing a configuration causes Deployment Manager to start creating your deployment but then stop before actually creating any resources. The **--preview** flag is especially useful when you [update a deployment](https://cloud.google.com/deployment-manager/docs/step-by-step-guide/updating-a-deployment#previewing_an_update).

1. Run the following command to create the deployment:

gcloud deployment-manager deployments update dminfra

The **update** command commits the preview. If you don't preview a configuration, you can directly run the following command:

gcloud deployment-manager deployments create dminfra --config=config.yaml

1. Wait for the resources to be created and listed in Cloud Shell.

The output should look like this (**do not copy; this is example output**):

NAME TYPE STATE ERRORS INTENT

mynet-eu-vm compute.v1.instance COMPLETED []

mynet-us-vm compute.v1.instance COMPLETED []

mynetwork compute.v1.network COMPLETED []

mynetwork-allow-http-ssh-rdp-icmp compute.v1.firewall COMPLETED []

If something goes wrong with the preview or creation of the deployment, try to use the error messages to troubleshoot the source of the issue. You must delete the Deployment Manager configuration before you can try deploying it again. This can be achieved with this command in Cloud Shell:

gcloud deployment-manager deployments delete dminfra

If you cannot troubleshoot the issue of your deployment, take a look at this finished configuration ([config.yaml](https://storage.googleapis.com/cloud-training/archinfra/dminfra/config.yaml)) and template ([instance-template.jinja](https://storage.googleapis.com/cloud-training/archinfra/dminfra/instance-template.jinja)).

Click Check my progress to verify the objective.

Create network, firewall rules and VM instances

**Task 5. Verify your deployment**

**Verify your network in the Cloud Console**

1. In the Cloud Console, on the **Navigation menu** (Navigation menu), click **VPC network** > **VPC networks**.
2. View the **mynetwork** VPC network with a subnetwork in every region.
3. On the **Navigation menu**, click **VPC network** > **Firewall Rules**.
4. Sort the firewall rules by **Network**.
5. View the **mynetwork-allow-http-ssh-rdp-icmp** firewall rule for **mynetwork**.

**Verify your VM instances in the Cloud Console**

1. On the **Navigation menu** (Navigation menu), click **Compute Engine** > **VM instances**.
2. View the **mynet-us-vm** and **mynet-eu-vm** instances.
3. Note the internal IP address for **mynet-eu-vm**.
4. For **mynet-us-vm**, click **SSH** to launch a terminal and connect.
5. To test connectivity to **mynet-eu-vm**'s internal IP address, run the following command in the SSH terminal (replacing mynet-eu-vm's internal IP address with the value noted earlier):

ping -c 3 <Enter mynet-eu-vm's internal IP here>

This should work because both VM instances are on the same network and the firewall rule allows ICMP traffic!

**Task 6. Review**

In this lab, you created a Deployment Manager configuration and template to automate the deployment of Google Cloud infrastructure. Templates can be very flexible because of their environment and template variables. Therefore, the benefit of creating templates is that they can be reused across many configurations.

Template variables are abstract properties that allow you to declare the value to be passed to the template in the \*.yaml configuration file. You can change the value for each deployment in the \*.yaml file without having to make changes to the underlying templates.

Environment variables allow you to reuse templates in different projects and deployments. Instead of representing properties of resources, they represent more global properties such as a Project ID or the name of the deployment. You can use the template that you created as a starting point for future deployments.

**End your lab**