GCP-Kubernetes-Lab/Demo

Securing your container workloads in Kubernetes

http://www.paloaltonetworks.com

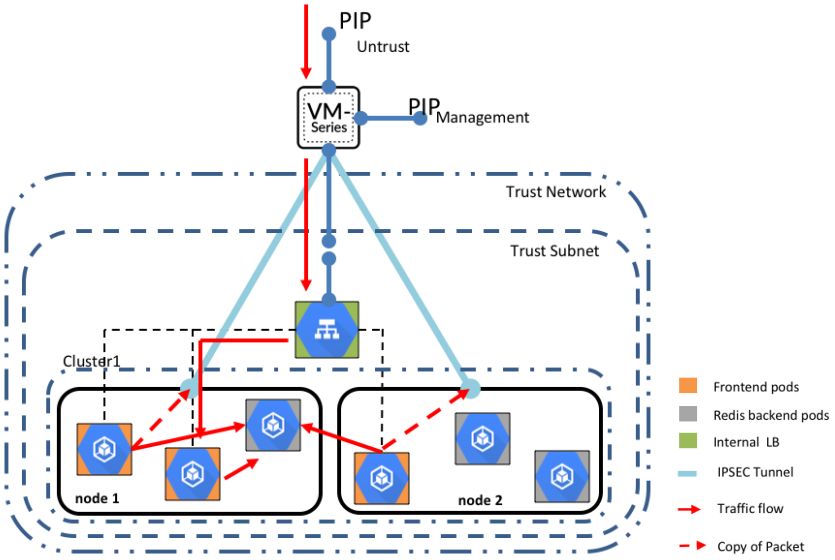
Table of Contents

About GCP Terraform Template

GCP Terraform Templates are files that can deploy, configure, and launch GCP resources such as VPC networks, subnets, security groups, firewall rules, route tables, Kubernetes clusters, and more. These templates are used for ease of deployment and are key to any cloud deployment model.

For more information on Templates refer to Google's documentation
https://cloud.google.com/community/tutorials/managing-gcp-projects-with-terraform
There are also many Terraform template s available here:
https://github.com/GoogleCloudPlatform/terraform-google-examples

This document will walk through deploying a Terraform template and a few Kubernetes(k8s) pods that host a two-tier web application. This template is designed to be deployed into an existing GCP project. After completing this guide, the following infrastructure will be instantiated:



This lab will also show one method to obtain east west traffic visibility between the pods which are hosting the two-tier application.

Support Policy

This template is released under an as-is, best effort, support policy. These scripts should be seen as community supported and Palo Alto Networks will contribute our expertise as and when possible. We do not provide technical support or help in using or troubleshooting the components of the project through our normal support options such as Palo Alto Networks support teams, or ASC (Authorized Support Centers) partners and backline support options. The underlying product used (the VM-Series firewall) by the scripts or templates are still supported, but the support is only for the product functionality and not for help in deploying or using the template or script itself.

Instances Used

When deploying this Terraform template the following machine types are used:

Instance	Machine Type	QTY
PayGo Bundle 1 - VM Series Firewall	n1-standard-4	1
Kubernetes Ubuntu Cluster Nodes	n1-standard-1	2
Internal Load Balancer		1

Note: There are GCP costs associated with each machine type launched, please refer to the Google instance pricing page https://cloud.google.com/compute/pricing

Prerequisites

Here are the prerequisites required to successfully launch this template:

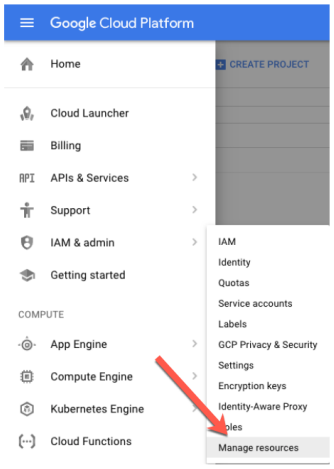
- Terraform application - Instructions on the installation can be found here: https://www.terraform.io/intro/getting-started/install.html
- GCP account- Account creation can be done here: https://cloud.google.com/free/
- Google Cloud SDK- GCP template installations in this guide are performed from the CLI. Install the SDK/CLI by selecting the relevant platform from the following link and following the installation instructions: https://cloud.google.com/sdk/

GCP Project Setup

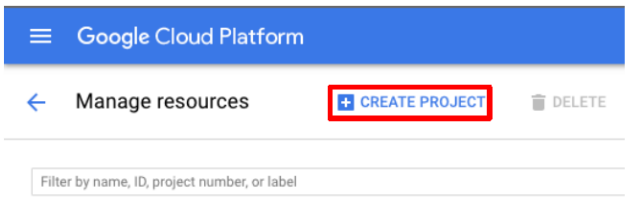
Task 1 – Create a Project

All GCP resources in this guide are deployed to a single project, which is an organizational boundary that separates users, resources, billing information, etc. It is similar to an AWS VPC or an Azure Resource Group. By default, GCP will create a project upon creation of an account.

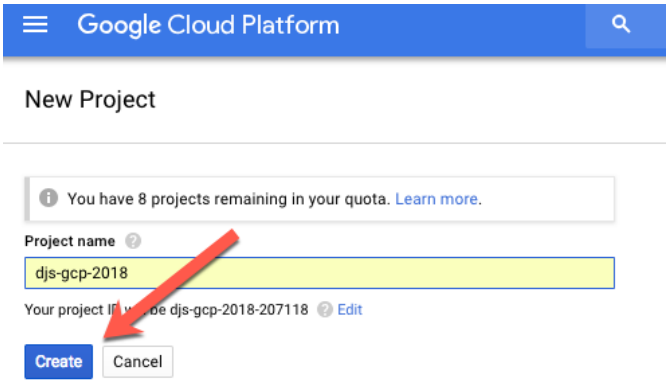
To create an additional dedicated project, use the drop-down on the left and select **IAM & admin > Manage Resources:**



Click **Create Project:**



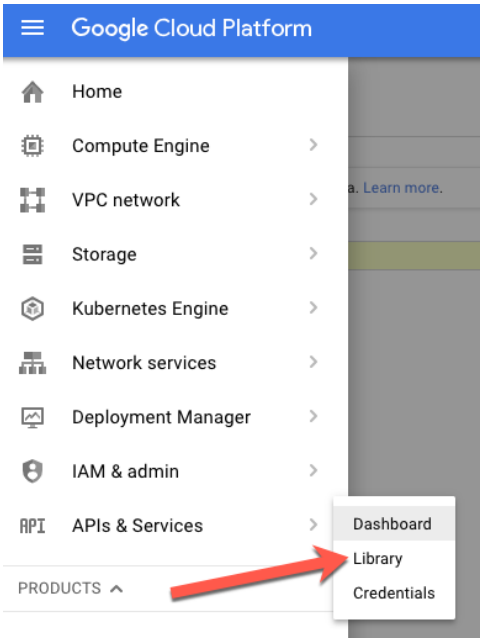
Specify a name for the project and click **Create**:



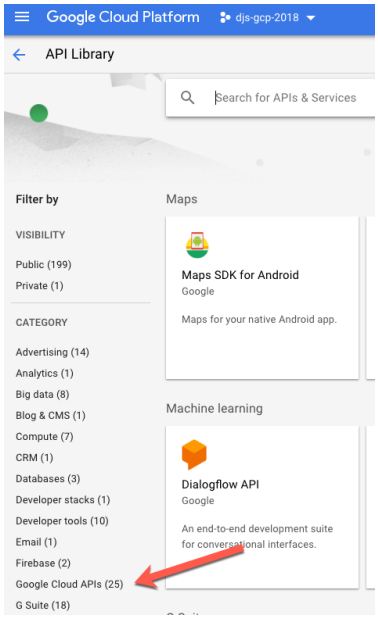
Note that project creation will take a few minutes.

Task 2 – Enable the API

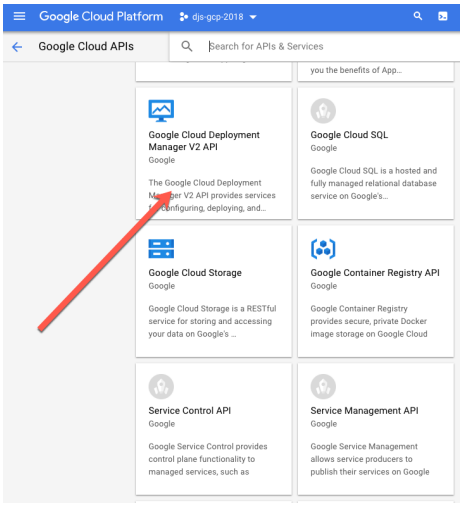
Deploying a template requires the Cloud Deployment Manager API be enabled on the project. Navigate to **APIs & Services >Library**:



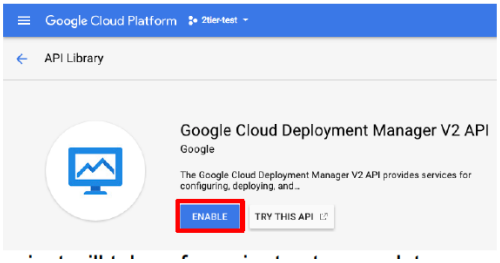
Select Google Cloud APIs on the Left-Hand-Side:



Select Google Cloud Deployment Manager V2 API:



Select **Enable**:



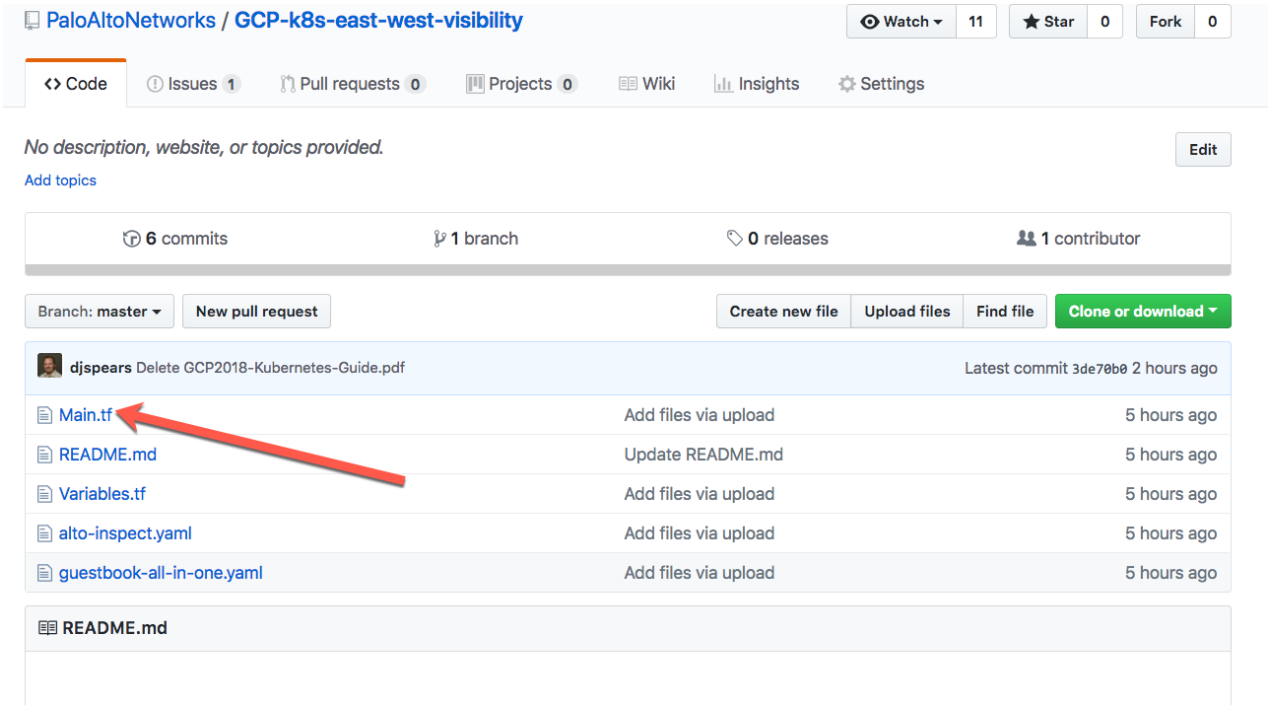
Enabling the API for the project will take a few minutes to complete.

Terraform File setup

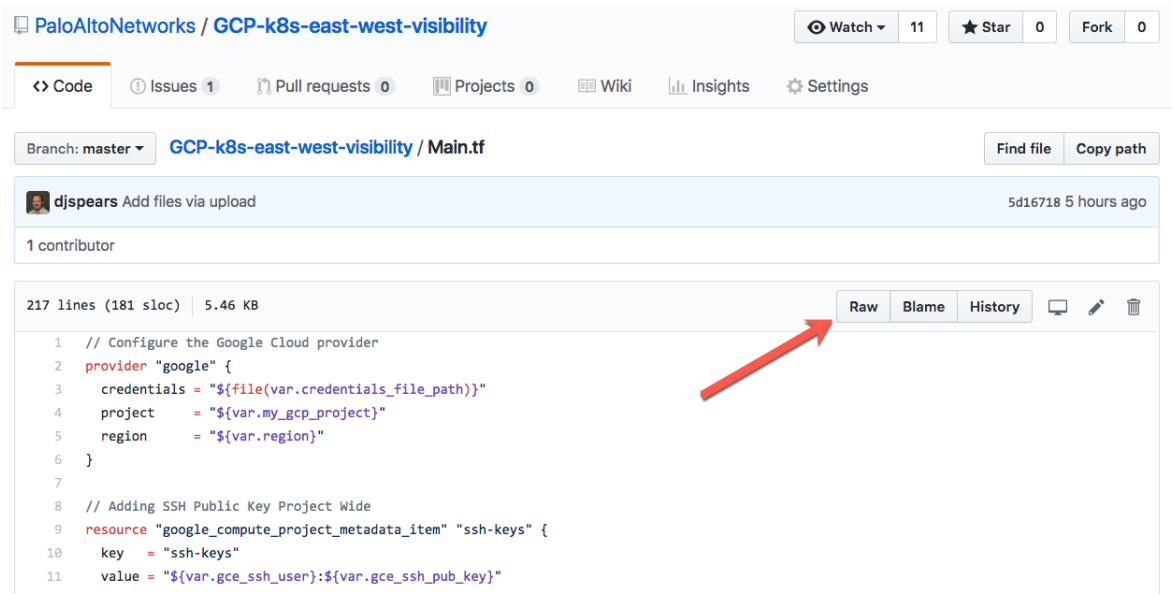
Task 1 - Download Terraform files

Navigate to the following GitHub repository to get the terraform files:
https://github.com/PaloAltoNetworks/GCP-k8s-east-west-visibility/tree/master

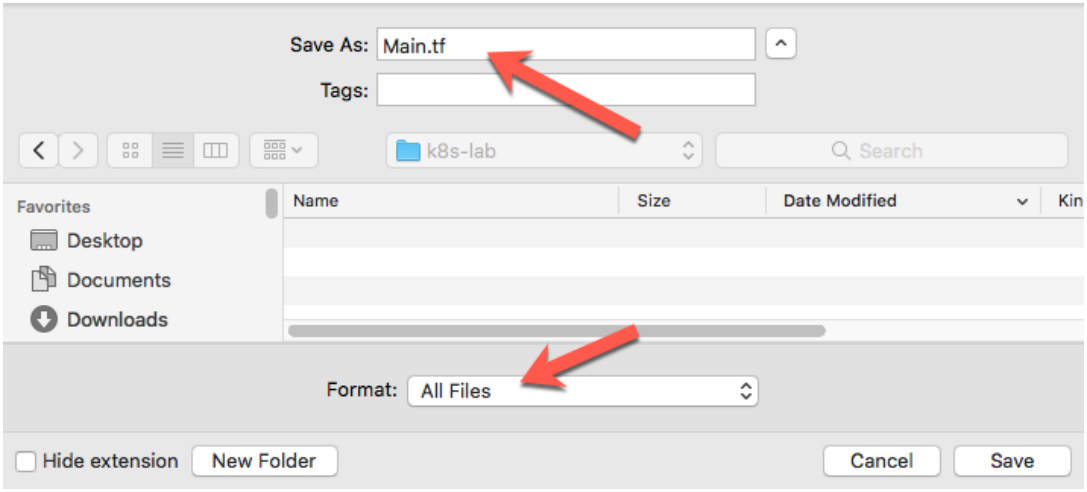
For this lab, the Main.tf and Variables.tf files will need to be downloaded to your local machine. Click on the Main.tf file:



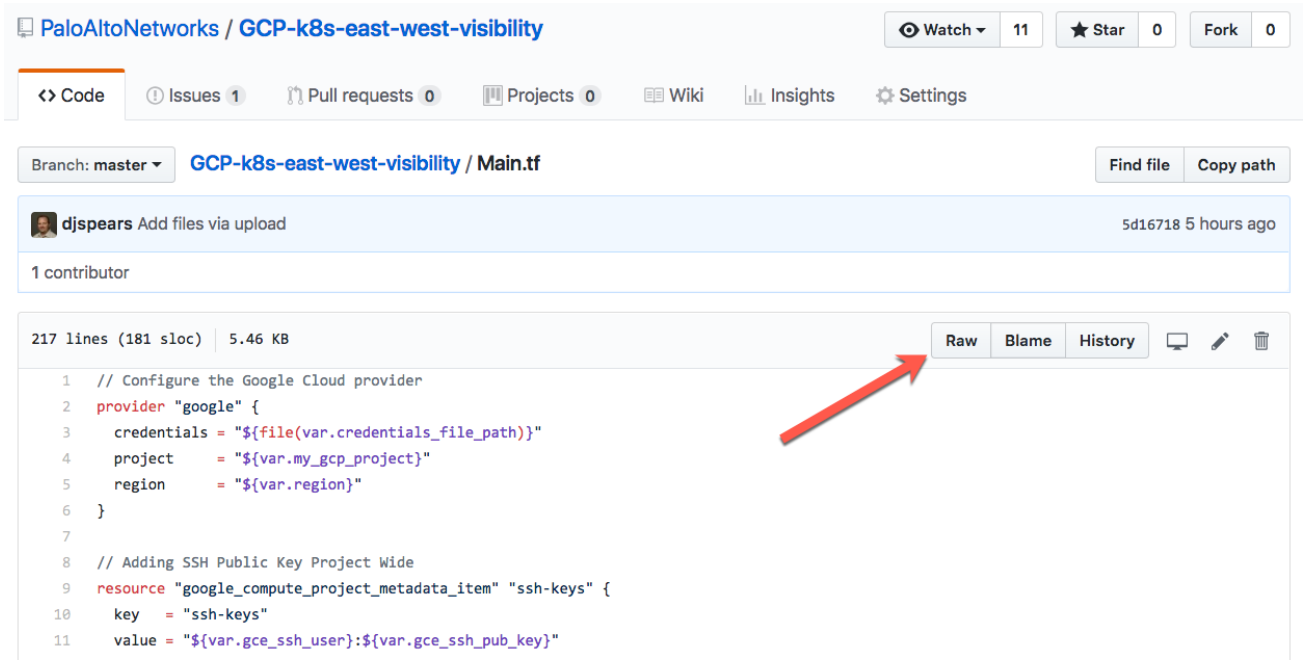
Next Select Raw and save to a known local location:



Make sure the .tf file extension is added to the file. The following screenshot shows the mac save as dialogue box where the extension needs to be added and the Format needs to be changed to All Files:

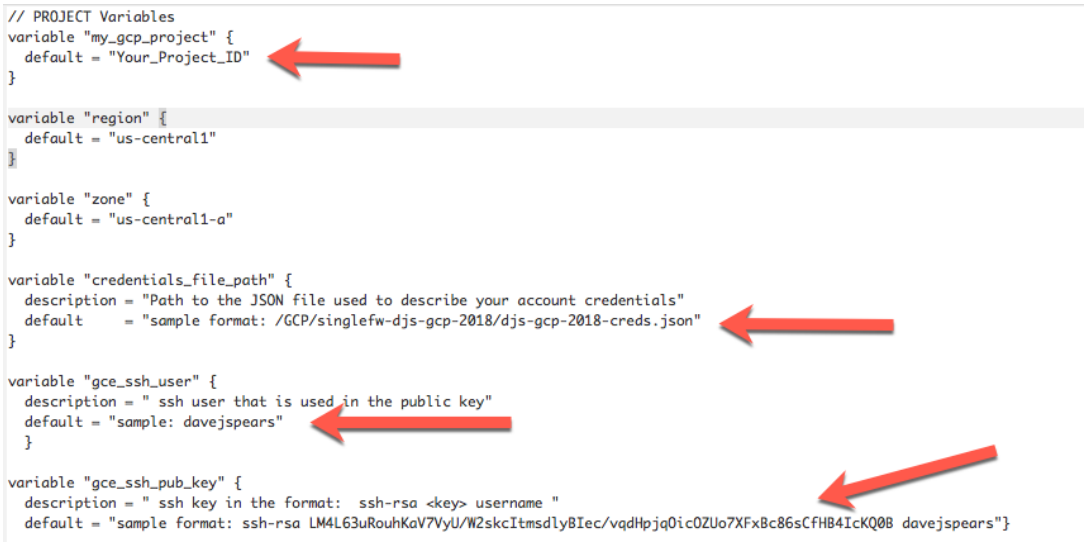


Return to the repository and follow the previous steps to download the Variables.tf file to the same location as the Main.tf:



Task 2 - Gather Information and Update the Template File

The Terraform deployment files consist of a main file and a variables file. The Variables.tf file contains information that is easily modified and commonly changed for various situations. The variables in the Variables.tf file are used by the Main.tf file during deployment. Deploying this Terraform template in GCP does require modification of the template Variable.tf file to include deployment-specific information. The fields that must be updated are shown below:



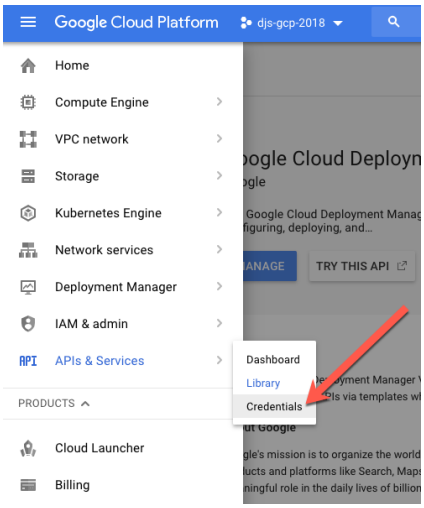
```
// PROJECT Variables
variable "my_gcp_project" {
  default = "Your_Project_ID"
}

variable "region" {
  default = "us-central1"
}

variable "zone" {
  default = "us-central1-a"
}

variable "credentials_file_path" {
  description = "Path to the JSON file used to describe your account credentials"
  default     = "sample format: /GCP/singlefw-djs-gcp-2018/djs-gcp-2018-creds.json"
}

variable "gce_ssh_user" {
  description = " ssh user that is used in the public key"
  default = "sample: davejspears"
  }

variable "gce_ssh_pub_key" {
  description = " ssh key in the format:  ssh-rsa <key> username "
  default = "sample format: ssh-rsa LM4L63uRouhKaV7VyU/W2skcItmsdlyBIec/vqdHpjqOicOZUo7XFxBc86sCfHB4IcKQ0B davejspears"}
```

Project ID is the id that is associated with the GCP project that was created previously.

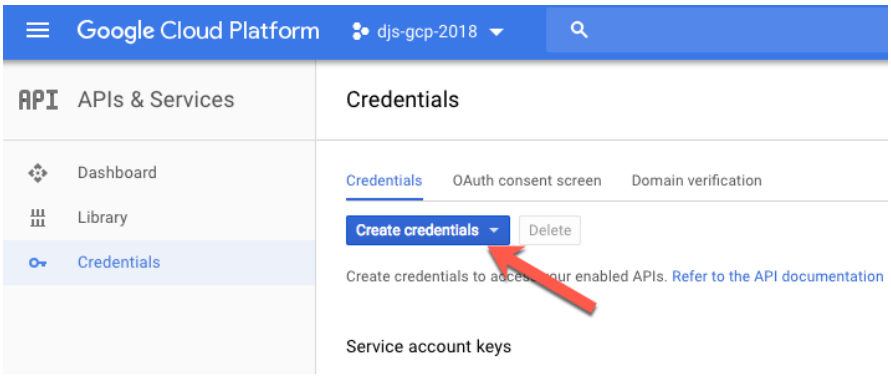
Credential file path is the path to the JSON file that has service credentials that will be used to deploy the template.

SSH User and SSH Pub Key are the SSH keys that can be used to access VM resources in the environment. In this lab the Firewall will be bootstrapped from an existing public GCP bucket. However, if that process fails, this SSH user can be used to access the firewall CLI.

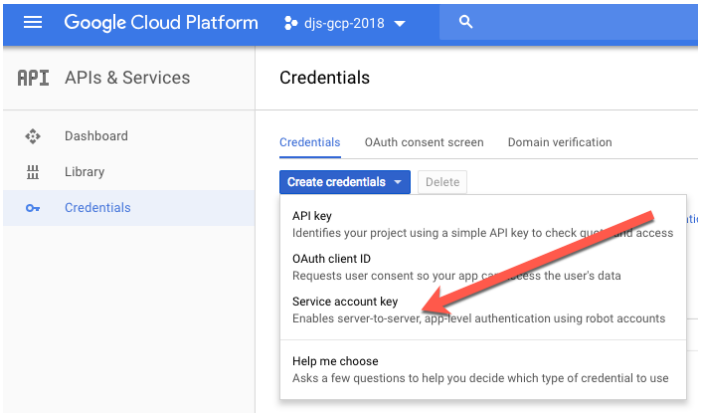
To create the credentials to access the APIs in JSON format. In GCP console go to (APIs & Services > Credentials:



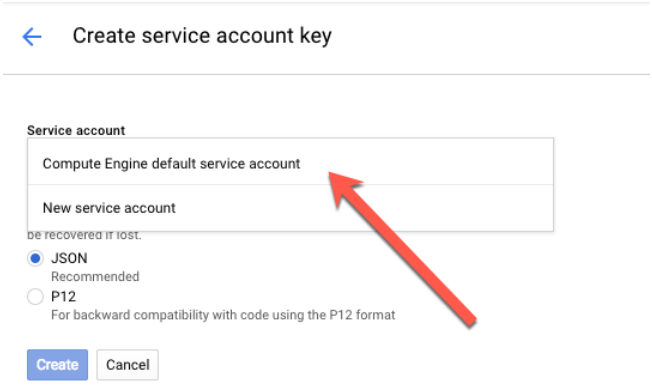
Click Create Credentials:



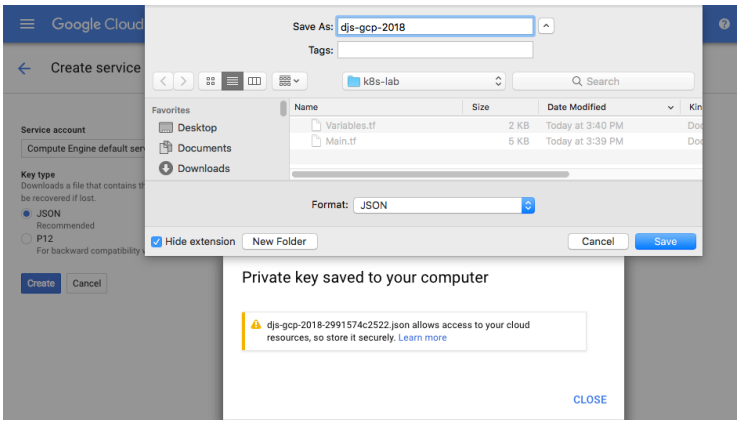
Then select Service Account Key:



Pick Compute Engine default service account and make sure the JSON format is ticked. Click Create:



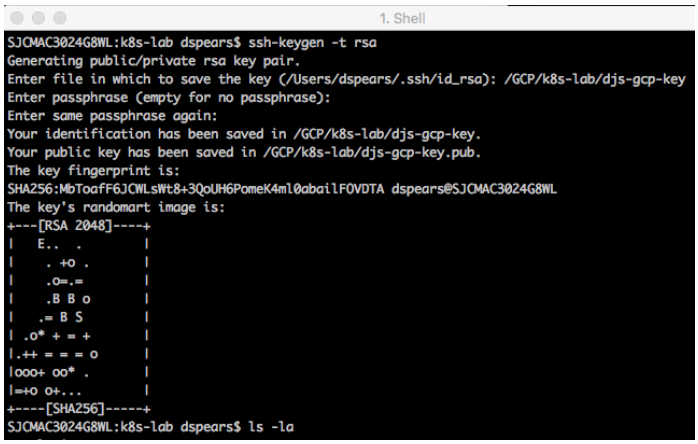
download the file to your computer. It is easy to put the credential file in the same folder as the terraform Main.tf and Variables.tf files:



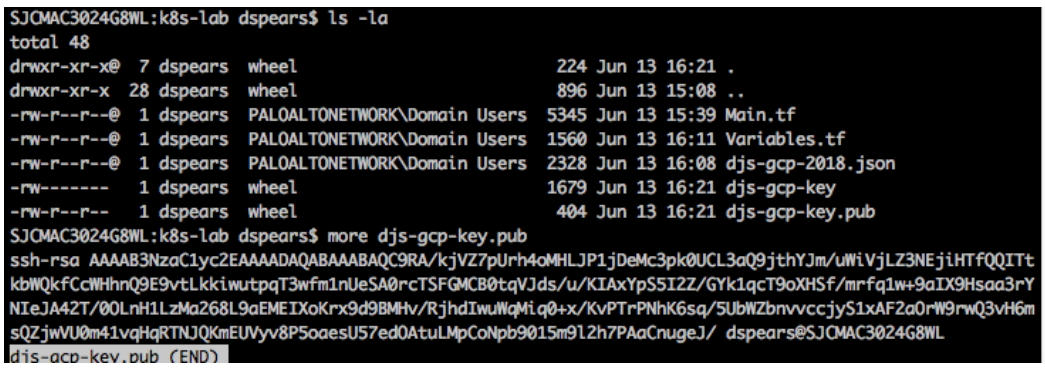
Use an editor of your choice to update the Variables.tf file with the appropriate path:

```
// PROJECT Variables
variable "my_gcp_project" {
  default = "Your_Project_ID"
}

variable "region" {
  default = "us-central1"
}

variable "zone" {
  default = "us-central1-a"
}

variable "credentials_file_path" {
  description = "Path to the JSON file used to describe ...ccount credentials"
  default     = "/GCP/k8s-lab/djs-gcp-2018.json"
}
```

If you do not already have an SSH key, the follow example shows how to create a SSH key on a Mac using the ssh-keygen -t rsa command:



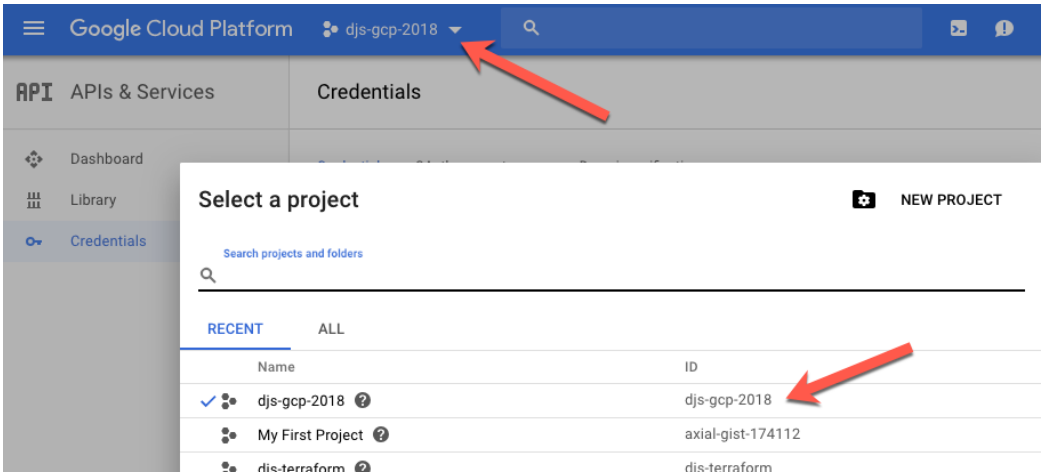
In the previous example the keys were generated and stored in the same directory as the other lab files. The public and private keys can be seen using the ls -la command. I have also displayed the public key via the more command in the following screenshot:



Retrieve the username and key from the public key file. Then update the Variables.tf file:

```
// PROJECT Variables
variable "my_gcp_project" {
  default = "djs-gcp-2018"
}

variable "region" {
  default = "us-central1"
}

variable "zone" {
  default = "us-central1-a"
}

variable "credentials_file_path" {
  description = "Path to the JSON file used to describe your account credentials"
  default     = "/GCP/k8s-lab/djs-gcp-2018.json"
}

variable "gce_ssh_user" {
  description = " ssh user that is used in the public key"
  default = "dspears@SJCMAC3024G8WL"
  }

variable "gce_ssh_pub_key" {
  description = " ssh key in the format:  ssh-rsa <key> username "
  default = "ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQC9RA/kjVZ7pUrh4oMHLJP1jDeMc3pk0UCL3aQ9jthYJm/uWiVjLZ3NEjiHTfQQITtkbWQkfCcWHhnQ9E9vtLkkiwutpqT3wfm1nUeSA0rcTSFGMCB0tqVJds/u/KIAxYpS5I2Z/GYk1qcT9oXHSf/mrfq1w+9aI
//The rest of the variables do not need to be modified for the K8s Lab
// VM-Series Firewall Variables
```

To get the GCP Project ID click the project selection drop down at the top of the GCP Console. The Project ID is displayed in the project selection window:



Update the Variables.tf file with the GCP ID:

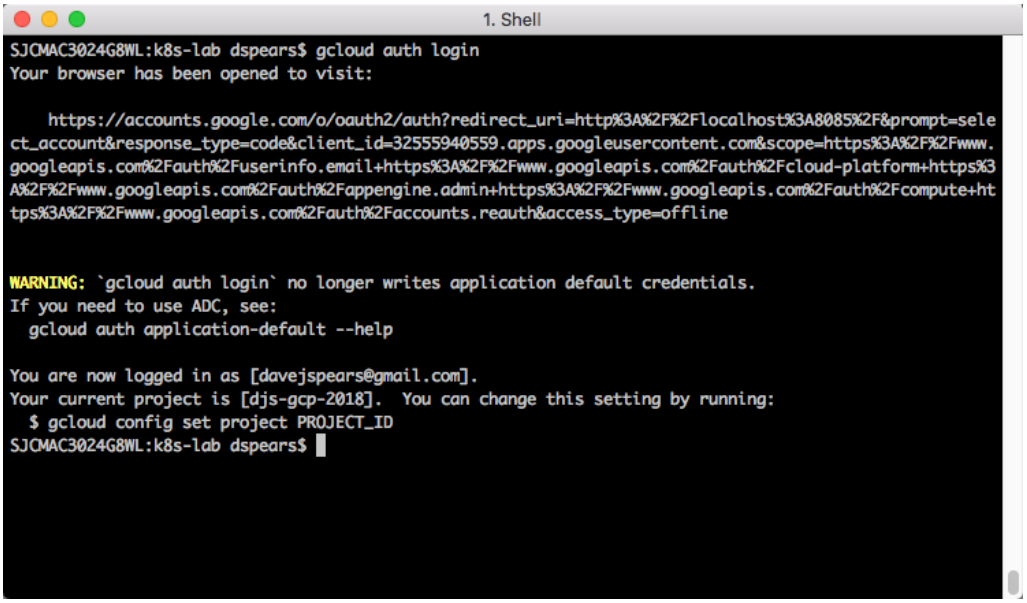
```
// PROJECT Variables
variable "my_gcp_project" {
  default = "djs-gcp-2018"
}

variable "region" {
  default = "us-central1"
}

variable "zone" {
  default = "us-central1-a"
}

variable "credentials_file_path" {
  description = "Path to the JSON file used to describe your account credentials"
  default     = "/GCP/k8s-lab/djs-gcp-2018.json"
}

variable "gce_ssh_user" {
  description = " ssh user that is used in the public key"
  default = "dspears@SJCMAC3024G8WL"
  }

variable "gce_ssh_pub_key" {
  description = " ssh key in the format:  ssh-rsa <key> username "
  default = "ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQC9RA/kjVZ7pUrh4oMHLJP1jDeMc3pk0UCL3aQ9jthYJm/uWiVjLZ3NEjiHTfQQITtkbWQkfCcWHhnQ9E9vtLkkiwutpqT3wfm1nUeSA0rcTSFGMCB0tqVJds/u/KIAxYpS5I2Z/GYk1qcT9oXHSf/mrfq1w+9aI
//The rest of the variables do not need to be modified for the K8s Lab
// VM-Series Firewall Variables
```

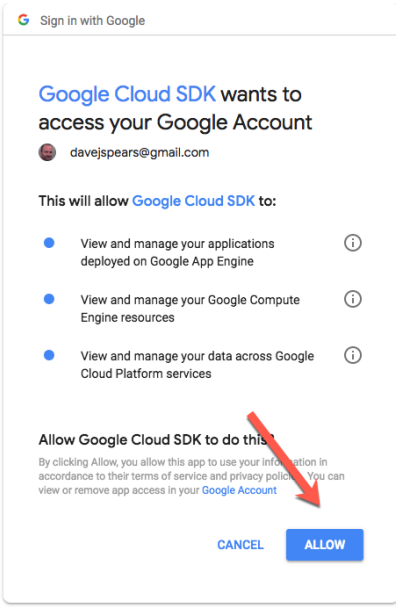
Deploy the Terraform Template

Task 1 – Connect to GCP

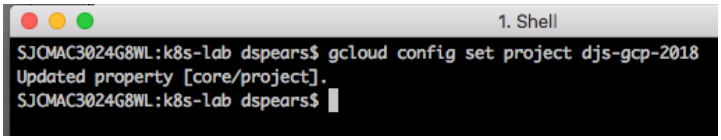
Open a terminal shell and navigate to the directory containing the Terraform template files.

Authenticate to the GCP environment from the command line with the command:
$ gcloud auth login
• Copy/paste the link into a browser(if needed) and select the account to authenticate if a browser does not automatically launch:
• Review the requested permissions and click Allow:





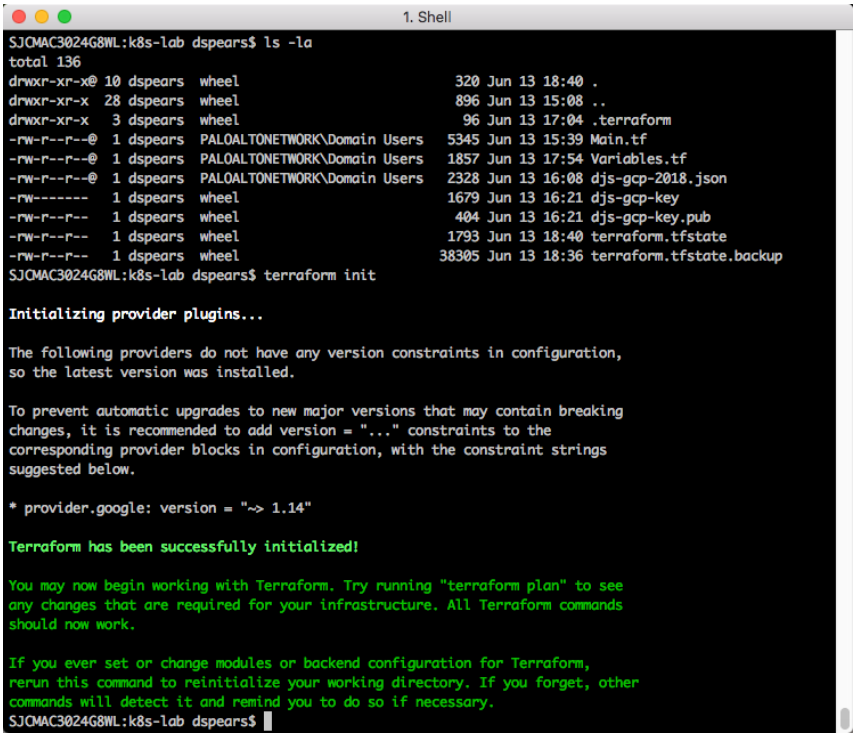
Note the Project ID. If it is not the project that was created for the lab, use the **$ gcloud config set project my_Project_id** command to change the project:



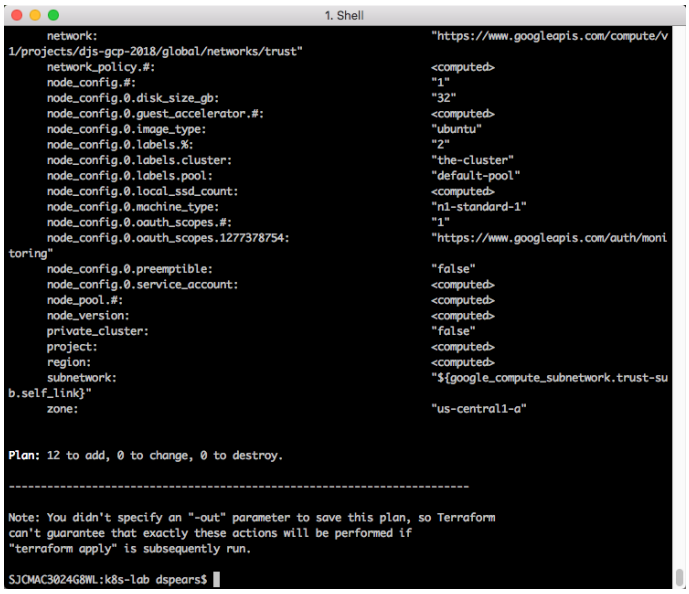
Task 2 – Deploy the Terraform Template

Ensure you are in the directory with the Main.tf and Variables.tf files and execute the

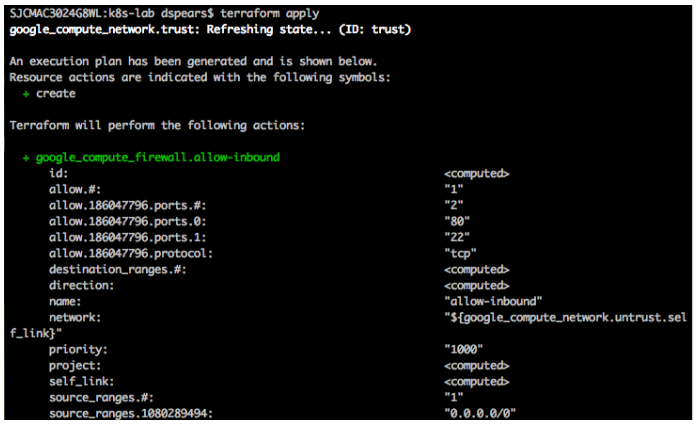
terraform init command which will initialize terraform:

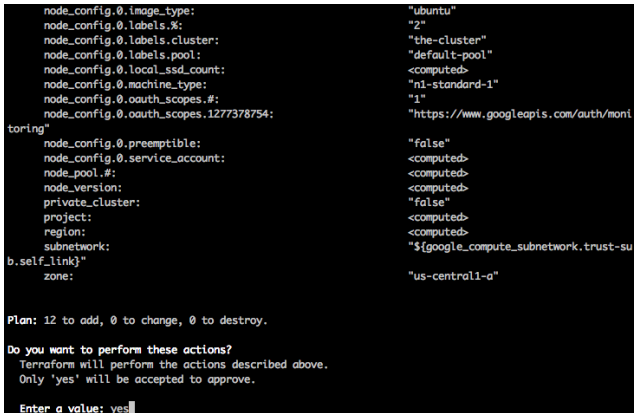


Once the terraform init has completed run the **terraform plan** command. This will show what changes will be implemented with the terraform script. This will also identify if there are any errors detected with the terraform files:

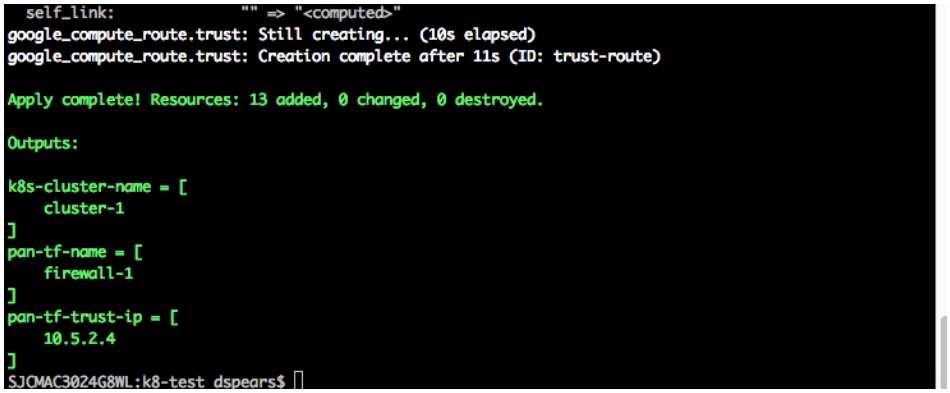


Now run the **terraform apply** command to deploy the template. At the action prompt enter **yes**.





It will take a few minutes to complete. If all goes well, Terraform will output; “Apply Complete!” and provide some additional output information about the resources deployed:



Review what was deployed

In this activity, you will:

Review the resources that have been launched

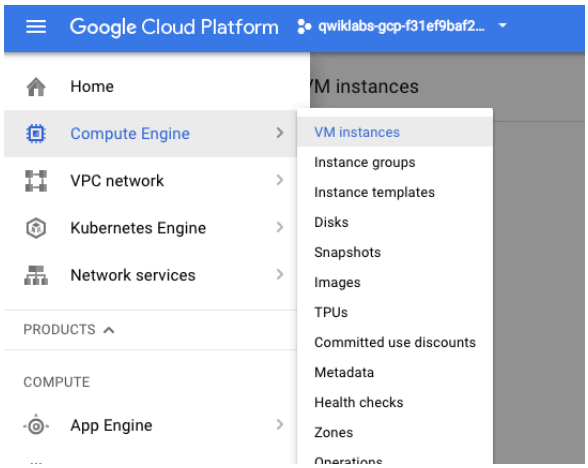
Log into the VM-Series firewall

Confirm bootstrap success

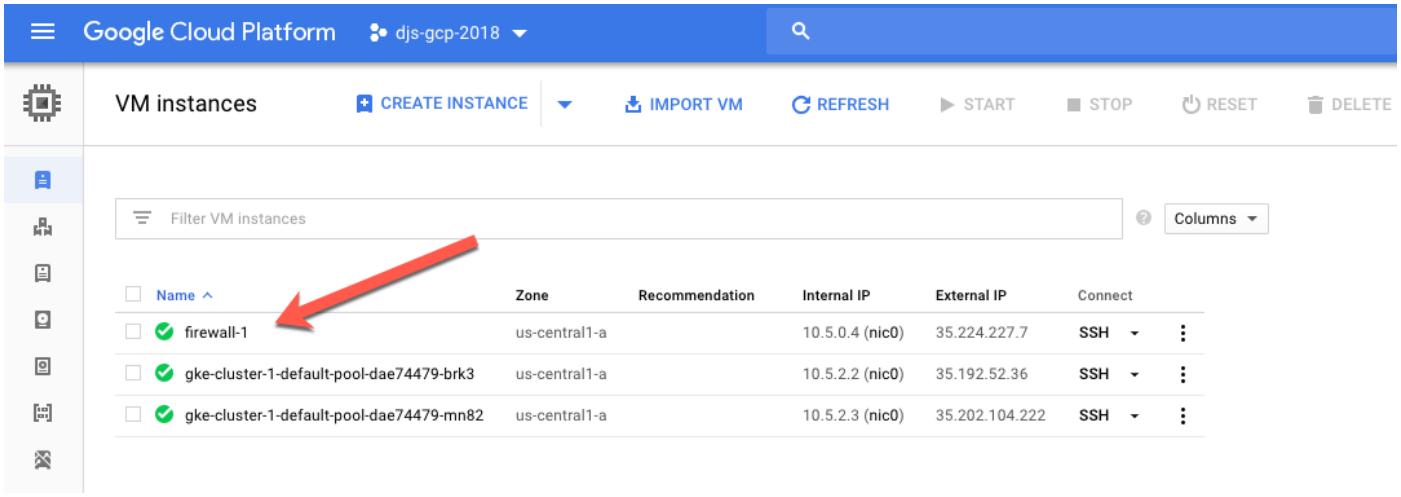
Inspect k8s cluster

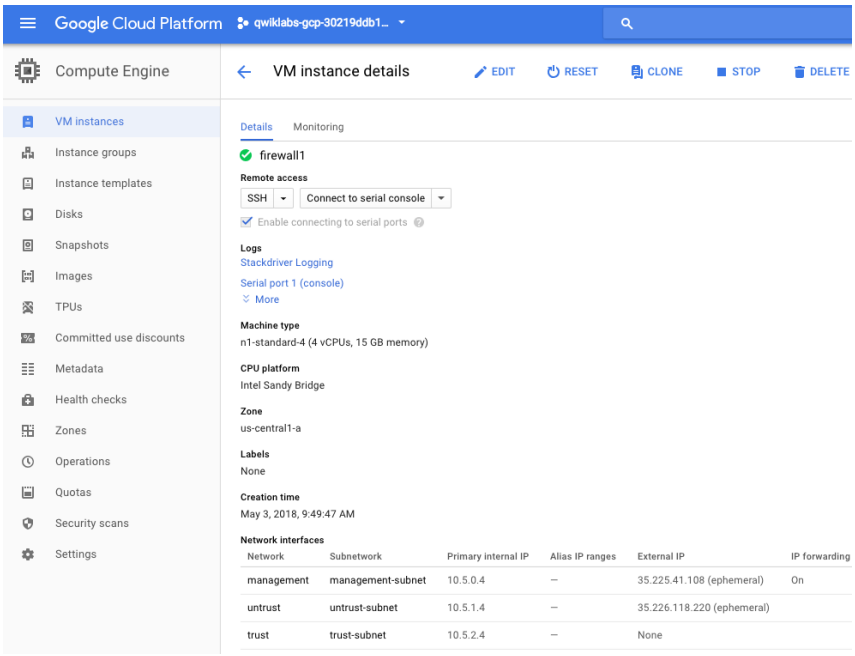
Task 1 – Look around GCP console

Navigate to Compute Engine > VM Instances to see what VMs the terraform template deployed:

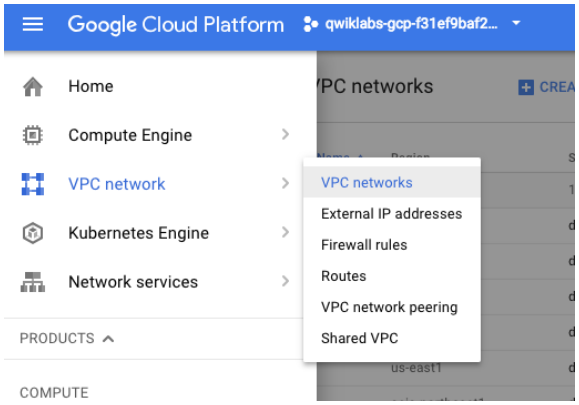


There should be 1 firewall and two k8s nodes displayed. Click on the firewall to open a detailed view of the deployed firewall:

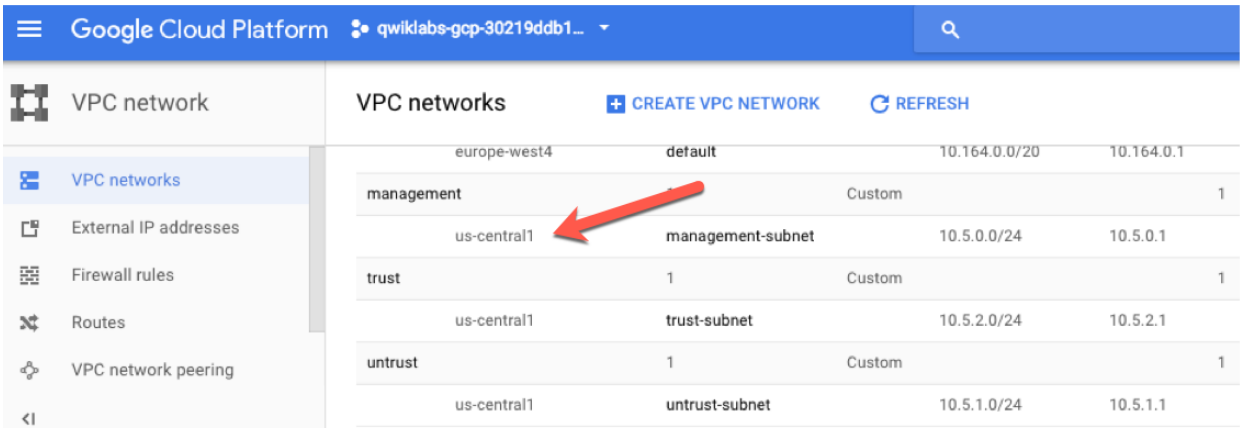




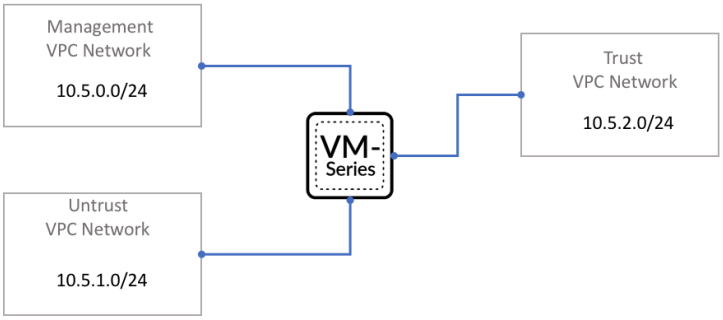
Navigate to VPC Network > VPC Networks to see the different networks that have been created as part of the lab.



You should see 3 VPC networks: management, trust and untrust.



The following diagram describes the network topology of what has been deployed:



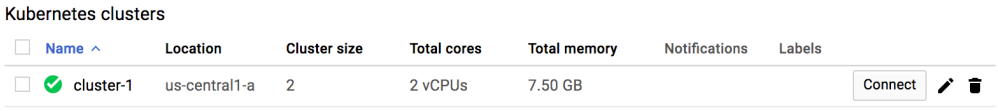
Feel free to navigate through other parts of the Google Cloud Console such as VPC Networks > Routes, Firewall Rules. This will come in handy in activities later on.

Task 2 – Review the Kubernetes Cluster

Kubernetes is a portable, extensible, open-source orchestrator that is used to manage containerized workloads. Kubernetes has a large and rapidly growing ecosystem. The portability of Kubernetes allows for workloads to be migrated between various clouds (public or private). Further documentation is available at:

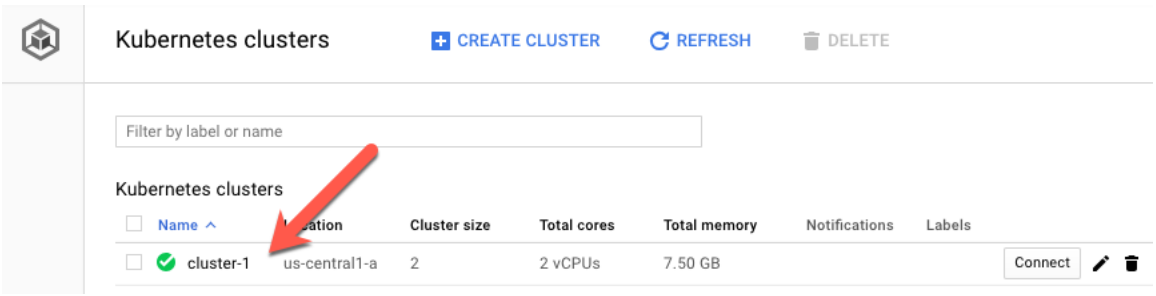
https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/

Navigate to Kubernetes Engine > Kubernetes clusters:

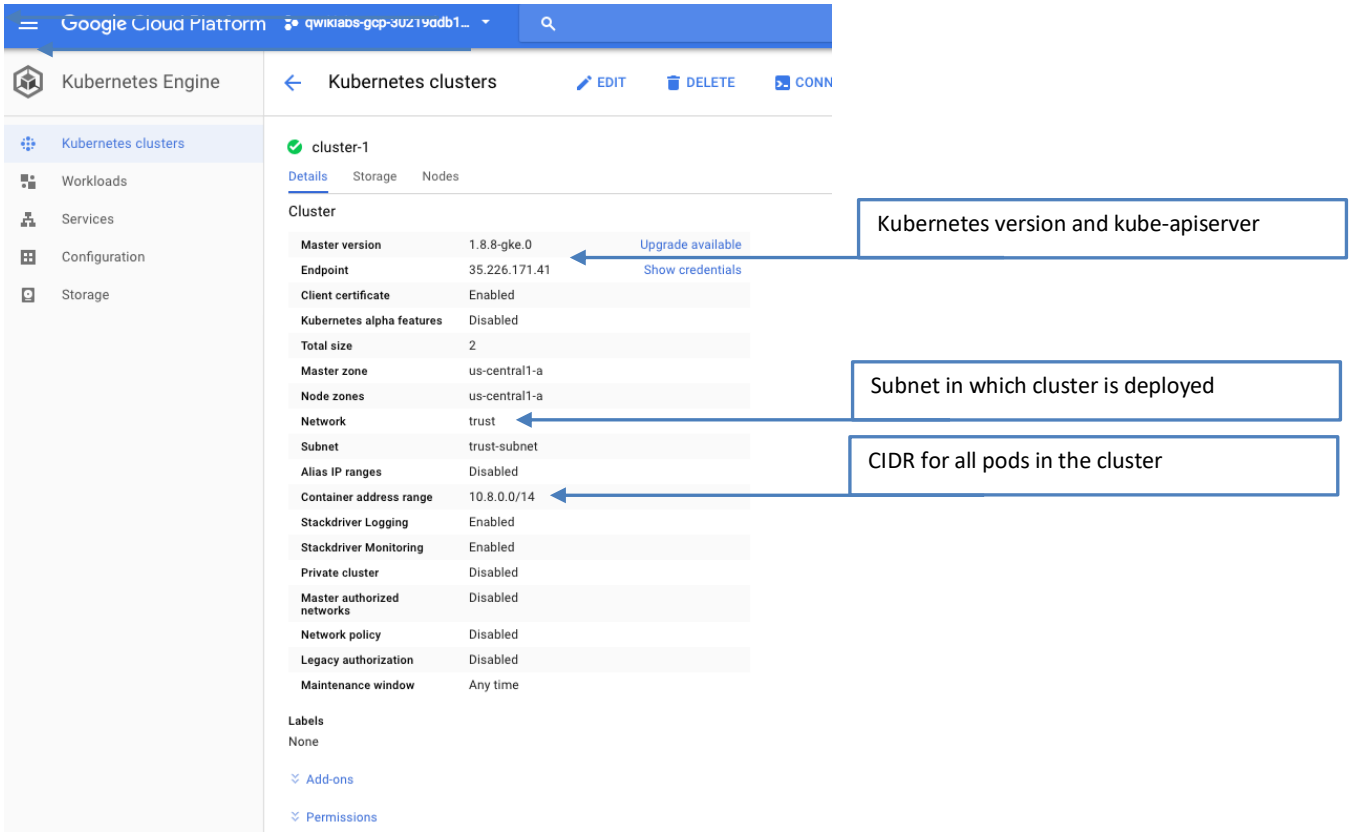


Task 2 – Explore the cluster

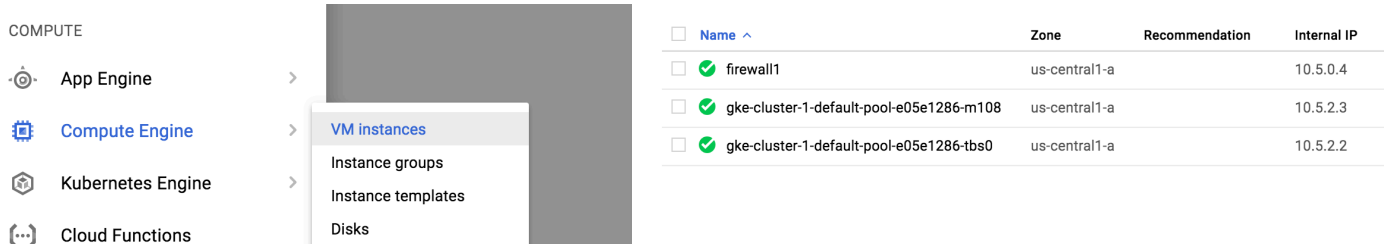
Let's explore some aspects of the Kubernetes cluster that were deployed as part of the terraform template. Click into the cluster to see some of the details of the cluster:



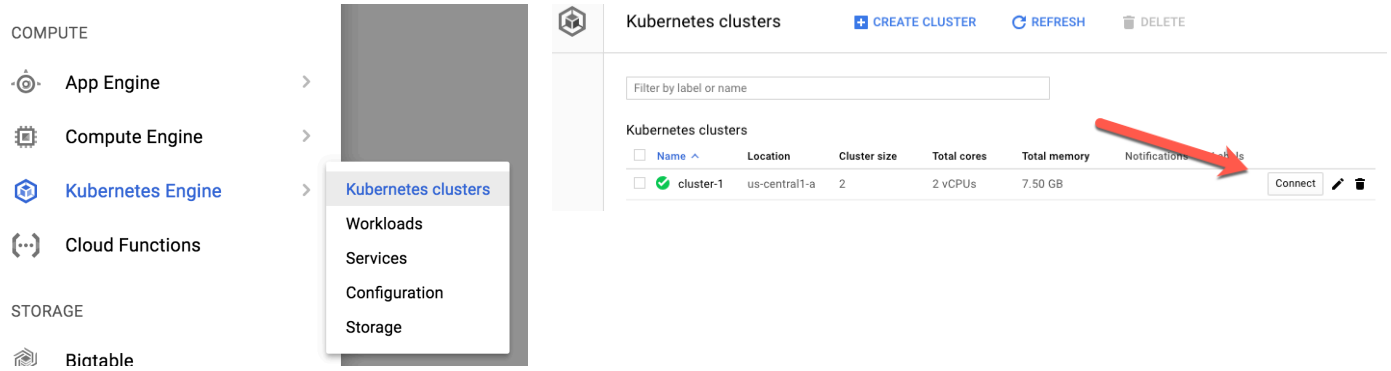
Below outlines some interesting info. As you can see the nodes reside in the same trust subnet as the firewall's trust interface:



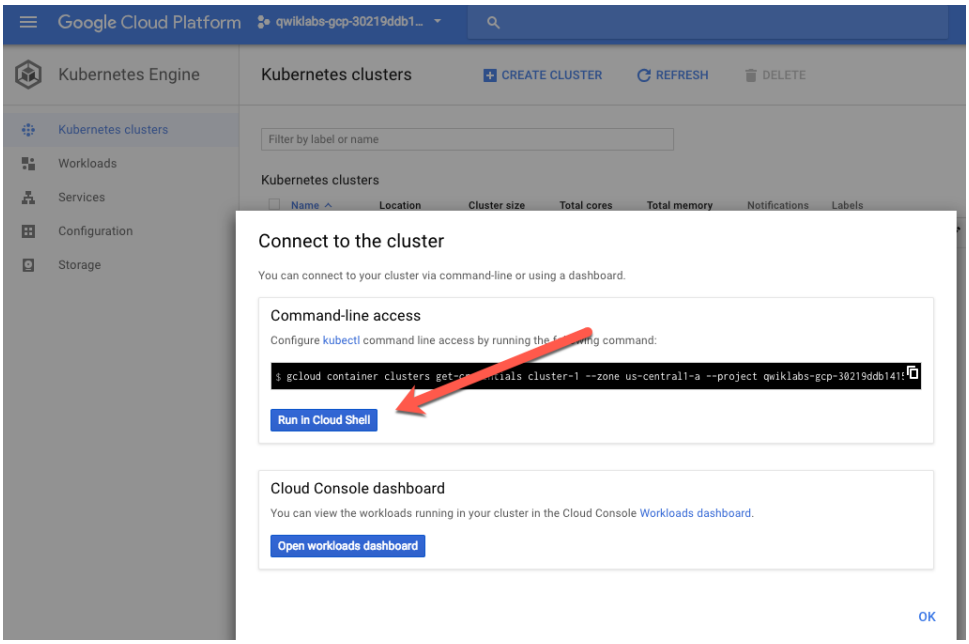
Navigate to the VM Instances to verify that the nodes are deployed. Note the Node internal IP addressing:



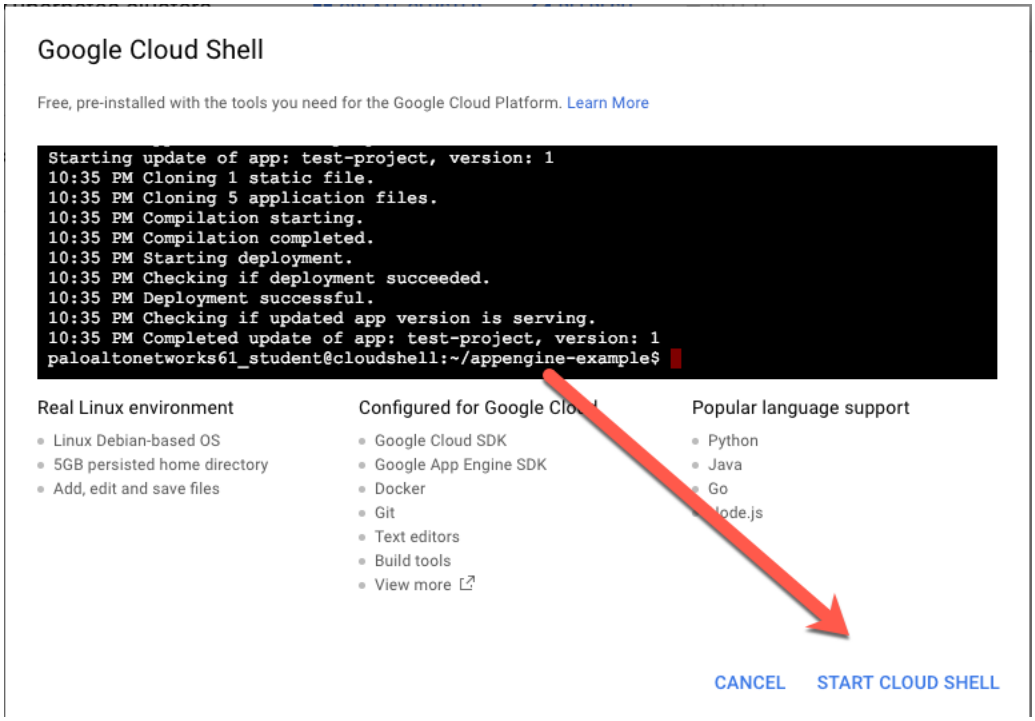
Head back to the cluster console and connect to the cluster:



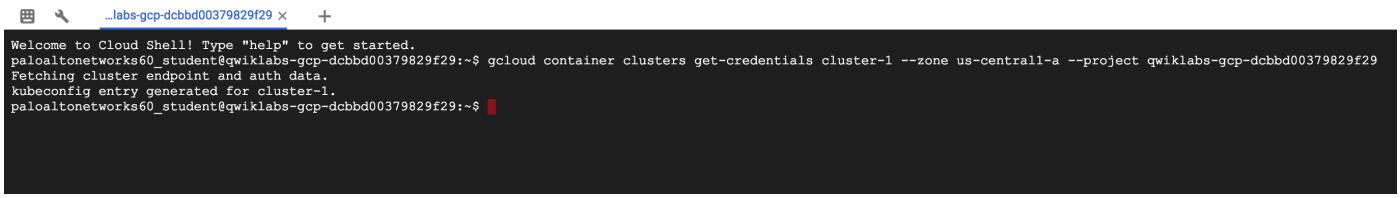
And connect via the cloud shell:



Clicking Run in Cloud Shell starts a google cloud shell. If this dialog box opens, click start cloud shell:



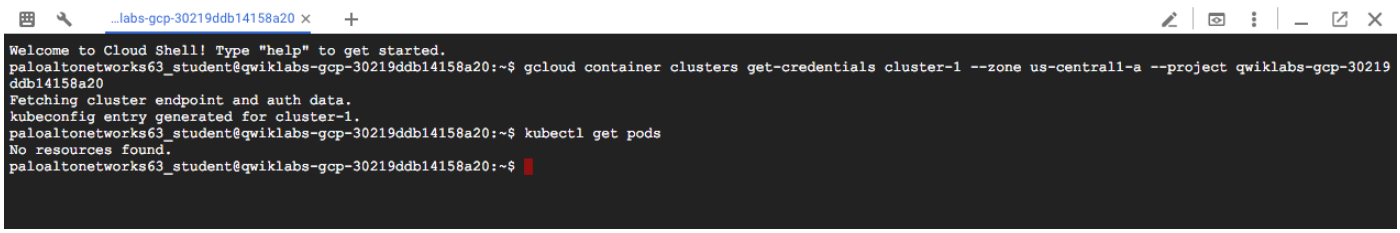
It may take a while to provision. Once provisioned press the Enter key to authenticate to the cluster. Afterwards the $ prompt will be present:



Let us explore some pods and services that have deployed.

Run this command in the cloud shell:

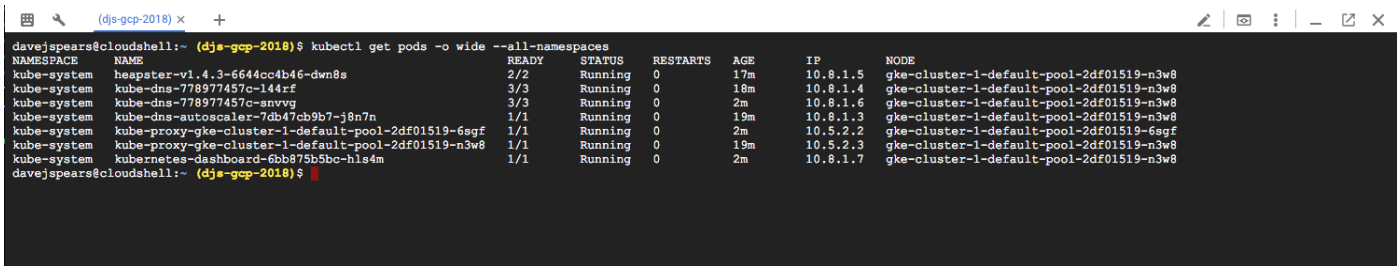
$ kubectl get pods



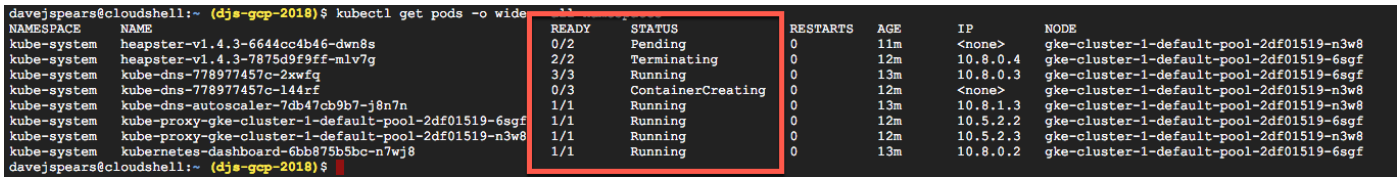
Since we have not deployed any resources this is normal.

Now let us see what system pods have been deployed. Run this command in the shell:

$ kubectl get pods --all-namespaces -o wide



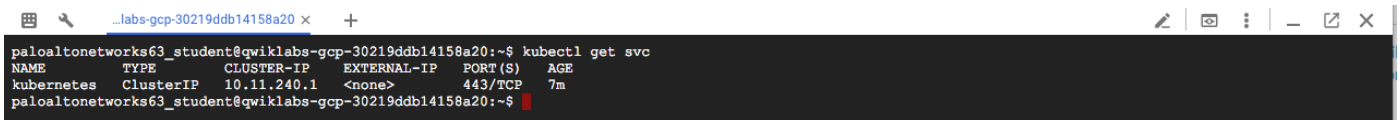
Note: If the output does not show all the pods in a running state, wait and rerun the **kubectl get pods --all-namespaces -o wide** command until they do. An example of this is state is in the following screen-print:



Now let us see what services have been deployed as part of the system:

Run the following in the shell:

$ kubectl get svc

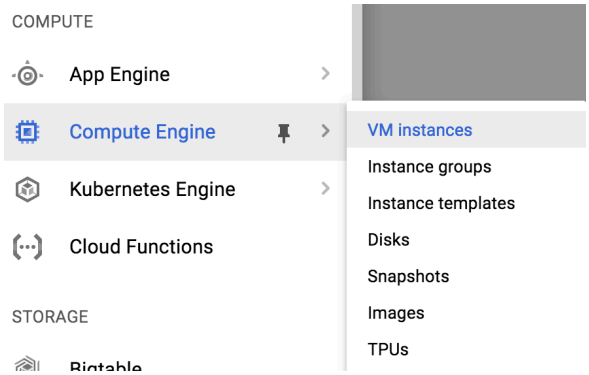


As you can see no services besides the system cluster have been deployed.

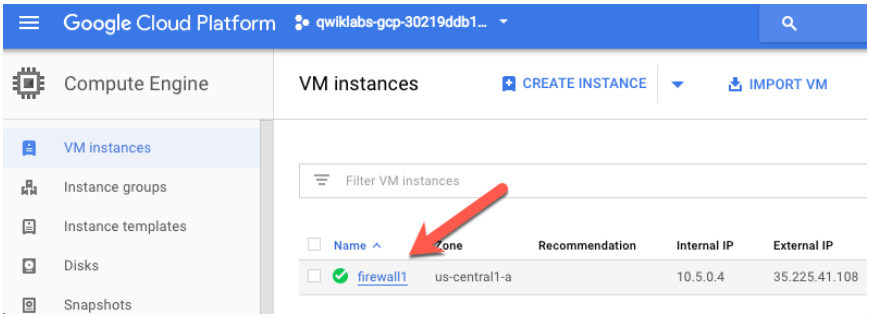
Task 3 – Log into the firewall

The VM-Series firewall deployed as part of the lab has been bootstrapped. Bootstrapping is a feature of the VM-Series firewall that allows you to load a pre-defined configuration into the firewall during boot-up. This ensures that the firewall is configured and ready at initial boot-up, thereby removing the need for manual configuration. The bootstrapping feature also enables automated deployment of the VM-Series.

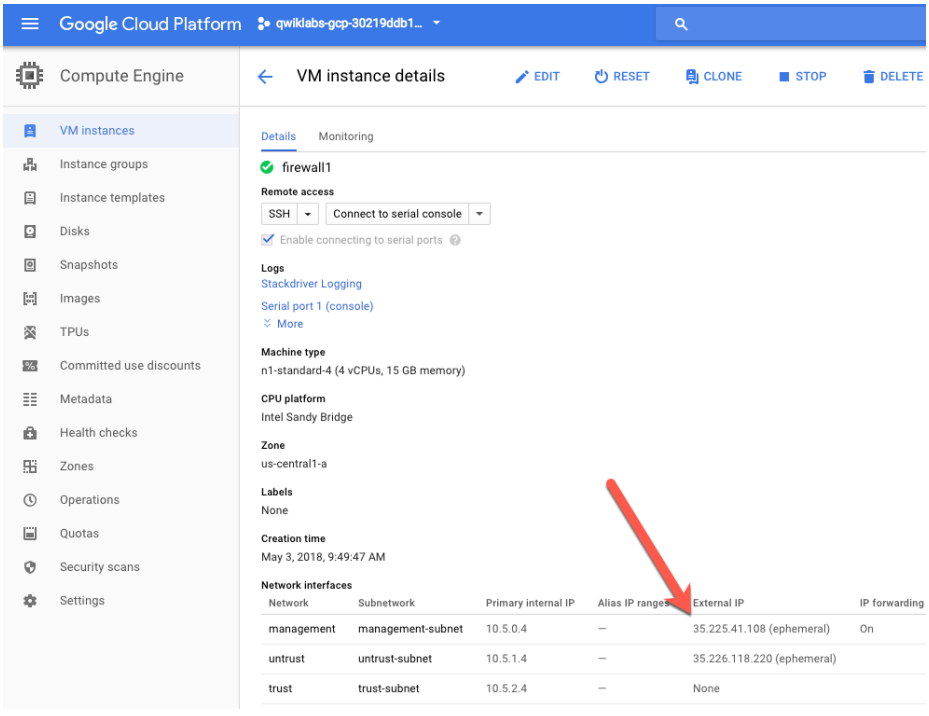
Navigate to Compute Instances > VM Instances



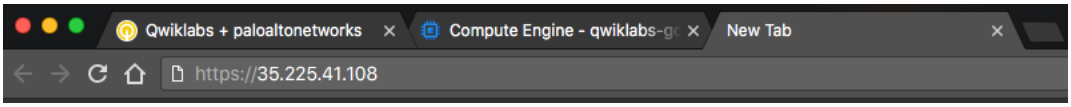
Click on firewall instance name to get more information and identify the management interface IP.



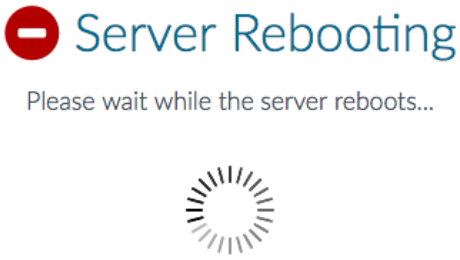
Copy the External public IP of the management interface



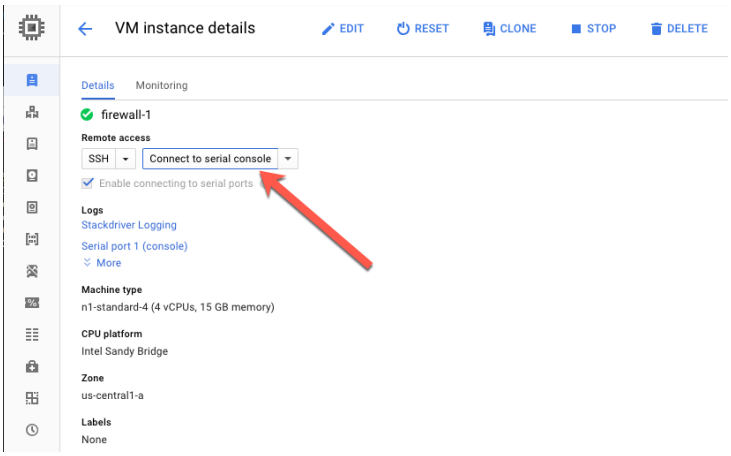
Open another browser tab and navigate to the firewall management interface:



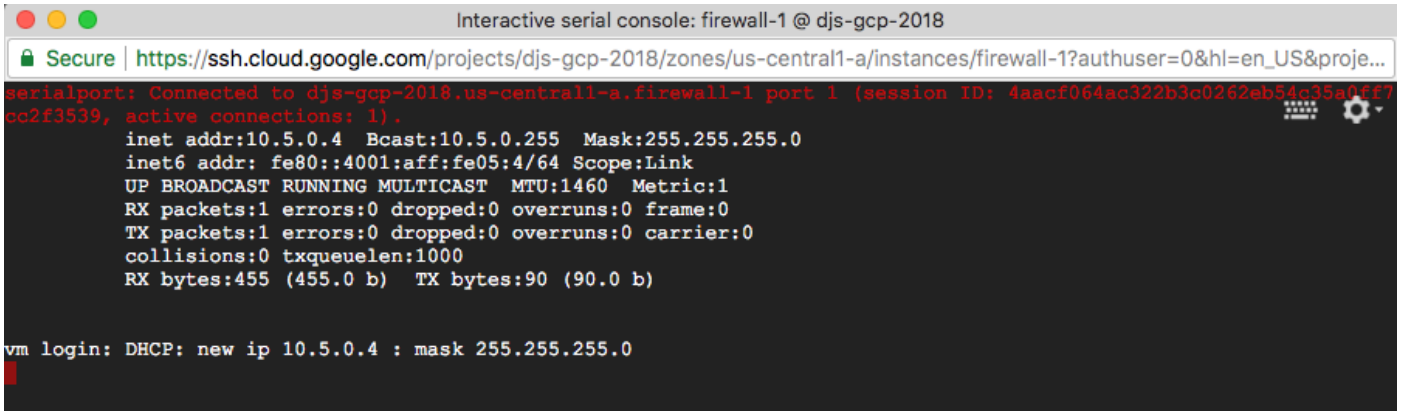
If you get a security exception, please ignore for this lab and proceed to the firewall login page. We are using a self-signed certificate which causes the exception. Depending on how quickly you do this, you might see the following message. It is normal and part of the bootup process:



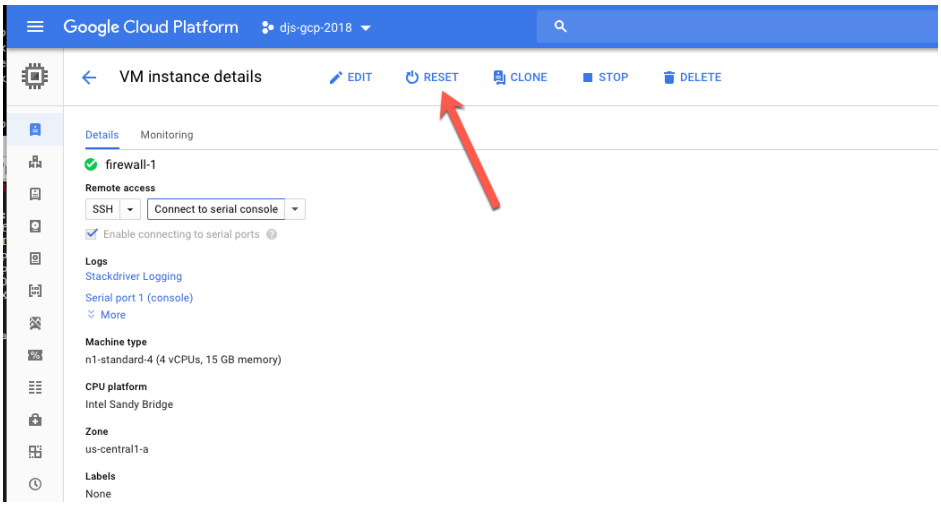
If the firewall does not seem available, it is possible to check the firewall via the GCP portal. You can click on the Connect to serial console to see if the firewall is up and running:



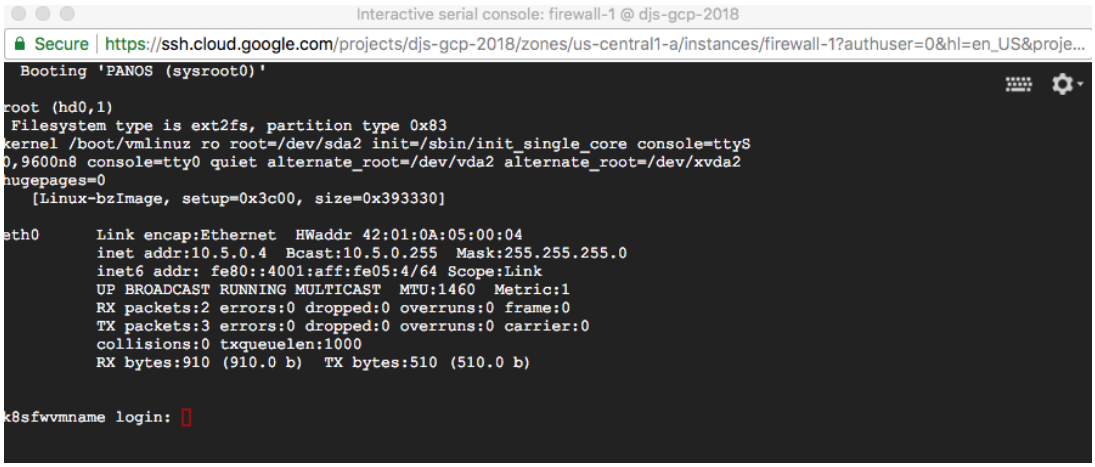
In some cases, the firewall may not be up and running yet. You can see in the prompt below the firewall is not all the way up.



You can wait for the VM to finish the bootup process or a quick reset of the VM seems to be a short fix:



After the reset the console shows the following prompt which is a good indication the firewall is booting up:



Also, If you wish to SSH into the FW, the SSH key defined in the terraform template can be used. This syntax is:

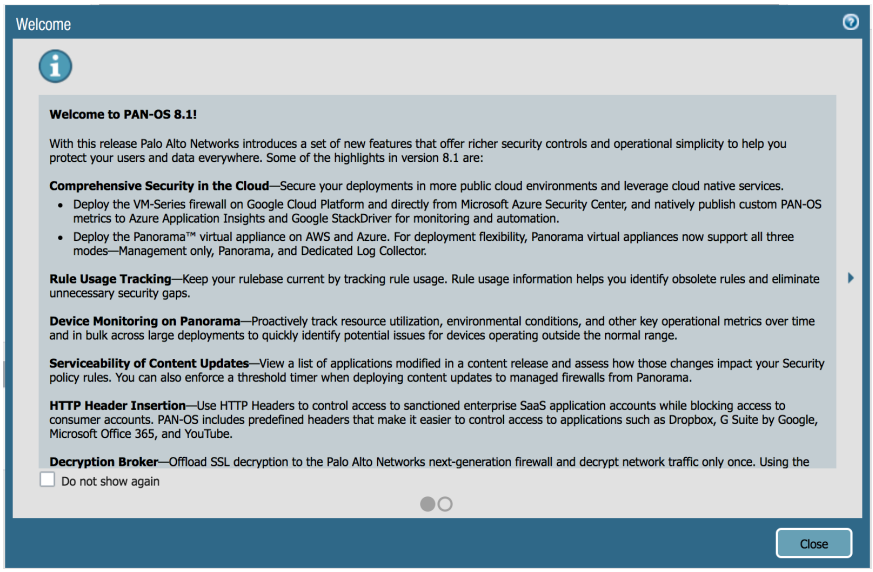
ssh -i <private key file> admin@<ip address of firewall>

When presented with the login screen you should be able to login to the firewall using (Hint: It's a good idea to jot this password down or save it to a notepad as you will regularly need it):

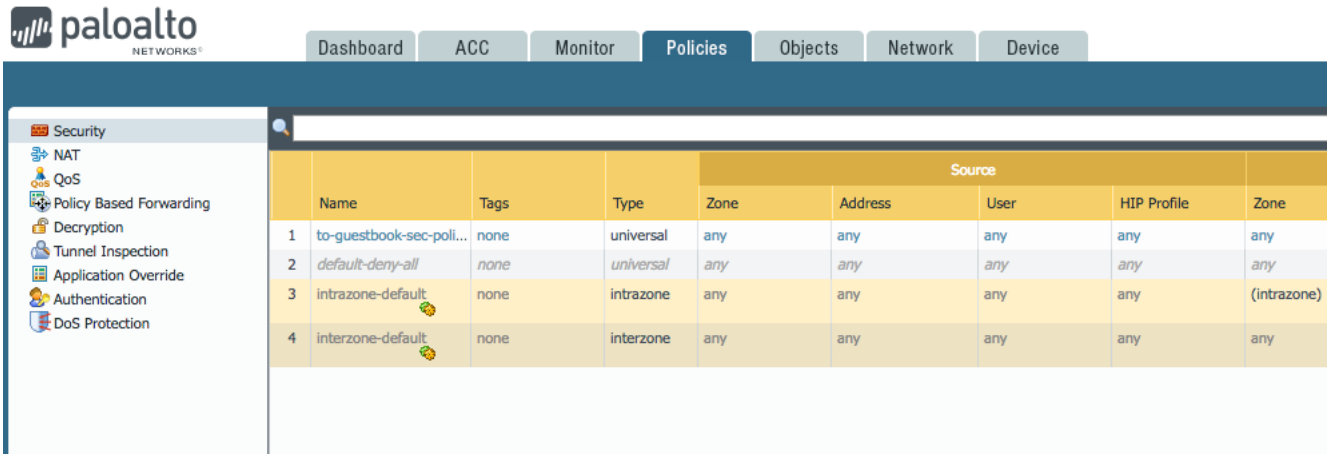
username: **admin**

password: **Pal0Alt0@123**

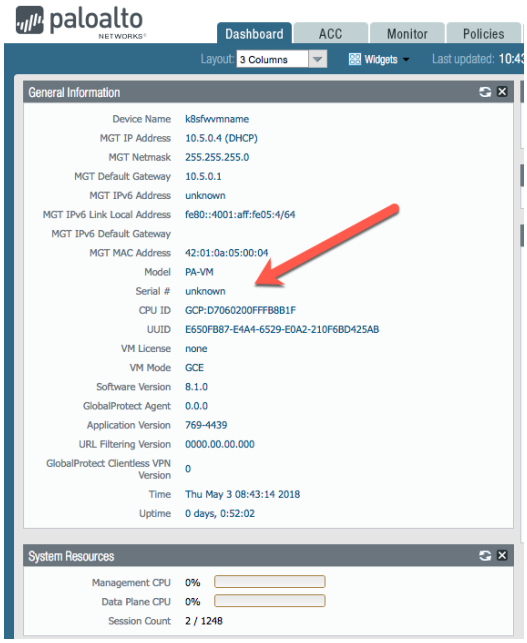
Once logged in you will see a welcome screen, dismiss the welcome dialog box by clicking Close.



Click the Policies tab and you will notice a predefined security policy which was imported using the bootstrapping feature. There are also some predefined NAT policies:



Click on the Dashboard tab, check to verify that the firewall has a serial number. The image defined in the terraform template is a PayGo bundle 1. This was used because a license will be required to view the logs later in the lab.



Launch a two tiered application

In this activity, you will:

Optionally: Explore the application's manifest file

Launch a two-tier application within your cluster

Explore aspects of the application

In this activity we will start using Kubernetes specific terms such as Pods, Services, etc.

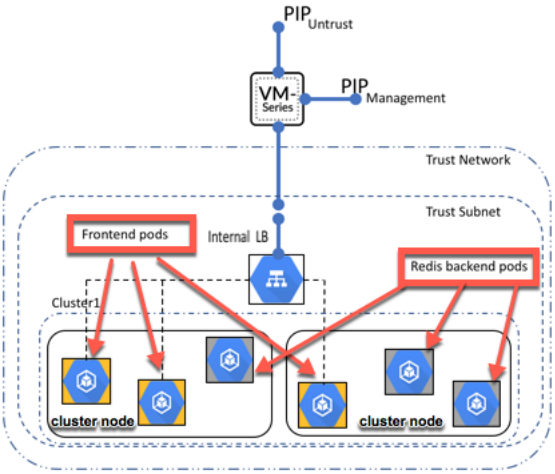
Here is a good primer: https://kubernetes.io/docs/concepts/workloads/pods/pod-overview/

Task 1 – Application Deployment YAML file

Guestbooks have been used by businesses for many years as a way to connect with customers and obtain contact information for future events and promotions. Today, businesses such as popular retail stores, 5-star hotels and even small family-owned B & B's are turning to iPad guestbook apps to help them gather information and enhance the customer's "in-biz" experience. Acquiring email addresses and a **social media** following is a crucial part of any marketing plan. With much of the population using computers on a

daily basis, an email marketing plan is of the utmost importance. Using a guest book app in your store makes collecting email addresses a snap and offers enticing features with which the traditional paper and pen guestbook just can't compete. The guestbook application we will build and secure today could be used for Hotel website visits, shopping sites or any other business that wants to keep track of their customer and provide them with promotions or advertisements.

This lab will deploy the following simple Guestbook application on the cluster nodes created during the Terraform template deployment:



As you can see this is a two-tiered application with Pods that are dedicated to front-end web services and backend DB services.

If interested, the following section dives a bit deeper into the templates being used to create this application. This is a link to the application manifest. Optionally, click the link below and open it in a browser of your choice.

https://github.com/PaloAltoNetworks/GCP-k8s-east-west-visibility/blob/master/alto-inspect.yaml

The manifest file declares various aspects of the application. For instance, it tells the orchestrator what type of resources you intend to deploy. In this case we will deploy a 2-tier simple redis application with a fronted and backend tier. The backend tier will consist of a redis-master and redis-slave for db redundancy. Front-end Service:

```
apiVersion: v1
kind: Service
metadata:
  name: frontend
  annotations:
    cloud.google.com/load-balancer-type: "Internal"
  labels:
    app: guestbook
    tier: frontend
spec:
  # if your cluster supports it, uncomment the following to automatically create
  # an external load-balanced IP for the frontend service.
  type: LoadBalancer
  ports:
  - port: 80
  selector:
    app: guestbook
    tier: frontend
---
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: frontend
spec:
  replicas: 3
  template:
    metadata:
      labels:
        app: guestbook
        tier: frontend
    spec:
      containers:
      - name: php-redis
        image: gcr.io/google-samples/gb-frontend:v4
        resources:
          requests:
            cpu: 100m
            memory: 100Mi
        env:
        - name: GET_HOSTS_FROM
          value: dns
          # If your cluster config does not include a dns service, then to
          # instead access environment variables to find service host
          # info, comment out the 'value: dns' line above, and uncomment the
          # line below:
          # value: env
        ports:
        - containerPort: 80
```

This tells the orchestrator that the service will be exposed via an internal load balancer

Redis-backend-master :

```yaml
apiVersion: v1
kind: Service
metadata:
  name: redis-master
  labels:
    app: redis
    tier: backend
    role: master
spec:
  #type: LoadBalancer
  ports:
  - port: 6379
    targetPort: 6379
  selector:
    app: redis
    tier: backend
    role: master
---
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: redis-master
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: redis
        role: master
        tier: backend
    spec:
      containers:
      - name: master
        image: gcr.io/google_containers/redis:e2e  # or just image: redis
        resources:
          requests:
            cpu: 100m
            memory: 100Mi
        ports:
        - containerPort: 6379
```

Redis-backend-slave:

```yaml
apiVersion: v1
kind: Service
metadata:
  name: redis-slave
  labels:
    app: redis
    tier: backend
    role: slave
spec:
  #type: LoadBalancer
  ports:
  - port: 6379
  selector:
    app: redis
    tier: backend
    role: slave
---
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: redis-slave
spec:
  replicas: 2
  template:
    metadata:
      labels:
        app: redis
        role: slave
        tier: backend
    spec:
      containers:
      - name: slave
        image: gcr.io/google_samples/gb-redisslave:v1
        resources:
          requests:
            cpu: 100m
            memory: 100Mi
        env:
        - name: GET_HOSTS_FROM
          value: dns
          # If your cluster config does not include a dns service, then to
          # instead access an environment variable to find the master
          # service's host, comment out the 'value: dns' line above, and
          # uncomment the line below:
          # value: env
        ports:
        - containerPort: 6379
---
```

Even though we have two tiers in our application, only one (the frontend service) is exposed to the outside world via a load balancer. The annotation listed above tells GCP and Kubernetes that the load balancer would be of type: Internal.

More on this later.

Task 2 – Launch the Application

Back in your cloud shell type the following command to deploy the application pods:

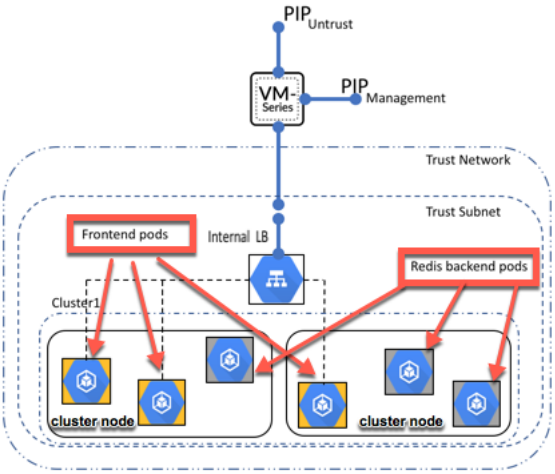
$ kubectl apply -f https://raw.githubusercontent.com/PaloAltoNetworks/GCP-k8s-east-west-visibility/master/guestbook-all-in-one.yaml

```
...labs-gcp-30219ddb14158a20 ×
paloaltonetworks63_student@qwiklabs-gcp-30219ddb14158a20:~$ kubectl create -f https://raw.githubusercontent.com/PaloAltoNetworks/ignite2018-how16/master/guestboo
k-all-in-one.yaml
service "redis-master" created
deployment "redis-master" created
service "redis-slave" created
deployment "redis-slave" created
service "frontend" created
deployment "frontend" created
paloaltonetworks63_student@qwiklabs-gcp-30219ddb14158a20:~$
```

You should see the services and deployments being created.

Task 3 – Explore what was just deployed

As shown previously, this is what has been built:



Let's validate this by list the new pods in your cluster. In your cloud shell:

$ kubectl get pods -o wide

You may see the status as Pending or ContainerCreating. This is usually a normal situation:

```
davejspears@cloudshell:~ (djs-gcp-2018)$ kubectl get pods -o wide
NAME                              READY     STATUS    RESTARTS   AGE       IP        NODE
frontend-685d7ff496-4kvqd         0/1       Pending   0          45s       <none>    <none>
frontend-685d7ff496-6zlmz         0/1       Pending   0          45s       <none>    <none>
frontend-685d7ff496-vxdrr         0/1       Pending   0          45s       <none>    <none>
redis-master-57cc594f67-qp6g5     0/1       Pending   0          52s       <none>    <none>
redis-slave-84845b8fd8-npz2r      0/1       Pending   0          49s       <none>    <none>
redis-slave-84845b8fd8-v5g81      0/1       Pending   0          49s       <none>    <none>
```

```
paloaltonetworks63_student@qwi...$ kubectl get pods -o wide
NAME                              READY     STATUS              RESTARTS   AGE       IP          NODE
frontend-685d7ff496-h2vwk         0/1       ContainerCreating   0          37s       <none>      gke-cluster-1-default-pool-b04e7f9e-ghbs
frontend-685d7ff496-tbbns         0/1       ContainerCreating   0          37s       <none>      gke-cluster-1-default-pool-b04e7f9e-ghbs
frontend-685d7ff496-z9g49         0/1       ContainerCreating   0          37s       <none>      gke-cluster-1-default-pool-b04e7f9e-s61t
redis-master-57cc594f67-7x4ws     0/1       ContainerCreating   0          37s       <none>      gke-cluster-1-default-pool-b04e7f9e-s61t
redis-slave-84845b8fd8-qdkr5      1/1       Running             0          37s       10.8.1.8    gke-cluster-1-default-pool-b04e7f9e-ghbs
redis-slave-84845b8fd8-rk4xk      0/1       ContainerCreating   0          37s       <none>      gke-cluster-1-default-pool-b04e7f9e-s61t
```

By executing the **kubectl get pods -o wide** again, you start seeing that the Ready and Status of pods change as they start up. Verify that all the pods get to created and running.

```
paloaltonetworks66_student@qwiklabs-gcp-0e34a12413067298:~$ kubectl get pods -o wide
NAME                              READY     STATUS    RESTARTS   AGE       IP          NODE
frontend-685d7ff496-c8fd7         1/1       Running   0          1m        10.8.1.9    gke-cluster-1-default-pool-168614d7-5b66
frontend-685d7ff496-njtw8         1/1       Running   0          1m        10.8.1.8    gke-cluster-1-default-pool-168614d7-5b66
frontend-685d7ff496-vfmrh         1/1       Running   0          1m        10.8.0.10   gke-cluster-1-default-pool-168614d7-bmlv
redis-master-57cc594f67-7g6hn     1/1       Running   0          1m        10.8.0.8    gke-cluster-1-default-pool-168614d7-bmlv
redis-slave-84845b8fd8-8n2fb      1/1       Running   0          1m        10.8.1.7    gke-cluster-1-default-pool-168614d7-5b66
redis-slave-84845b8fd8-qhddt      1/1       Running   0          1m        10.8.0.9    gke-cluster-1-default-pool-168614d7-bmlv
```

NOTE: In most cases the containers will move to a running status within 5min. If they stay in the Pending state for more than 5min, it is often quicker to delete the containers and redeploy them. To do this enter the following command:

$ kubectl delete -f https://raw.githubusercontent.com/PaloAltoNetworks/GCP-k8s-east-west-visibility/master/guestbook-all-in-one.yaml

and then redeploy the containers:

$ kubectl apply -f https://raw.githubusercontent.com/PaloAltoNetworks/GCP-k8s-east-west-visibility/master/guestbook-all-in-one.yaml

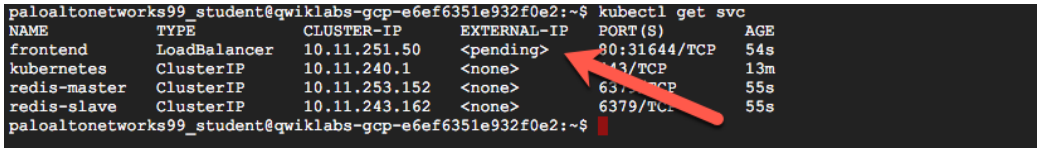
```
davejspears@cloudshell:~ (djs-gcp-2018)$ kubectl delete -f https://raw.githubusercontent.com/PaloAltoNetworks/GCP-k8s-east-west-visibility/master/guestbook-all-in-one.yaml
service "redis-master" deleted
deployment "redis-master" deleted
service "redis-slave" deleted
deployment "redis-slave" deleted
service "frontend" deleted
deployment "frontend" deleted
davejspears@cloudshell:~ (djs-gcp-2018)$ kubectl apply -f https://raw.githubusercontent.com/PaloAltoNetworks/GCP-k8s-east-west-visibility/master/guestbook-all-in-one.yaml
service "redis-master" created
deployment "redis-master" created
service "redis-slave" created
deployment "redis-slave" created
service "frontend" created
deployment "frontend" created
davejspears@cloudshell:~ (djs-gcp-2018)$
```

Next let's look at the load balancing service for the front-end pod. Execute the following command in the shell:

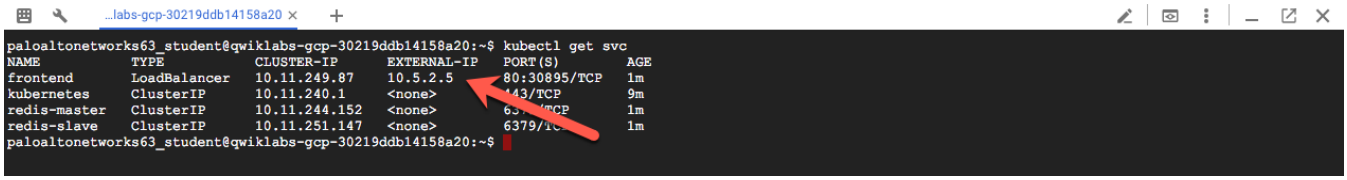
$ kubectl get svc

If you notice the "external-ip" of the frontend service's load balancer you will see that

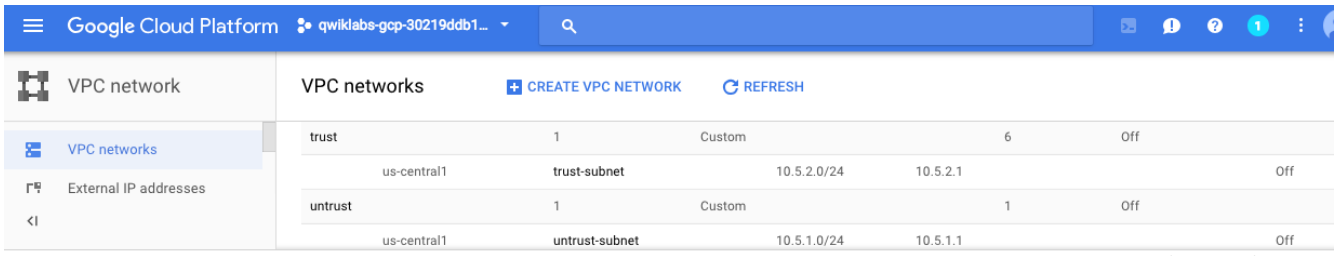
a) Initially this may show as pending:



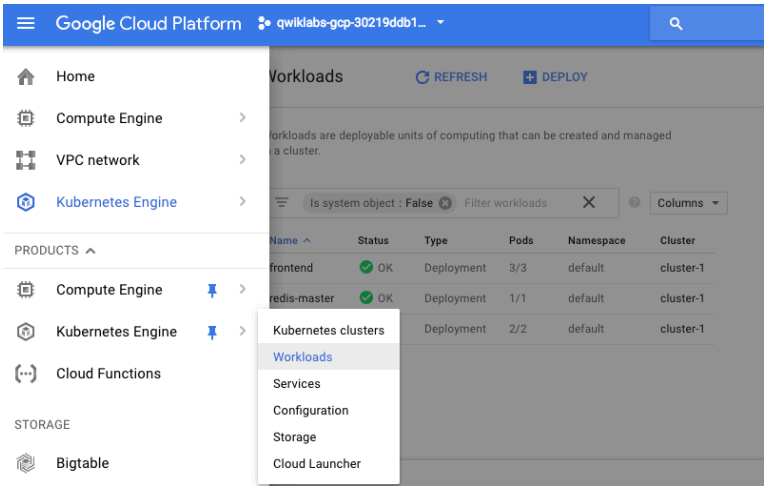
Repeating the "kubectl get svc" command will eventually show a private ip address for the internal load balancer:



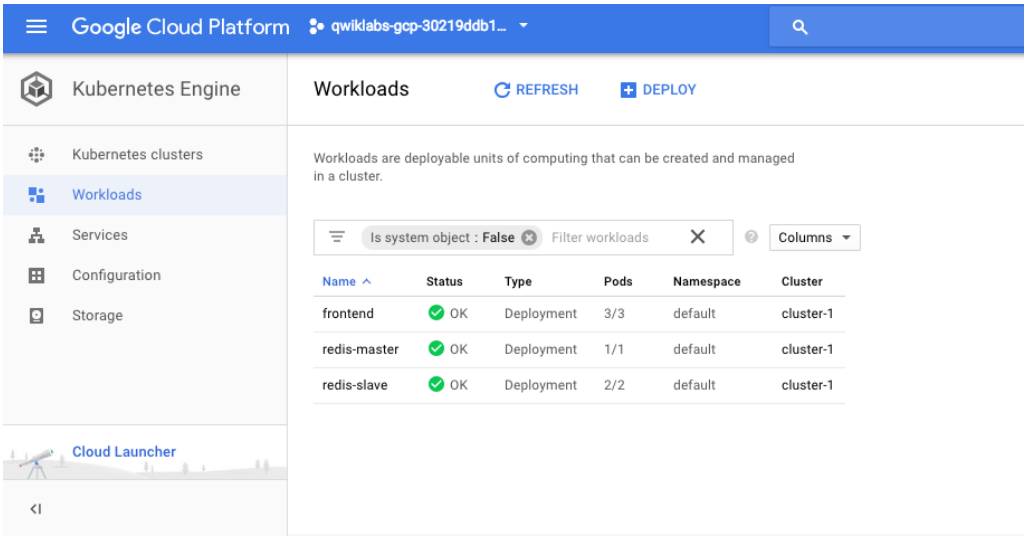
b) The private ip address is from the trust subnet's CIDR block



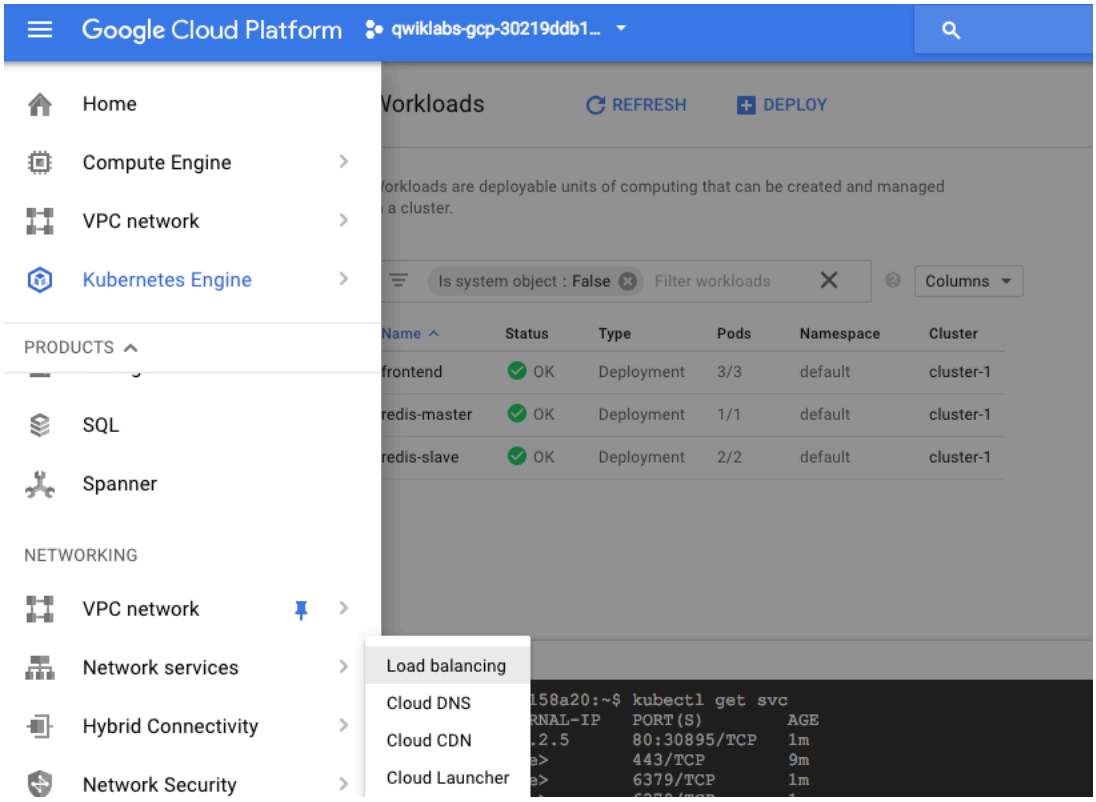
To see the new Kubernetes resources in the GCP console navigate to Kubernetes Engine > Workloads:



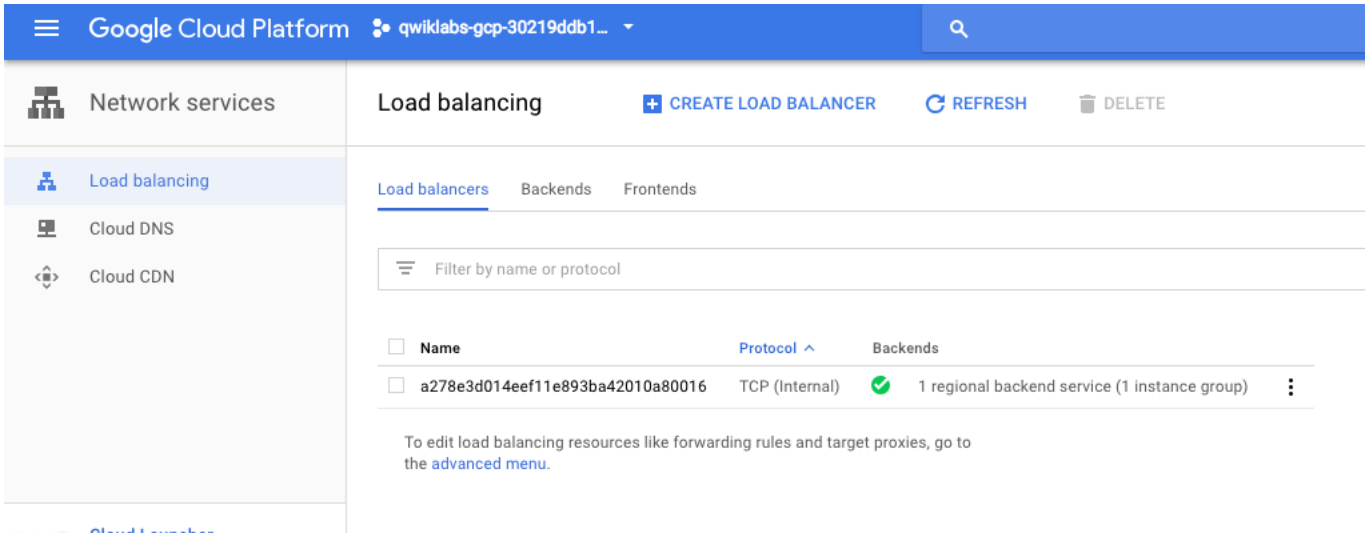
Here you can see the workloads that were just deployed. You can click on any of these to get more information:



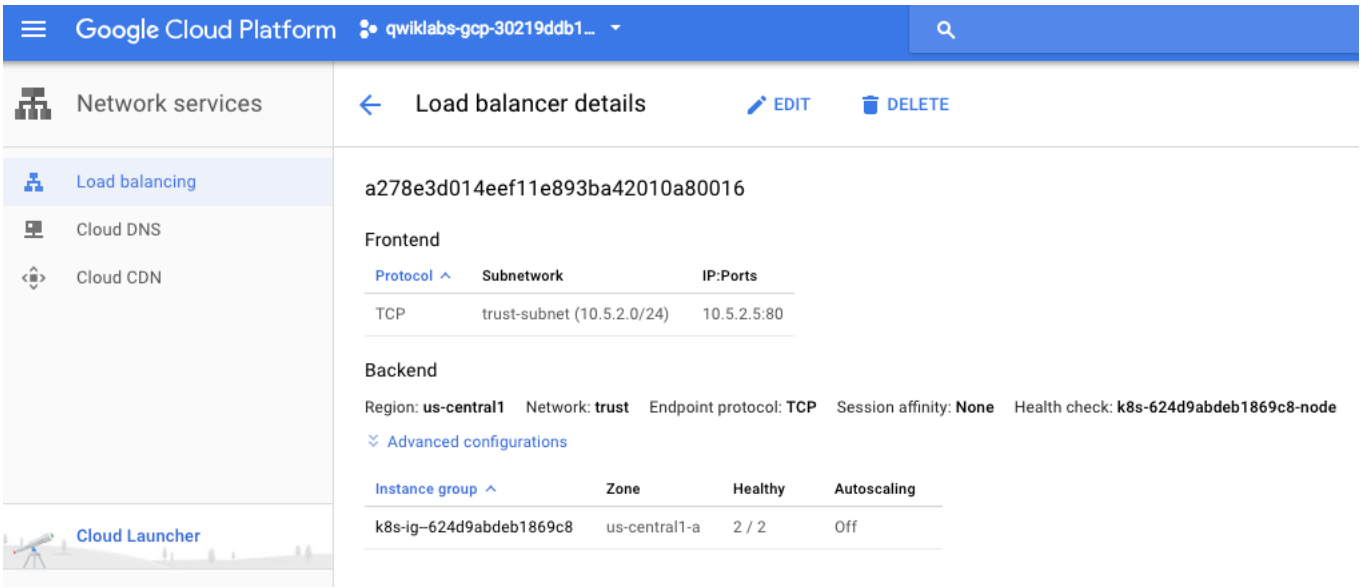
The Internal Load Balancer can be seen by navigating to Network Services > Load Balancing



Click on the load balancer to get more details:



This also shows that the internal load balancer is on the trust network and has an internal IP address that will be used later in the firewall configuration:



Securing Inbound Traffic

In this activity, you will:

Secure traffic that is inbound to your frontend service

Validate that traffic is visible in the Firewall logs

Task 1 – Note the Internal Load Balancer's IP Address

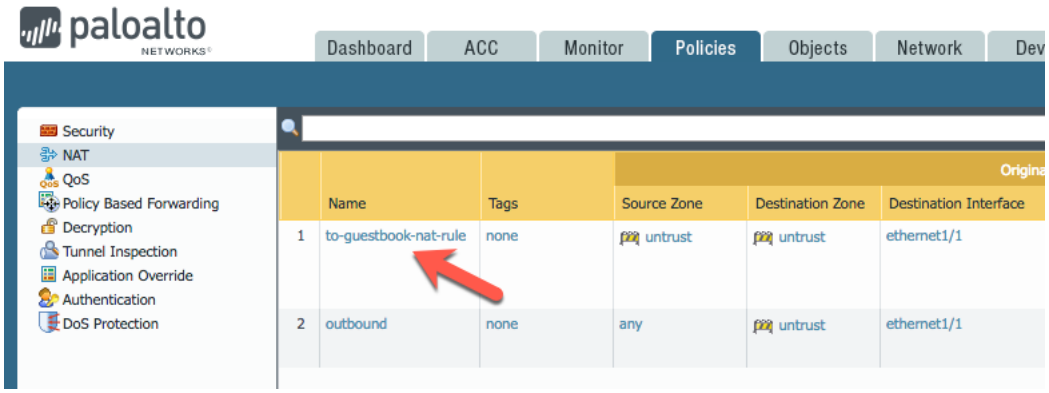
From the previous activity: make a note of the internal load balancer's ip address. (very probable it is 10.5.2.5)

Task 2 – Update the Firewall's NAT Policy

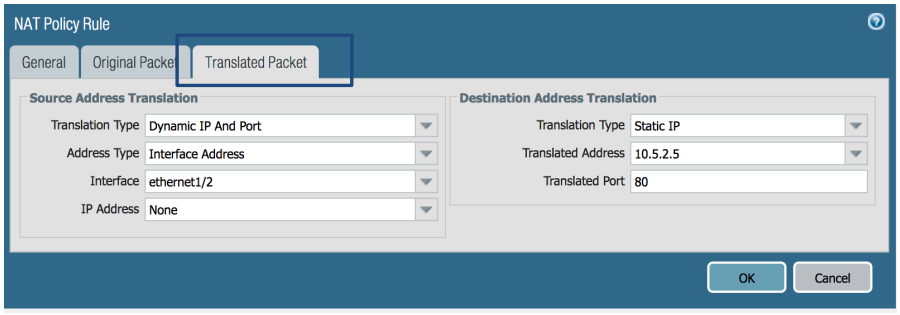
Login in to the VM-Series firewall using the management interface's ip address.

Hint: Check Activity 1, Task 2 if you don't have ip address jotted down

Click the Policies Tab and navigate to NAT on the left. Click on the to-guestbook-nat-rule to edit it:

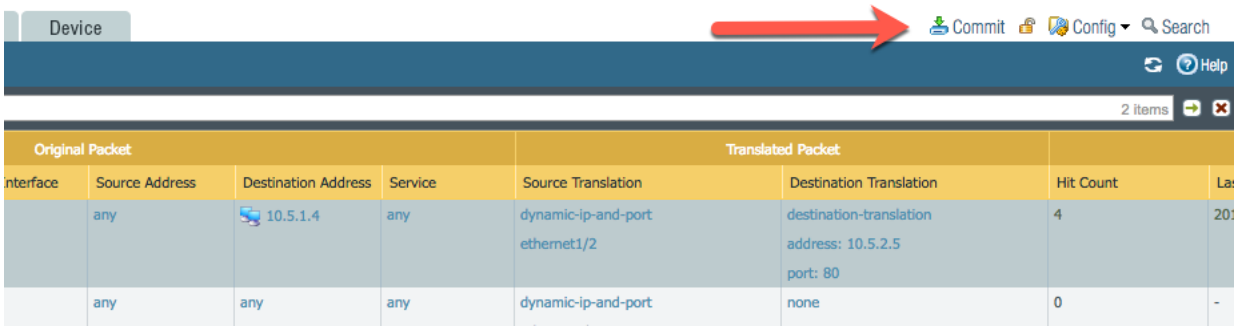


Click on the Translated Packet tab:

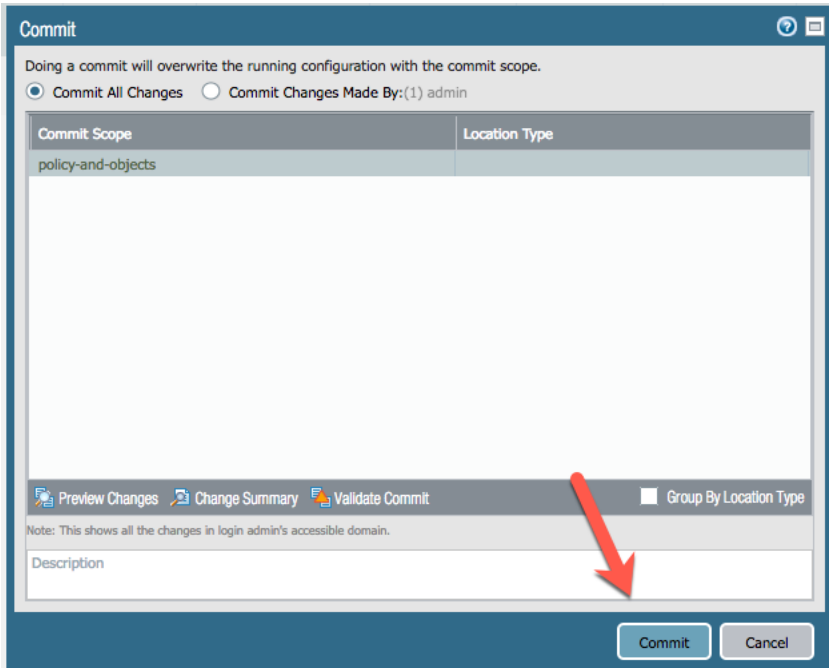


In the Translated Address field, enter the load balancer's ip address from Task 1 and click OK. It might not need to be changed.

If a change was needed, click the commit link on the top right



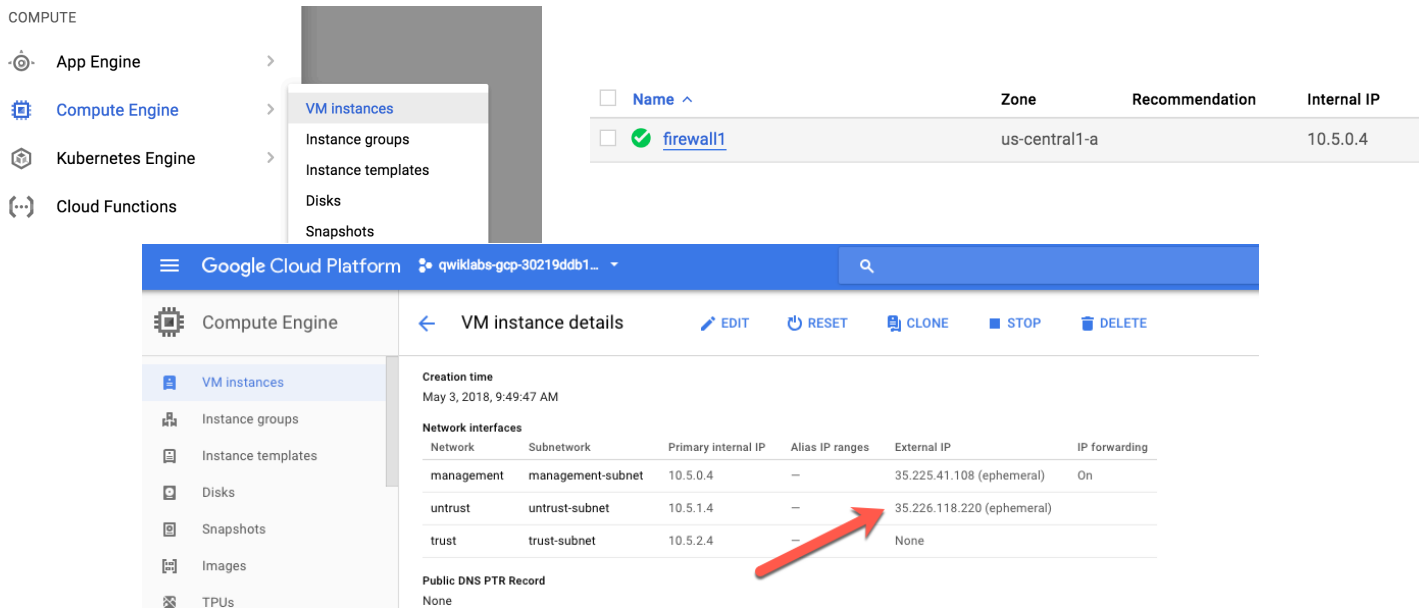
When the next dialog box opens, click the commit button



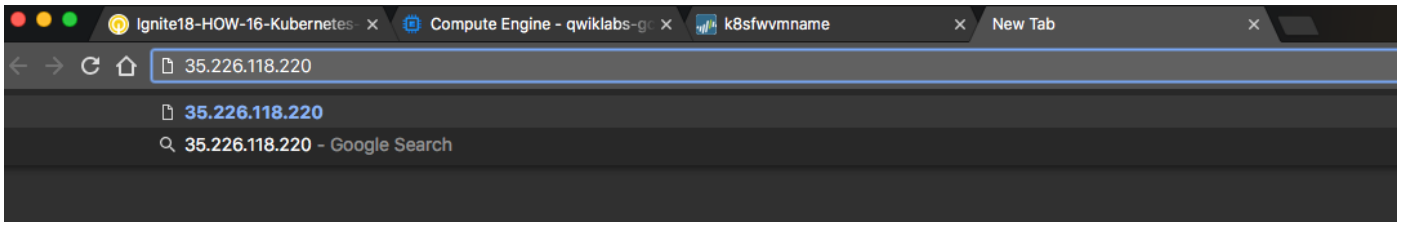
Task 3 – Connect to the Guestbook Frontend

The VM-Series is now protecting your Kubernetes workload. In order to connect to the guestbook's frontend service, you will connect to the untrust ip-address of the firewall. Navigate to the Compute Engine> VM Instances screen.

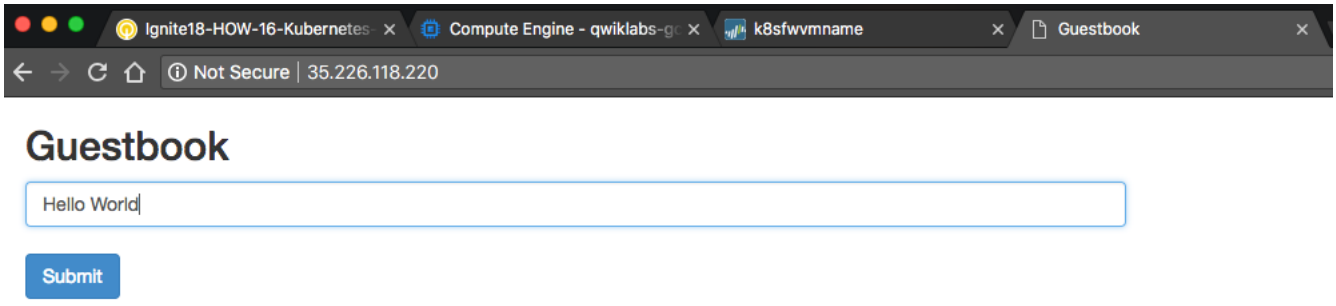
Then click on the firewall and copy the External IP address of the untrust interface:



In your browser navigate to the untrust-ip address of the firewall

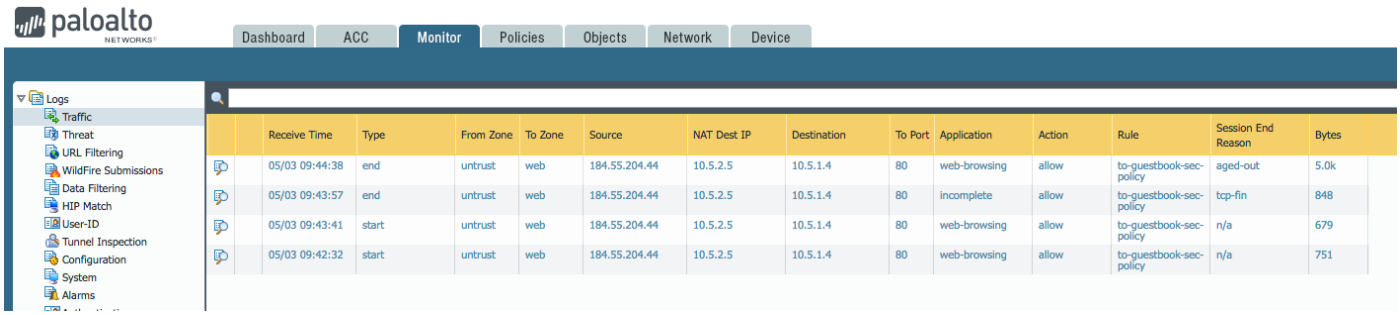


And you should be greeted with the frontend of your guestbook application:



Type a message in the dialog box and click Submit. The message should be echoed on the website.

Open the firewall interface and navigate to the monitor tab. I have added the NAT Dest IP Column so it is possible to see the address translation to the internal load balancer IP address.



Securing Outbound Traffic

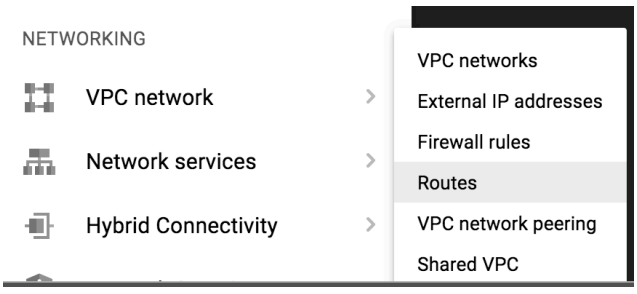
In this activity, you will:

Secure outbound traffic from the cluster

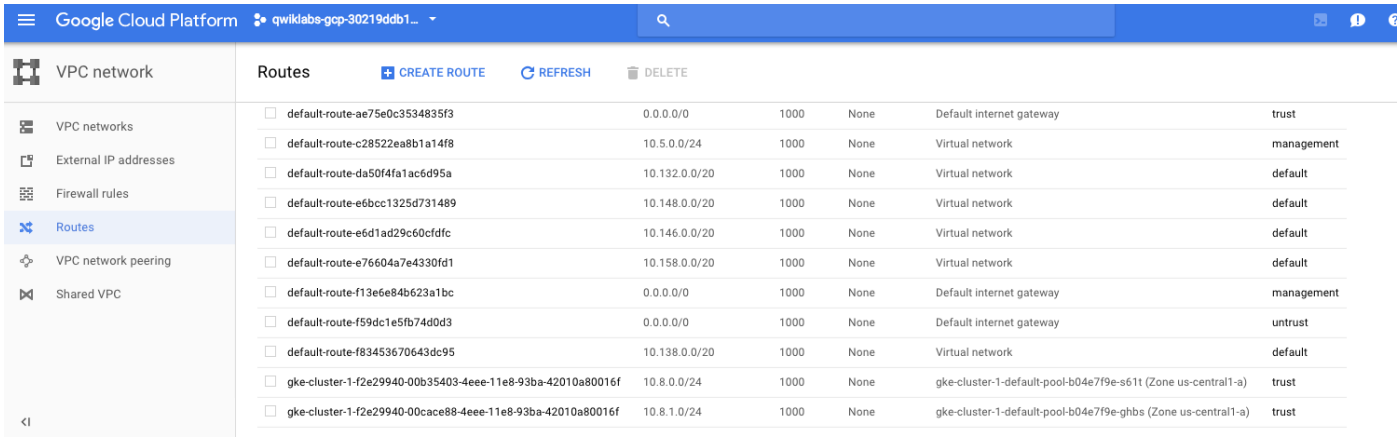
Validate traffic is in the Firewall logs

Task 1 – Add Outbound Route

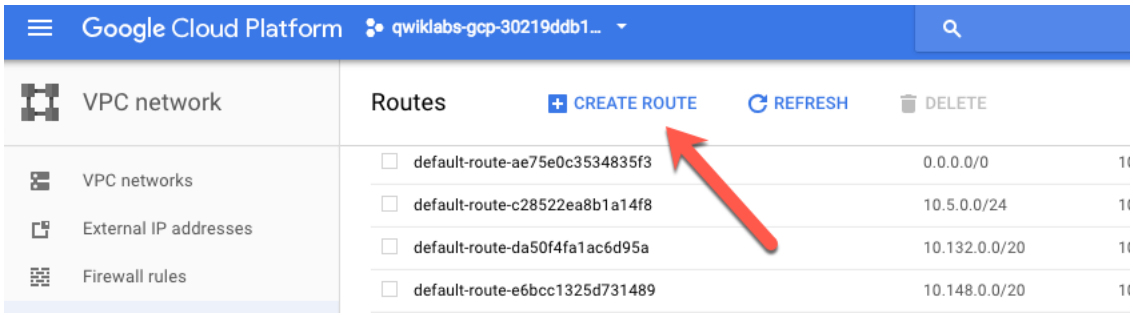
To secure any traffic that is originating from within the cluster we need to add a couple of routes in the GCP console. Go to VPC Network > Routes



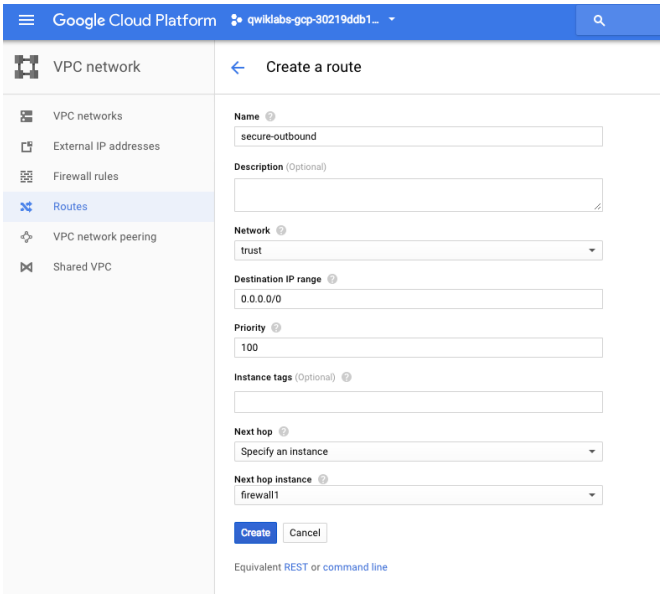
You will see a few default routes added by Kubernetes during the initial template launch:



Click on Create Route:



Create a route with the following Parameters:

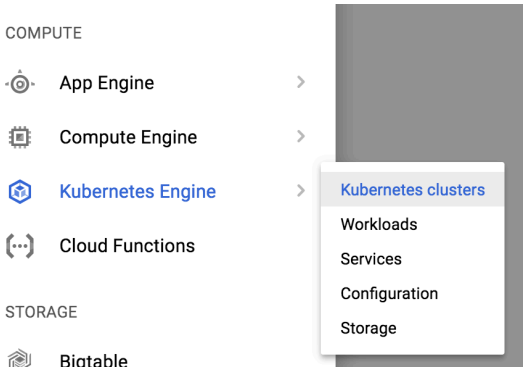


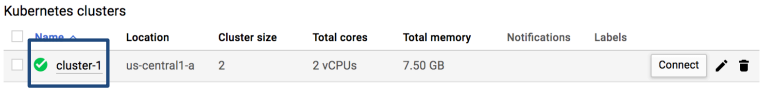
Name: secure-outbound
Network: trust
Destination IP Range: 0.0.0.0/0
Priority: 100
Next hop: Specify an instance
Next Hop Instance: firewall1

And then click Create

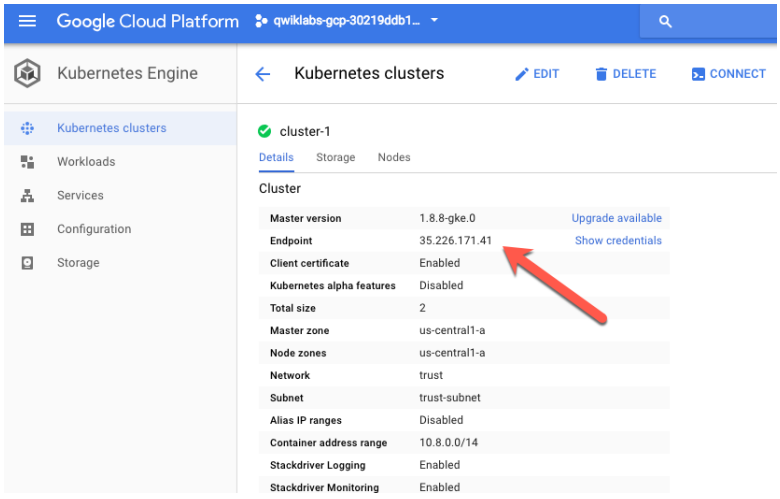
Task 2 – Add Kube-Apiserver route

Navigate to Kubernetes Engine > Kubernetes clusters and click on the cluster.

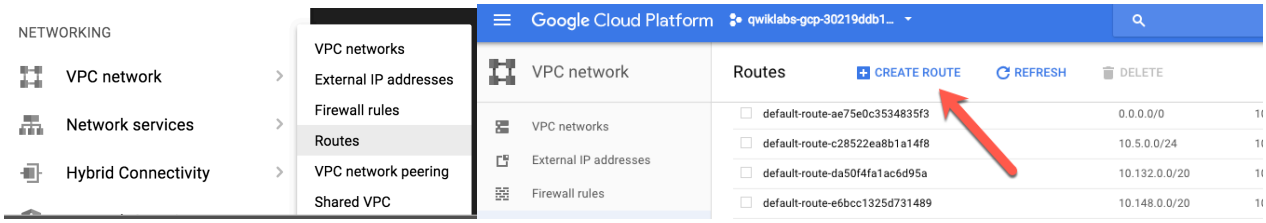




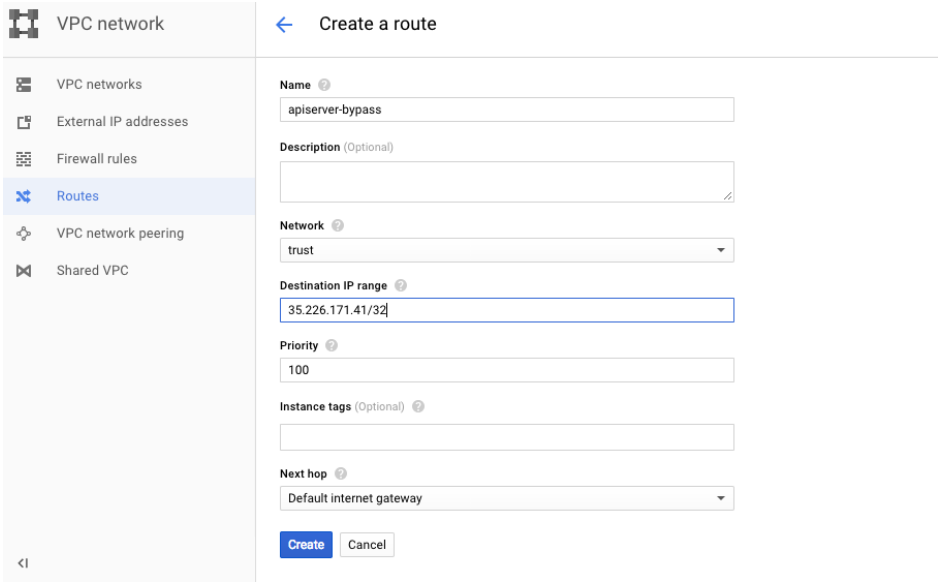
When the cluster details page opens, identify the kube-apiserver ip address. Labeled as Endpoint in the page. Copy this address:



Navigate back to VPC Networks > Routes and create another route that will allow the kube-apiserver to perform health checks without the firewall getting in the way:

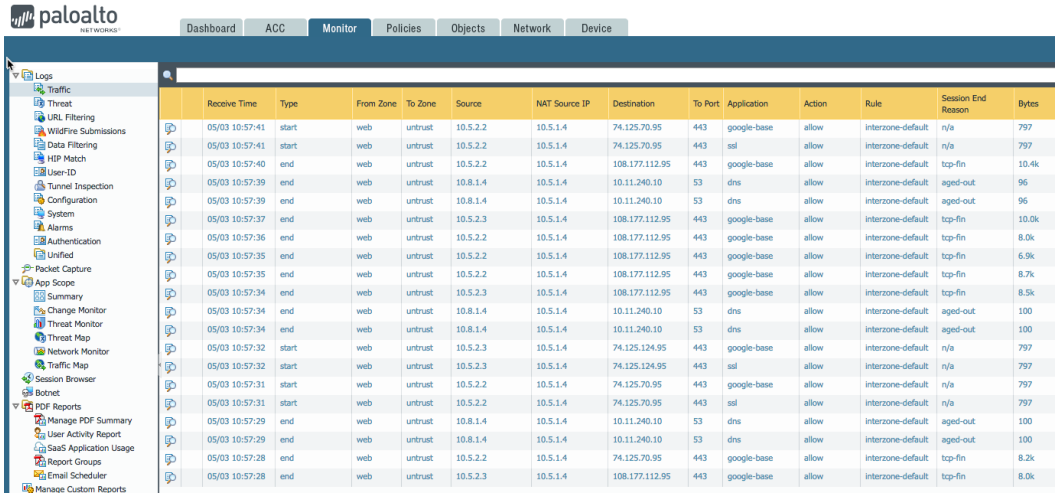


Create a route with the following parameters:

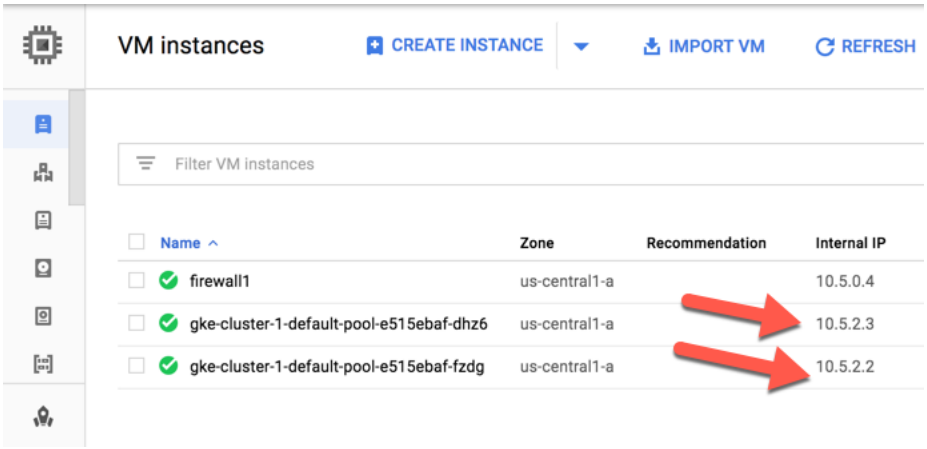


Name: apiserver-bypass
Network: trust
Destination IP Range: [Insert kube-apiserver IP] /32
Priority: 100
Next hop: Default internet gateway
And then click Create.

Navigate back to the firewall monitor tab and you can now see outbound traffic as well from the cluster nodes.



These source addresses are the instance addresses of the Kubernetes cluster servers:



Inspection of Inter-Pod traffic

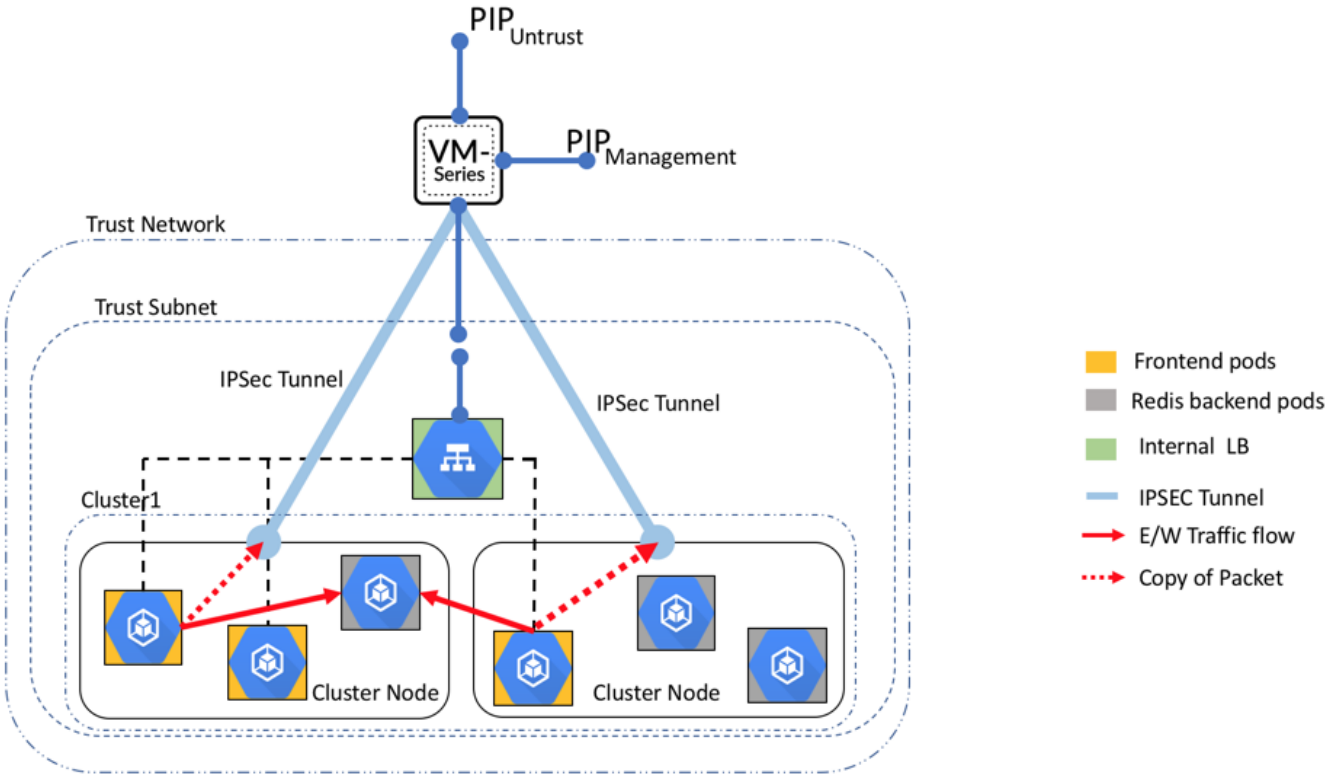
In this activity, you will:

Configure the k8s cluster to be able to inspect Pod-to-Pod traffic by the firewall

Validate that traffic is visible in the Firewall logs

One of the key challenges when deploying microservices is the lack of visibility of traffic that is flowing between services (or pods) within the cluster.

In this activity you will configure your Kubernetes nodes to redirect a copy of the packet to the VM-Series, which will provide visibility into the traffic between pods and services within your cluster. Task 1 will deploy a DaemonSet which will send a copy of the intra pod traffic between the frontend and backend pods over an IPSec tunnel to the PANW firewall for inspection. The following drawing provides a visual of this concept:



Task 1 – Deploy a DaemonSet

A DaemonSet is a Kubernetes resource that ensures that a copy of a pod is run on all (or some) of your nodes

https://kubernetes.io/docs/concepts/workloads/controllers/daemonset/

In your cloud Shell, run the following command:

$ kubectl apply -f https://raw.githubusercontent.com/PaloAltoNetworks/GCP-k8s-east-west-visibility/master/alto-inspect.yaml

NOTE: The following link can be used as the WWCE is a private repo.

$ kubectl apply -f https://raw.githubusercontent.com/djspears/GCP-K8s/master/alto-inspect.yaml

```
paloaltonetworks122_student@qwiklabs-gcp-250f11d5b7b9ef75:~$ kubectl apply -f https://raw.githubusercontent.com/PaloAltoNetworks/ignite2018-how16/master/alto-inspect.yaml
daemonset "alto-inspect" created
paloaltonetworks122_student@qwiklabs-gcp-250f11d5b7b9ef75:~$
```

You should see “daemonset alto-inspect” created in the shell. To verify that the pods were launched you can list all pods in all namespaces:

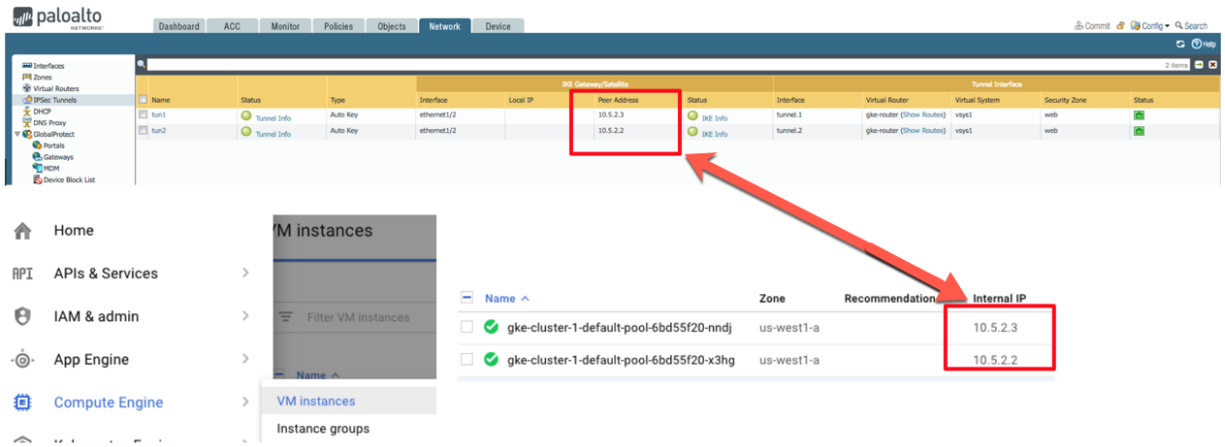
$ kubectl get pods --all-namespaces -o wide

```
NAMESPACE     NAME                                                READY   STATUS    RESTARTS   AGE   IP           NODE
default       frontend-685d7ff496-gfkd9                           1/1     Running   0          3m    10.28.0.9    gke-cluster-1-default-pool-6bd55f20-nndj
default       frontend-685d7ff496-kpf9g                           1/1     Running   0          3m    10.28.0.10   gke-cluster-1-default-pool-6bd55f20-nndj
default       frontend-685d7ff496-n8lzn                           1/1     Running   0          3m    10.28.1.9    gke-cluster-1-default-pool-6bd55f20-x3hg
default       redis-master-57cc594f67-gbvzq                       1/1     Running   0          4m    10.28.1.7    gke-cluster-1-default-pool-6bd55f20-x3hg
default       redis-slave-84845b8fd8-7frjt                        1/1     Running   0          4m    10.28.1.8    gke-cluster-1-default-pool-6bd55f20-x3hg
default       redis-slave-84845b8fd8-fztmf                        1/1     Running   0          4m    10.28.0.8    gke-cluster-1-default-pool-6bd55f20-nndj
kube-system   alto-inspect-g2z4f                                  1/1     Running   0          3m    10.5.2.3     gke-cluster-1-default-pool-6bd55f20-nndj
kube-system   alto-inspect-xqcnt                                  1/1     Running   0          3m    10.5.2.2     gke-cluster-1-default-pool-6bd55f20-x3hg
kube-system   event-exporter-v0.1.7-5c4d9556cf-b124j              2/2     Running   0          14m   10.28.0.4    gke-cluster-1-default-pool-6bd55f20-nndj
kube-system   fluentd-gcp-v2.0.9-kq42h                            2/2     Running   0          13m   10.28.0.2    gke-cluster-1-default-pool-6bd55f20-nndj
kube-system   fluentd-gcp-v2.0.9-x57pb                            2/2     Running   0          13m   10.28.1.2    gke-cluster-1-default-pool-6bd55f20-x3hg
kube-system   heapster-v1.4.3-699fc4bd5b-2d9sq                    3/3     Running   0          12m   10.28.0.7    gke-cluster-1-default-pool-6bd55f20-nndj
kube-system   kube-dns-778977457c-qd79d                           3/3     Running   0          14m   10.28.1.4    gke-cluster-1-default-pool-6bd55f20-x3hg
kube-system   kube-dns-778977457c-w9cv7                           3/3     Running   0          12m   10.28.0.6    gke-cluster-1-default-pool-6bd55f20-nndj
kube-system   kube-dns-autoscaler-7db47cb9b7-sqjmv                1/1     Running   0          14m   10.28.0.5    gke-cluster-1-default-pool-6bd55f20-nndj
kube-system   kube-proxy-gke-cluster-1-default-pool-6bd55f20-nndj  1/1     Running   0          13m   10.5.2.3     gke-cluster-1-default-pool-6bd55f20-nndj
kube-system   kube-proxy-gke-cluster-1-default-pool-6bd55f20-x3hg  1/1     Running   0          13m   10.5.2.2     gke-cluster-1-default-pool-6bd55f20-x3hg
kube-system   kubernetes-dashboard-768854d6dc-cm88g               1/1     Running   0          14m   10.28.1.5    gke-cluster-1-default-pool-6bd55f20-x3hg
kube-system   l7-default-backend-6497bcdb4d-zvq78                 1/1     Running   0          14m   10.28.1.3    gke-cluster-1-default-pool-6bd55f20-x3hg
```

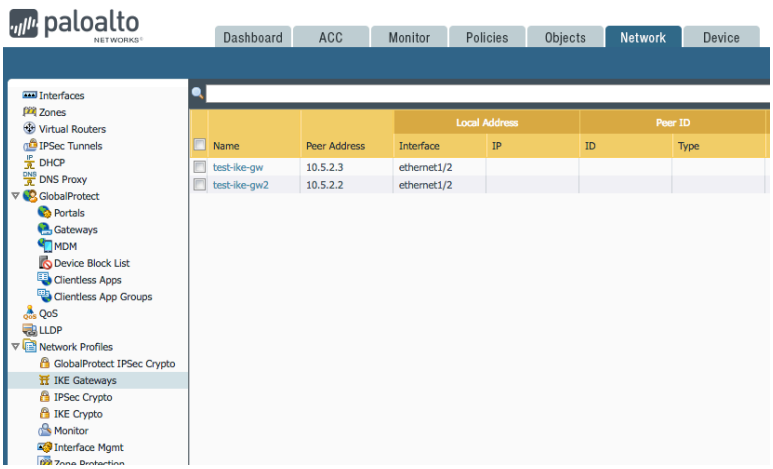
By deploying the alto-inspect DaemonSet, we have now established an IPSec tunnel between each of the nodes in the cluster to the trust interface of the firewall.

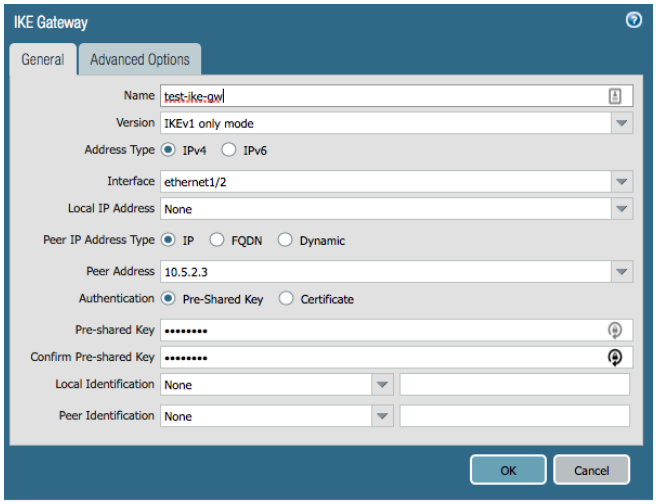
In order to ensure that the tunnels are up, log into the firewall an Navigate to Network > IPSec Tunnels. If the status is green, the tunnels are up.

Verify that the Peer Address matches the private ip addresses of your Kubernetes nodes



If not, in the firewall, go to Network> IKE Gateways and modify the test-ike-gw and test-ike-gw2 peer addresses to match the internal ip addresses of the nodes. It doesn't matter which gateway gets which peer ip address.

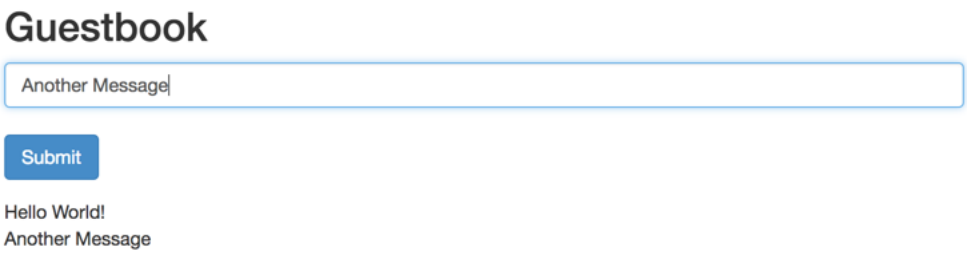


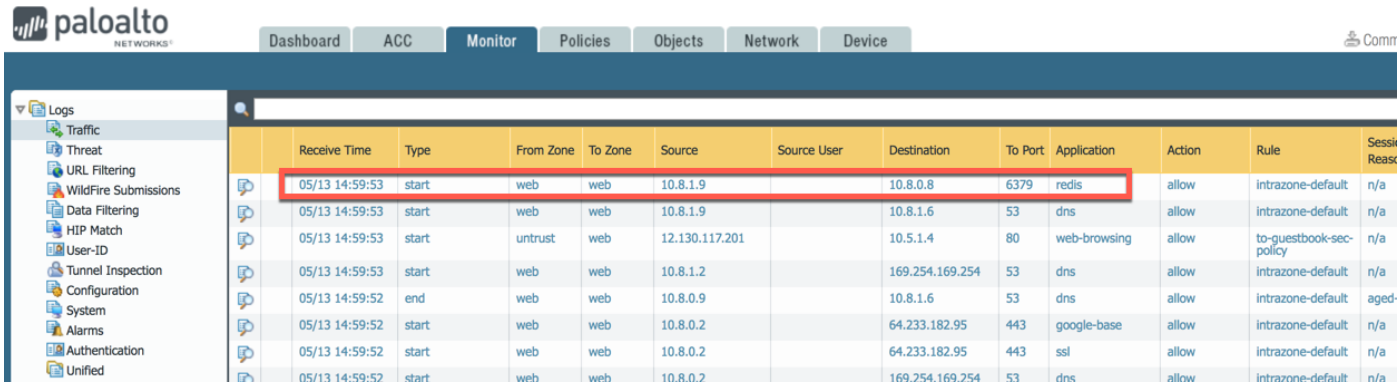


IF any changes were needed, be sure to Commit the changes.

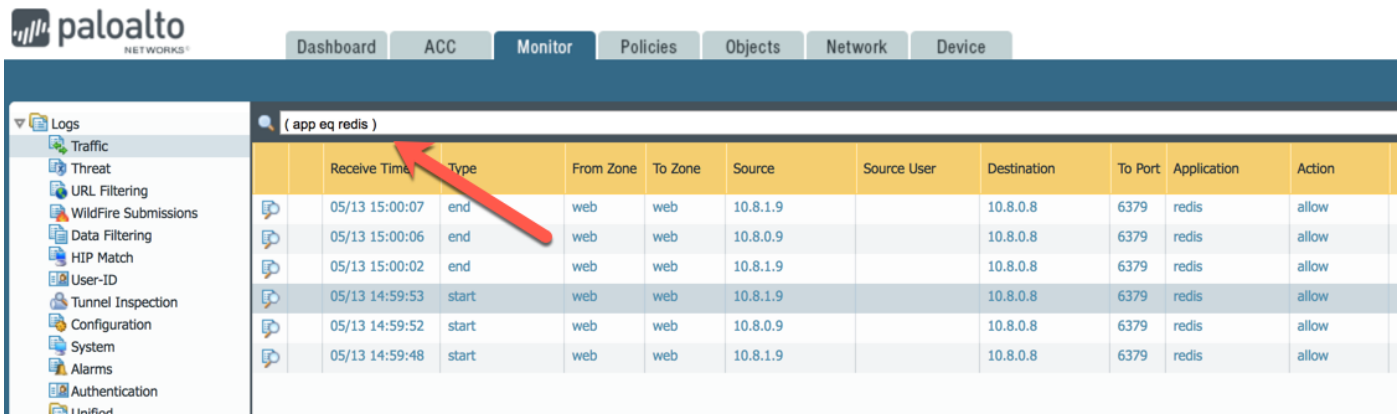
Task 2 – Verify firewall logs to see inter-pod traffic

Go to guestbook and submit another message. You should see the traffic between the pods.





If there is a lot of traffic that is making it difficult to see the intra pod traffic, it is possible to filter by application:

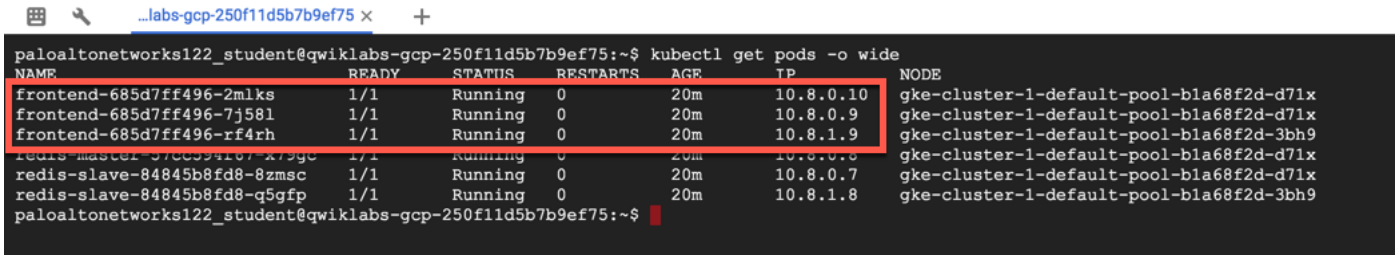


Take note of the Source and Destination addresses of the redis application traffic.

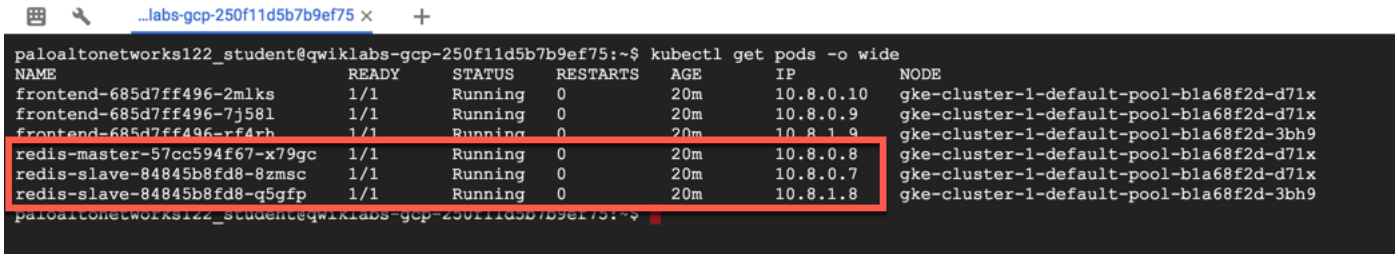
Now navigate back to the cluster command shell and run the following command:

$ kubectl get pods -o wide

Notice that the Source Addresses from the redis traffic in the monitor tab corresponds to the frontend pods:



And the destination address corresponds to the redis-backend:



End of Activity

Conclusion

Congratulations! You have now successfully integrated the PANW VM series firewall to gain visibility into North/South traffic entering and leaving the Kubernetes cluster and E/W traffic between a Guestbook Application Frontend and Backend pods.