Set Up and Configure a Cloud Environment in Google Cloud

* Learn multiple ways to deploy and monitor applications including how to: explore IAM rols and add/remove project access, create VPC networks, deploy and monitor Compute Engine VMs, write SQL queries, deploy and monitor VMs in Compute Engine, and deploy applications using Kubernetes with multiple deployment approaches.
* 1. Cloud IAM: Qwik Start
  + GC IAM unifies access control for Cloud Platform services into a single system to present a consistent set of operations. Watch the short video Manage Access Control with Google Cloud IAM.
* 2. Introduction to SQL for BigQuery and Cloud SQL
  + In this lab you will learn fundamental SQL clauses and will get hands on practice running structured queries on BigQuery and Cloud SQL.
* 3. Multiple VPC Networks
  + In this lab, you create several VPC networks and VM instances and test connectivity across networks.
* 4. Cloud Monitoring: Qwik Start
  + This lab shows you how to monitor a Google Compute Engine virtual machine (VM) instance with Cloud Monitoring. Watch the short videos Monitor Health of All Your Cloud Apps with Google Cloud monitoring and Monitor a VM Instance with Cloud monitoring, GCP Essentials.
* 5. Managing Deployments Using Kubernetes Engine
  + Dev Ops best practices make use of multiple deployments to manage application deployment scenarios. This lab provides practice in scaling and managing containers to accomplish common scenarios where multiple heterogeneous deployments are used.
* 6. Set Up and Configure a Cloud Environment in Google Cloud: Challenge Lab
  + This challenge lab tests your skills and knowledge from the labs in the Kubernetes in Google Cloud quest. You should be familiar with the content of the labs before attempting this lab.

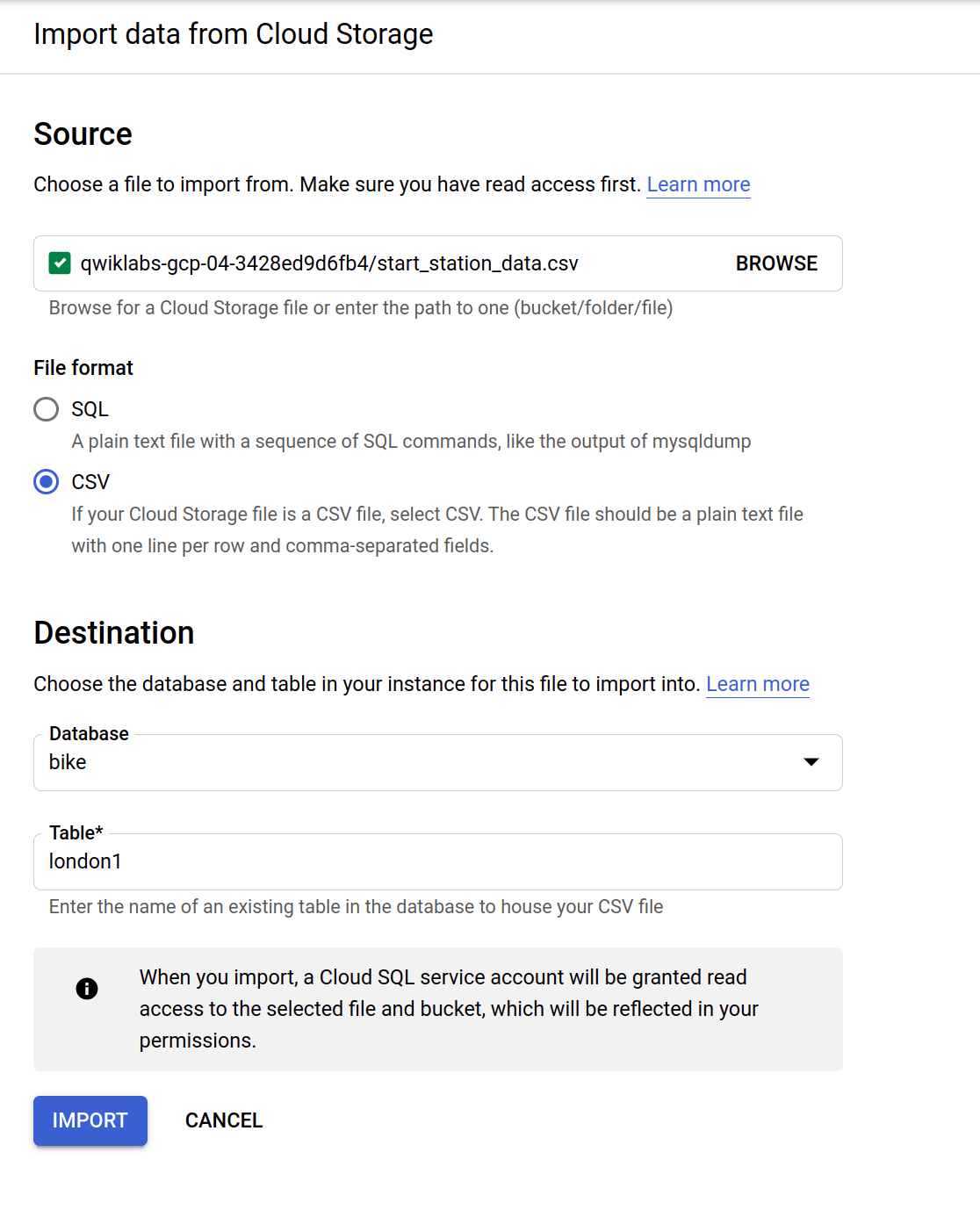
## 2. Introduction to SQL for BigQuery and Cloud SQL - https://www.cloudskillsboost.google/focuses/2802?parent=catalog

* Overview
  + learn fundamental SQL querying keywords, which you will run in the BigQuery console on a public dataset that contains information on London bikeshares.
  + learn how to export subsets of the London bikeshare dataset into CSV files, which you will then upload to Cloud SQL. From there you will learn how to use Cloud SQL to create and manage databases and tables. Towards the end, you will get hands-on practice with additional SQL keywords that manipulate and edit data.
* Objectives. In this lab, you will learn how to:
  + Distinguish databases from tables and projects.
  + Use the SELECT, FROM, and WHERE keywords to construct simple queries.
  + Identify the different components and hierarchies within the BigQuery console.
  + Load databases and tables into BigQuery.
  + Execute simple queries on tables.
  + Learn about the COUNT, GROUP BY, AS, and ORDER BY keywords.
  + Execute and chain the above commands to pull meaningful data from datasets.
  + Export a subset of data into a CSV file and store that file into a new Cloud Storage bucket.
  + Create a new Cloud SQL instance and load your exported CSV file as a new table.
  + Run CREATE DATABASE, CREATE TABLE, DELETE, INSERT INTO, and UNION queries in Cloud SQL.
* The Basics of SQL
  + Databases and Tables
    - An example of unstructured data would be an image file. Unstructured data is inoperable with SQL and cannot be stored in BigQuery datasets or tables (at least natively.) To work with image data (for instance), you would use a service like Cloud Vision, perhaps through its API directly.
    - The following is an example of a structured dataset—a simple table:

|  |  |  |
| --- | --- | --- |
| **User** | **Price** | **Shipped** |
| Sean | $35 | Yes |
| Rocky | $50 | No |

* + - A Database is essentially a collection of one or more tables.
  + A keyword that specifies the fields (e.g. column values) that you want to pull from your dataset. SELECT
  + Specifies what table or tables we want to pull our data from. FROM
  + Allows us to filter tables for specific column values. WHERE
* Exploring the BigQuery Console
  + The BigQuery paradigm
    - BigQuery is a fully-managed petabyte-scale data warehouse that runs on the Google Cloud. Data analysts and data scientists can quickly query and filter large datasets, aggregate results, and perform complex operations without having to worry about setting up and managing servers. It comes in the form of a command line tool (preinstalled in cloudshell) or a web console—both ready for managing and querying data housed in Google Cloud projects.
  + Open BigQuery, select Navigation menu > BigQuery: Click Done
    - The query "Editor", where you write and run SQL commands.
    - "Query history", which is a list of queries you ran previously.
    - In BigQuery, projects contain datasets, and datasets contain tables. the project → dataset → table.
  + Uploading queryable data
    - In this section you pull in some public data into your project so you can practice running SQL commands in BigQuery.
    - Click on the + ADD DATA link then select Explore public datasets:
    - In the search bar, Enter, then select the London Bicycle Hires tile, then View Dataset.
    - A new tab will open, and you will now have a new project called bigquery-public-data added to the Explorer panel:
    - Graphical user interface, application

      Description automatically generated
    - Expand bigquery-public-data > london\_bicycles and select cycle\_hire. You now have data that follows the BigQuery paradigm:
      * Google Cloud Project → bigquery-public-data
      * Dataset → london\_bicycles
      * Table → cycle\_hire
  + Running SELECT, FROM, and WHERE in BigQuery
    - A fully-managed petabyte-scale data warehouse that runs on the Google Cloud. BigQuery
    - Projects contain datasets, and datasets contain tables. True
    - With BigQuery, you can access datasets shared publicly from other Google Cloud projects. True
  + More SQL Keywords: GROUP BY, COUNT, AS (alias), and ORDER BY
    - Aggregates rows that share common criteria (e.g. a column value) and will return all of the unique entries found for such criteria. GROUP BY
    - A SQL function will count and return the number of rows that share common criteria. COUNT
    - Creates an alias of a table or column. AS
    - Sorts the returned data from a query in ascending or descending order based on a specified criteria or column value. ORDER BY
* Working with Cloud SQL
  + Exporting queries as CSV files
    - Cloud SQL is a fully-managed database service that makes it easy to set up, maintain, manage, and administer your relational PostgreSQL and MySQL databases in the cloud. There are two formats of data accepted by Cloud SQL: dump files (.sql) or CSV files (.csv). You will learn how to export subsets of the cycle\_hire table into CSV files and upload them to Cloud Storage as an intermediate location.
    - Back in the BigQuery Console, this should have been the last command that you ran:
      * SELECT start\_station\_name, COUNT(\*) AS num FROM `bigquery-public-data.london\_bicycles.cycle\_hire` GROUP BY start\_station\_name ORDER BY num DESC;
    - In the Query Results section click SAVE RESULTS > CSV(start\_station\_name .csv) SAVE.
      * SELECT end\_station\_name, COUNT(\*) AS num FROM `bigquery-public-data.london\_bicycles.cycle\_hire` GROUP BY end\_station\_name ORDER BY num DESC;
    - In the Query Results section click SAVE RESULTS > CSV(end\_station\_name .csv)
  + Upload CSV files to Cloud Storage
    - Select Navigation menu > Cloud Storage > Browser > click CREATE BUCKET.
    - Enter a unique name for your bucket, keep all other settings as default, and click Create:
    - Click UPLOAD FILES both csv files
    - Graphical user interface

      Description automatically generated
  + Create a Cloud SQL instance. Navigation menu > SQL > CREATE INSTANCE> Select MySQL.
    - Now enter in a name for your instance (like "qwiklabs-demo") and enter a secure password, then click CREATE INSTANCE:
* New Queries in Cloud SQL
  + Run the following command in Cloud Shell to connect to your SQL instance:
    - gcloud sql connect qwiklabs-demo --user=root
    - When prompted, enter the root password you set for the instance.
  + A Cloud SQL instance comes with pre-configured databases, but you will create your own to store the London bikeshare data.
  + Run the following command at the MySQL server prompt to create a database called bike:
    - CREATE DATABASE bike;
  + Make a table inside of the bike database by running the following command:
    - USE bike;
    - CREATE TABLE london1 (start\_station\_name VARCHAR(255), num INT);
  + Create another table named "london2" by running the following command:
    - USE bike;
    - CREATE TABLE london2 (end\_station\_name VARCHAR(255), num INT);
  + Now confirm that your empty tables were created. Run the following commands at the MySQL server prompt:
    - SELECT \* FROM london1;
    - SELECT \* FROM london2;
  + Upload CSV files to tables
    - Return to the Cloud SQL console. You will now upload the start\_station\_name and end\_station\_name CSV files into your newly created london1 and london2 tables.
    - In your Cloud SQL instance page, click IMPORT.
    - In the Cloud Storage file field, click Browse, and then click the arrow opposite your bucket name, and then click start\_station\_data.csv. Click Select.
    - Select CSV as File format.
    - Select the bike database and type in "london1" as your table. Click Import:
    - 
    - Do the same for the other CSV file.
  + You should now have both CSV files uploaded to tables in the bike database.
  + Return to your Cloud Shell session and run the following command at the MySQL server prompt to inspect the contents of london1:
    - SELECT \* FROM london1;
    - SELECT \* FROM london2;
    - DELETE FROM london1 WHERE num=0;
    - DELETE FROM london2 WHERE num=0;
    - INSERT INTO london1 (start\_station\_name, num) VALUES ("test destination", 1);

## 3. Multiple VPC Networks - https://www.cloudskillsboost.google/focuses/22772?parent=catalog

* Overview
  + In this lab you create several VPC networks and VM instances and test connectivity across networks. Specifically, you create two custom mode networks (managementnet and privatenet) with firewall rules and VM instances as shown in this network diagram:
  + Graphical user interface

    Description automatically generated
  + The mynetwork network with its firewall rules and two VM instances (mynet-eu-vm and mynet-us-vm) have already been created for you in this Qwiklabs project.
* Objectives, In this lab, you will learn how to perform the following tasks:
  + Create custom mode VPC networks with firewall rules
  + Create VM instances using Compute Engine
  + Explore the connectivity for VM instances across VPC networks
  + Create a VM instance with multiple network interfaces
* Create custom mode VPC networks with firewall rules
  + Create two custom networks managementnet and privatenet, along with firewall rules to allow SSH, ICMP, and RDP ingress traffic.
  + Create the managementnet network
    - Create the managementnet network using the Cloud Console.
    - In the Cloud Console, navigate to Navigation menu (mainmenu.png) > VPC network > VPC networks.
      * Notice the default and mynetwork networks with their subnets.
      * Each Google Cloud project starts with the default network. In addition, the mynetwork network has been premade as part of your network diagram.
    - Click Create VPC Network.
      * Set the Name to managementnet. For Subnet creation mode, click Custom.
      * Set the following values, leave all other values at their defaults: Click Done. Click Close. Click Create.
      * Graphical user interface, text, application

        Description automatically generated

|  |  |
| --- | --- |
| **Property** | **Value (type value or select option as specified)** |
| Name | managementsubnet-us |
| Region | us-central1 |
| IP address range | 10.130.0.0/2 |

* + Create the privatenet network using the Cloud Shell command line.
    - Run the following command to create the privatenet network:
      * gcloud compute networks create privatenet --subnet-mode=custom
    - Run the following command to create the privatesubnet-us subnet:
      * gcloud compute networks subnets create privatesubnet-us --network=privatenet --region=us-central1 --range=172.16.0.0/24
    - Run the following command to create the privatesubnet-eu subnet:
      * gcloud compute networks subnets create privatesubnet-eu --network=privatenet --region=europe-west4 --range=172.20.0.0/20
    - Run the following command to list the available VPC networks:
      * gcloud compute networks list
    - default and mynetwork are auto mode networks, whereas, managementnet and privatenet are custom mode networks. Auto mode networks create subnets in each region automatically, while custom mode networks start with no subnets, giving you full control over subnet creation
    - Run the following command to list the available VPC subnets (sorted by VPC network):
      * gcloud compute networks subnets list --sort-by=NETWORK
    - As expected, the default and mynetwork networks have subnets in each region as they are auto mode networks. The managementnet and privatenet networks only have the subnets that you created as they are custom mode networks.
    - In the Cloud Console, navigate to Navigation menu > VPC network > VPC networks.
      * You see that the same networks and subnets are listed in the Cloud Console.
      * Graphical user interface, text, application

        Description automatically generated
  + Create the firewall rules for managementnet
    - * Create firewall rules to allow SSH, ICMP, and RDP ingress traffic to VM instances on the managementnet network.
      * In the Cloud Console, navigate to Navigation menu (mainmenu.png) > VPC network > Firewall.
      * Click + Create Firewall Rule. Set the following values, leave all other values at their defaults: Click Close. Click Create.

|  |  |
| --- | --- |
| **Property** | **Value (type value or select option as specified)** |
| Name | managementnet-allow-icmp-ssh-rdp |
| Network | managementnet |
| Targets | All instances in the network |
| Source filter | IPv4 Ranges |
| Source IPv4 ranges | 0.0.0.0/0 |
| Protocols and ports | Specified protocols and ports, and then *check* tcp, *type:* 22, 3389; and *check* Other protocols, *type:* icmp. |

* + Create the firewall rules for privatenet network using the Cloud Shell command line.
    - In Cloud Shell, run the following command to create the privatenet-allow-icmp-ssh-rdp firewall rule:
      * gcloud compute firewall-rules create privatenet-allow-icmp-ssh-rdp --direction=INGRESS --priority=1000 --network=privatenet --action=ALLOW --rules=icmp,tcp:22,tcp:3389 --source-ranges=0.0.0.0/0
    - Run the following command to list all the firewall rules (sorted by VPC network):
      * gcloud compute firewall-rules list --sort-by=NETWORK
    - The firewall rules for mynetwork network have been created for you. You can define multiple protocols and ports in one firewall rule (privatenet and managementnet), or spread them across multiple rules (default and mynetwork).
    - In the Cloud Console, navigate to Navigation menu > VPC network > Firewall.
      * You see that the same firewall rules are listed in the Cloud Console.
      * Graphical user interface, application

        Description automatically generated
* Create VM instances
  + - Create two VM instances:
      * managementnet-us-vm in managementsubnet-us
      * privatenet-us-vm in privatesubnet-us
    - Create the managementnet-us-vm instance
      * Create the managementnet-us-vm instance using the Cloud Console.
      * In the Cloud Console, navigate to Navigation menu > Compute Engine > VM instances.
        + The mynet-eu-vm and mynet-us-vm has been created for you, as part of your network diagram.
      * Click Create instance. Set the following values, leave all other values at their defaults:

|  |  |
| --- | --- |
| **Property** | **Value (type value or select option as specified)** |
| Name | managementnet-us-vm |
| Region | us-central1 |
| Zone | us-central1-f |
| Series | N1 |
| Machine type | 1 vCPU (f1-micro) |

* + - * Click NETWORKING, DISKS, SECURITY, MANAGEMENT, SOLE-TENANCY.
        + Click Networking.
        + For Network interfaces, click the dropdown to edit.
        + Set the following values, leave all other values at their defaults:

Property Value (type value or select option as specified)

Network managementnet

Subnetwork managementsubnet-us

* + - * Click Done. Click Create.
* Create the privatenet-us-vm instance using the Cloud Shell command line.
  + In Cloud Shell, run the following command to create the privatenet-us-vm instance:
    - gcloud compute instances create privatenet-us-vm --zone=us-central1-f --machine-type=n1-standard-1 --subnet=privatesubnet-us
  + Run the following command to list all the VM instances (sorted by zone):
    - gcloud compute instances list --sort-by=ZONE
  + In the Cloud Console, navigate to Navigation menu (mainmenu.png) > Compute Engine > VM instances.
  + You see that the VM instances are listed in the Cloud Console.
    - Click on Column display options, then select Network. Click Ok.
  + There are three instances in us-central1-f and one instance in europe-west4-c. However, these instances are spread across three VPC networks (managementnet, mynetwork and privatenet), with no instance in the same zone and network as another. In the next section, you explore the effect this has on internal connectivity.
* Explore the connectivity between VM instances
  + Explore the connectivity between the VM instances. Specifically, determine the effect of having VM instances in the same zone versus having instances in the same VPC network.
  + Ping the external IP addresses
    - Ping the external IP addresses of the VM instances to determine if you can reach the instances from the public internet.
    - In the Cloud Console, navigate to Navigation menu > Compute Engine > VM instances.
    - Note the external IP addresses for mynet-eu-vm, managementnet-us-vm, and privatenet-us-vm.
    - For mynet-us-vm, click SSH to launch a terminal and connect.
    - To test connectivity to mynet-eu-vm's external IP, run the following command, replacing mynet-eu-vm's external IP:
      * ping -c 3 <Enter mynet-eu-vm's external IP here>
      * This should work!
    - To test connectivity to managementnet-us-vm's external IP, run the following command, replacing managementnet-us-vm's external IP:
      * ping -c 3 <Enter managementnet-us-vm's external IP here>
      * This should work!
    - To test connectivity to privatenet-us-vm's external IP, run the following command, replacing privatenet-us-vm's external IP:
      * ping -c 3 <Enter privatenet-us-vm's external IP here>
      * This should work!
    - You are able to ping the external IP address of all VM instances, even though they are either in a different zone or VPC network. This confirms public access to those instances is only controlled by the ICMP firewall rules that you established earlier.
  + Ping the internal IP addresses
    - Ping the internal IP addresses of the VM instances to determine if you can reach the instances from within a VPC network.
    - In the Cloud Console, navigate to Navigation menu > Compute Engine > VM instances.
    - Note the internal IP addresses for mynet-eu-vm, managementnet-us-vm, and privatenet-us-vm.
    - Return to the SSH terminal for mynet-us-vm.
    - To test connectivity to mynet-eu-vm's internal IP, run the following command, replacing mynet-eu-vm's internal IP:
      * ping -c 3 <Enter mynet-eu-vm's internal IP here>
    - You are able to ping the internal IP address of mynet-eu-vm because it is on the same VPC network as the source of the ping (mynet-us-vm), even though both VM instances are in separate zones, regions and continents!
    - To test connectivity to managementnet-us-vm's internal IP, run the following command, replacing managementnet-us-vm's internal IP:
      * ping -c 3 <Enter managementnet-us-vm's internal IP here>
      * This should not work as indicated by a 100% packet loss!
    - To test connectivity to privatenet-us-vm's internal IP, run the following command, replacing privatenet-us-vm's internal IP:
      * ping -c 3 <Enter privatenet-us-vm's internal IP here>
      * This should not work either as indicated by a 100% packet loss! You are unable to ping the internal IP address of managementnet-us-vm and privatenet-us-vm because they are in separate VPC networks from the source of the ping (mynet-us-vm), even though they are all in the same zone us-central1.
    - VPC networks are by default isolated private networking domains. However, no internal IP address communication is allowed between networks, unless you set up mechanisms such as VPC peering or VPN.
    - Which instance(s) should you be able to ping from mynet-us-vm using internal IP addresses? mynet-eu-vm
* Create a VM instance with multiple network interfaces
  + Every instance in a VPC network has a default network interface. You can create additional network interfaces attached to your VMs. Multiple network interfaces enable you to create configurations in which an instance connects directly to several VPC networks (up to 8 interfaces, depending on the instance's type).
  + Create the VM instance with multiple network interfaces
    - Create the vm-appliance instance with network interfaces in privatesubnet-us, managementsubnet-us and mynetwork. The CIDR ranges of these subnets do not overlap, which is a requirement for creating a VM with multiple network interface controllers (NICs).
    - In the Cloud Console, navigate to Navigation menu > Compute Engine > VM instances.
      * Click Create instance. Set the following values, leave all other values at their defaults:
        + Property Value (type value or select option as specified)
        + Name vm-appliance
        + Region us-central1
        + Zone us-central1-f
        + Series N1
        + Machine type 4 vCPUs (n1-standard-4)
      * The number of interfaces allowed in an instance is dependent on the instance's machine type and the number of vCPUs. The n1-standard-4 allows up to 4 network interfaces. Refer here for more information.
      * Click NETWORKING, DISKS, SECURITY, MANAGEMENT, SOLE-TENANCY.
        + Click Networking. For Network interfaces, click the dropdown to edit.

Set the following values, leave all other values at their defaults:

Property Value (type value or select option as specified)

Network privatenet

Subnetwork privatesubnet-us

Click Done.

* + - * + Click Add network interface.

Set the following values, leave all other values at their defaults:

Property Value (type value or select option as specified)

Network managementnet

Subnetwork managementsubnet-us

Click Done.

* + - * + Click Add network interface.

Set the following values, leave all other values at their defaults:

Property Value (type value or select option as specified)

Network mynetwork

Subnetwork mynetwork

Click Done.

* + - * Click Create.
* Explore the network interface details of vm-appliance within the Cloud Console and within the VM's terminal.
  + In the Cloud Console, navigate to Navigation menu (mainmenu.png) > Compute Engine > VM instances.
    - Click nic0 within the Internal IP address of vm-appliance to open the Network interface details page.
  + Verify that nic0 is attached to privatesubnet-us, is assigned an internal IP address within that subnet (172.16.0.0/24), and has applicable firewall rules.
    - Click nic0 and select nic1.
  + Verify that nic1 is attached to managementsubnet-us, is assigned an internal IP address within that subnet (10.130.0.0/20), and has applicable firewall rules.
    - Click nic1 and select nic2.
  + Verify that nic2 is attached to mynetwork, is assigned an internal IP address within that subnet (10.128.0.0/20), and has applicable firewall rules.
    - Each network interface has its own internal IP address so that the VM instance can communicate with those networks.
  + In the Cloud Console, navigate to Navigation menu > Compute Engine > VM instances.
  + For vm-appliance, click SSH to launch a terminal and connect.
  + Run the following, to list the network interfaces within the VM instance:
    - sudo ifconfig
      * The sudo ifconfig command lists a Linux VM's network interfaces along with the internal IP addresses for each interface.
* Explore the network interface connectivity
  + Demonstrate that the vm-appliance instance is connected to privatesubnet-us, managementsubnet-us and mynetwork by pinging VM instances on those subnets.
  + In the Cloud Console, navigate to Navigation menu > Compute Engine > VM instances.
  + Note the internal IP addresses for privatenet-us-vm, managementnet-us-vm, mynet-us-vm, and mynet-eu-vm.
  + Return to the SSH terminal for vm-appliance.
  + To test connectivity to privatenet-us-vm's internal IP, run the following command, replacing privatenet-us-vm's internal IP:
    - ping -c 3 <Enter privatenet-us-vm's internal IP here>
    - This works!
  + Repeat the same test by running the following:
    - ping -c 3 privatenet-us-vm
  + You are able to ping privatenet-us-vm by its name because VPC networks have an internal DNS service that allows you to address instances by their DNS names rather than their internal IP addresses. When an internal DNS query is made with the instance hostname, it resolves to the primary interface (nic0) of the instance. Therefore, this only works for privatenet-us-vm in this case.
  + To test connectivity to managementnet-us-vm's internal IP, run the following command, replacing managementnet-us-vm's internal IP:
    - ping -c 3 <Enter managementnet-us-vm's internal IP here>
    - This works!
  + To test connectivity to mynet-us-vm's internal IP, run the following command, replacing mynet-us-vm's internal IP:
    - ping -c 3 <Enter mynet-us-vm's internal IP here>
    - This works!
  + To test connectivity to mynet-eu-vm's internal IP, run the following command, replacing mynet-eu-vm's internal IP:
    - ping -c 3 <Enter mynet-eu-vm's internal IP here>
    - This does not work! In a multiple interface instance, every interface gets a route for the subnet that it is in. In addition, the instance gets a single default route that is associated with the primary interface eth0. Unless manually configured otherwise, any traffic leaving an instance for any destination other than a directly connected subnet will leave the instance via the default route on eth0.
  + To list the routes for vm-appliance instance, run the following command:
    - ip route
  + The primary interface eth0 gets the default route (default via 172.16.0.1 dev eth0), and all three interfaces eth0, eth1 and eth2 get routes for their respective subnets. Since, the subnet of mynet-eu-vm (10.132.0.0/20) is not included in this routing table, the ping to that instance leaves vm-appliance on eth0 (which is on a different VPC network). You could change this behavior by configuring policy routing as documented here.

## 5. Managing Deployments Using Kubernetes Engine - https://www.cloudskillsboost.google/focuses/639?parent=catalog

* Overview
  + Dev Ops practices will regularly make use of multiple deployments to manage application deployment scenarios such as "Continuous Deployment", "Blue-Green Deployments", "Canary Deployments" and more. This lab is to provide practice in scaling and managing containers so you can accomplish these common scenarios where multiple heterogeneous deployments are being used.
* What you'll do
  + Practice with kubectl tool
  + Create deployment yaml files
  + Launch, update, and scale deployments
  + Practice with updating deployments and deployment styles
* Introduction to deployments
  + Heterogeneous deployments typically involve connecting two or more distinct infrastructure environments or regions to address a specific technical or operational need. Heterogeneous deployments are called "hybrid", "multi-cloud", or "public-private", depending upon the specifics of the deployment. For the purposes of this lab, heterogeneous deployments include those that span regions within a single cloud environment, multiple public cloud environments (multi-cloud), or a combination of on-premises and public cloud environments (hybrid or public-private).
  + Various business and technical challenges can arise in deployments that are limited to a single environment or region:
    - Maxed out resources: In any single environment, particularly in on-premises environments, you might not have the compute, networking, and storage resources to meet your production needs.
    - Limited geographic reach: Deployments in a single environment require people who are geographically distant from one another to access one deployment. Their traffic might travel around the world to a central location.
    - Limited availability: Web-scale traffic patterns challenge applications to remain fault-tolerant and resilient.
    - Vendor lock-in: Vendor-level platform and infrastructure abstractions can prevent you from porting applications.
    - Inflexible resources: Your resources might be limited to a particular set of compute, storage, or networking offerings.
  + Heterogeneous deployments can help address these challenges, but they must be architected using programmatic and deterministic processes and procedures. One-off or ad-hoc deployment procedures can cause deployments or processes to be brittle and intolerant of failures. Ad-hoc processes can lose data or drop traffic. Good deployment processes must be repeatable and use proven approaches for managing provisioning, configuration, and maintenance.
  + Three common scenarios for heterogeneous deployment are multi-cloud deployments, fronting on-premises data, and continuous integration/continuous delivery (CI/CD) processes.
  + Set zone gcloud config set compute/zone us-central1-a
  + Get sample code for creating and running containers and deployments:
    - gsutil -m cp -r gs://spls/gsp053/orchestrate-with-kubernetes .
    - cd orchestrate-with-kubernetes/kubernetes
  + Create a cluster with five n1-standard-1 nodes (this will take a few minutes to complete):
    - gcloud container clusters create bootcamp --num-nodes 5 --scopes "https://www.googleapis.com/auth/projecthosting,storage-rw"
* Learn about the deployment object
  + Let's get started with Deployments. First let's take a look at the Deployment object. The explain command in kubectl can tell us about the Deployment object.
    - kubectl explain deployment
  + We can also see all of the fields using the --recursive option.
    - kubectl explain deployment --recursive
  + You can use the explain command as you go through the lab to help you understand the structure of a Deployment object and understand what the individual fields do.
    - kubectl explain deployment.metadata.name
* Create a deployment
  + Update the deployments/auth.yaml configuration file:
    - vi deployments/auth.yaml
  + Start the editor: i
  + Change the image in the containers section of the Deployment to the following:
    - ...
    - containers:
    - - name: auth
    - image: "kelseyhightower/auth:1.0.0"
    - ...
  + Save the auth.yaml file: press <Esc> then type: :wq
  + Press <Enter>. Now let's create a simple deployment. Examine the deployment configuration file:
    - cat deployments/auth.yaml
  + Notice how the Deployment is creating one replica and it's using version 1.0.0 of the auth container.
  + When you run the kubectl create command to create the auth deployment, it will make one pod that conforms to the data in the Deployment manifest. This means we can scale the number of Pods by changing the number specified in the replicas field.
  + Go ahead and create your deployment object using kubectl create:
    - kubectl create -f deployments/auth.yaml
  + Once you have created the Deployment, you can verify that it was created.
    - kubectl get deployments
  + Once the deployment is created, Kubernetes will create a ReplicaSet for the Deployment. We can verify that a ReplicaSet was created for our Deployment:
    - kubectl get replicasets
  + We should see a ReplicaSet with a name like auth-xxxxxxx
  + Finally, we can view the Pods that were created as part of our Deployment. The single Pod is created by the Kubernetes when the ReplicaSet is created.
    - kubectl get pods
  + It's time to create a service for our auth deployment. You've already seen service manifest files, so we won't go into the details here. Use the kubectl create command to create the auth service.
    - kubectl create -f services/auth.yaml
  + Now, do the same thing to create and expose the hello Deployment.
    - kubectl create -f deployments/hello.yaml
    - kubectl create -f services/hello.yaml
  + And one more time to create and expose the frontend Deployment.
    - kubectl create secret generic tls-certs --from-file tls/
    - kubectl create configmap nginx-frontend-conf --from-file=nginx/frontend.conf
    - kubectl create -f deployments/frontend.yaml
    - kubectl create -f services/frontend.yaml
  + Note: You created a ConfigMap for the frontend.
  + Interact with the frontend by grabbing its external IP and then curling to it.
    - kubectl get services frontend
  + It may take a few seconds before the External-IP field is populated for your service. This is normal. Just re-run the above command every few seconds until the field is populated.
    - curl -ks https://<EXTERNAL-IP>
  + And you get the hello response back.
  + You can also use the output templating feature of kubectl to use curl as a one-liner:
    - curl -ks https://`kubectl get svc frontend -o=jsonpath="{.status.loadBalancer.ingress[0].ip}"`
* Scale a Deployment
  + Now that we have a Deployment created, we can scale it. Do this by updating the spec.replicas field. You can look at an explanation of this field using the kubectl explain command again.
    - kubectl explain deployment.spec.replicas
  + The replicas field can be most easily updated using the kubectl scale command:
    - kubectl scale deployment hello --replicas=5
  + Note: It may take a minute or so for all the new pods to start up.
  + After the Deployment is updated, Kubernetes will automatically update the associated ReplicaSet and start new Pods to make the total number of Pods equal 5.
  + Verify that there are now 5 hello Pods running:
    - kubectl get pods | grep hello- | wc -l
  + Now scale back the application:
    - kubectl scale deployment hello --replicas=3
  + Again, verify that you have the correct number of Pods.
    - kubectl get pods | grep hello- | wc -l
  + You learned about Kubernetes deployments and how to manage & scale a group of Pods.
* Rolling update
  + Deployments support updating images to a new version through a rolling update mechanism. When a Deployment is updated with a new version, it creates a new ReplicaSet and slowly increases the number of replicas in the new ReplicaSet as it decreases the replicas in the old ReplicaSet.
  + Timeline

    Description automatically generated
  + Trigger a rolling update
    - To update your Deployment, run the following command:
      * kubectl edit deployment hello
    - Change the image in the containers section of the Deployment to the following:
      * ...
      * containers:
      * image: kelseyhightower/hello:2.0.0
      * ...
    - Save and exit.
    - Once you save out of the editor, the updated Deployment will be saved to your cluster and Kubernetes will begin a rolling update.
    - See the new ReplicaSet that Kubernetes creates.:
      * kubectl get replicaset
    - You can also see a new entry in the rollout history:
      * kubectl rollout history deployment/hello
  + Pause a rolling update
    - If you detect problems with a running rollout, pause it to stop the update. Give that a try now:
      * kubectl rollout pause deployment/hello
    - Verify the current state of the rollout:
      * kubectl rollout status deployment/hello
    - You can also verify this on the Pods directly:
    - kubectl get pods -o jsonpath --template='{range .items[\*]}{.metadata.name}{"\t"}{"\t"}{.spec.containers[0].image}{"\n"}{end}'
  + Resume a rolling update
    - The rollout is paused which means that some pods are at the new version and some pods are at the older version. We can continue the rollout using the resume command.
      * kubectl rollout resume deployment/hello
    - When the rollout is complete, you should see the following when running the status command.
      * kubectl rollout status deployment/hello
  + Rollback an update
    - Assume that a bug was detected in your new version. Since the new version is presumed to have problems, any users connected to the new Pods will experience those issues.
    - You will want to roll back to the previous version so you can investigate and then release a version that is fixed properly.
    - Use the rollout command to roll back to the previous version:
      * kubectl rollout undo deployment/hello
    - Verify the roll back in the history:
      * kubectl rollout history deployment/hello
    - Finally, verify that all the Pods have rolled back to their previous versions:
      * kubectl get pods -o jsonpath --template='{range .items[\*]}{.metadata.name}{"\t"}{"\t"}{.spec.containers[0].image}{"\n"}{end}'
    - Great! You learned about rolling updates for Kubernetes deployments and how to update applications without downtime.
* Canary deployments
  + When you want to test a new deployment in production with a subset of your users, use a canary deployment. Canary deployments allow you to release a change to a small subset of your users to mitigate risk associated with new releases.
  + Create a canary deployment
    - A canary deployment consists of a separate deployment with your new version and a service that targets both your normal, stable deployment as well as your canary deployment.
      * Diagram

        Description automatically generated with medium confidence
    - First, create a new canary deployment for the new version:
      * cat deployments/hello-canary.yaml
    - Now create the canary deployment:
      * kubectl create -f deployments/hello-canary.yaml
    - After the canary deployment is created, you should have two deployments, hello and hello-canary. Verify it with this kubectl command:
      * kubectl get deployments
    - On the hello service, the selector uses the app:hello selector which will match pods in both the prod deployment and canary deployment. However, because the canary deployment has a fewer number of pods, it will be visible to fewer users.
  + Verify the canary deployment
    - You can verify the hello version being served by the request:
      * curl -ks https://`kubectl get svc frontend -o=jsonpath="{.status.loadBalancer.ingress[0].ip}"`/version
    - Run this several times and you should see that some of the requests are served by hello 1.0.0 and a small subset (1/4 = 25%) are served by 2.0.0.
  + Canary deployments in production - session affinity
    - In this lab, each request sent to the Nginx service had a chance to be served by the canary deployment. But what if you wanted to ensure that a user didn't get served by the Canary deployment? A use case could be that the UI for an application changed, and you don't want to confuse the user. In a case like this, you want the user to "stick" to one deployment or the other.
    - You can do this by creating a service with session affinity. This way the same user will always be served from the same version. In the example below the service is the same as before, but a new sessionAffinity field has been added, and set to ClientIP. All clients with the same IP address will have their requests sent to the same version of the hello application.
      * kind: Service
      * apiVersion: v1
      * metadata:
      * name: "hello"
      * spec:
      * sessionAffinity: ClientIP
      * selector:
      * app: "hello"
      * ports:
      * - protocol: "TCP"
      * port: 80
      * targetPort: 80
    - Due to it being difficult to set up an environment to test this, you don't need to here, but you may want to use sessionAffinity for canary deployments in production.
* Blue-green deployments
  + Rolling updates are ideal because they allow you to deploy an application slowly with minimal overhead, minimal performance impact, and minimal downtime. There are instances where it is beneficial to modify the load balancers to point to that new version only after it has been fully deployed. In this case, blue-green deployments are the way to go.
  + Kubernetes achieves this by creating two separate deployments; one for the old "blue" version and one for the new "green" version. Use your existing hello deployment for the "blue" version. The deployments will be accessed via a Service which will act as the router. Once the new "green" version is up and running, you'll switch over to using that version by updating the Service.
    - Timeline

      Description automatically generated
  + A major downside of blue-green deployments is that you will need to have at least 2x the resources in your cluster necessary to host your application. Make sure you have enough resources in your cluster before deploying both versions of the application at once.
  + The service
    - Use the existing hello service, but update it so that it has a selector app:hello, version: 1.0.0. The selector will match the existing "blue" deployment. But it will not match the "green" deployment because it will use a different version.
    - First update the service:
      * kubectl apply -f services/hello-blue.yaml
        + NOTE: Ignore the warning that says resource service/hello is missing as this is patched automatically.
  + Updating using Blue-Green Deployment
    - In order to support a blue-green deployment style, we will create a new "green" deployment for our new version. The green deployment updates the version label and the image path.
      * apiVersion: apps/v1
      * kind: Deployment
      * metadata:
      * name: hello-green
      * spec:
      * replicas: 3
      * selector:
      * matchLabels:
      * app: hello
      * template:
      * metadata:
      * labels:
      * app: hello
      * track: stable
      * version: 2.0.0
      * spec:
      * containers:
      * - name: hello
      * image: kelseyhightower/hello:2.0.0
      * ports:
      * - name: http
      * containerPort: 80
      * - name: health
      * containerPort: 81
      * resources:
      * limits:
      * cpu: 0.2
      * memory: 10Mi
      * livenessProbe:
      * httpGet:
      * path: /healthz
      * port: 81
      * scheme: HTTP
      * initialDelaySeconds: 5
      * periodSeconds: 15
      * timeoutSeconds: 5
      * readinessProbe:
      * httpGet:
      * path: /readiness
      * port: 81
      * scheme: HTTP
      * initialDelaySeconds: 5
      * timeoutSeconds: 1
    - Create the green deployment:
      * kubectl create -f deployments/hello-green.yaml
    - Once you have a green deployment and it has started up properly, verify that the current version of 1.0.0 is still being used:
      * curl -ks https://`kubectl get svc frontend -o=jsonpath="{.status.loadBalancer.ingress[0].ip}"`/version
    - Now, update the service to point to the new version:
      * kubectl apply -f services/hello-green.yaml
    - When the service is updated, the "green" deployment will be used immediately. You can now verify that the new version is always being used.
      * curl -ks https://`kubectl get svc frontend -o=jsonpath="{.status.loadBalancer.ingress[0].ip}"`/version
  + Blue-Green Rollback
    - If necessary, you can roll back to the old version in the same way. While the "blue" deployment is still running, just update the service back to the old version.
      * kubectl apply -f services/hello-blue.yaml
    - Once you have updated the service, your rollback will have been successful. Again, verify that the right version is now being used:
      * curl -ks https://`kubectl get svc frontend -o=jsonpath="{.status.loadBalancer.ingress[0].ip}"`/version
    - You did it! You learned about blue-green deployments and how to deploy updates to applications that need to switch versions all at once.

## 6. Set Up and Configure a Cloud Environment in Google Cloud: Challenge Lab - https://www.cloudskillsboost.google/focuses/10603?parent=catalog

* Topics tested:
  + Creating and using VPCs and subnets
  + Creating a Kubernetes cluster
  + Configuring and launching a Kubernetes deployment and service
  + Setting up stackdriver monitoring
  + Configuring an IAM role for an account
* Challenge scenario
  + As a cloud engineer in Jooli Inc. and recently trained with Google Cloud and Kubernetes you have been asked to help a new team (Griffin) set up their environment. The team has asked for your help and has done some work, but needs you to complete the work.
  + You need to complete the following tasks:
    - Create a development VPC with three subnets manually
    - Create a production VPC with three subnets manually
    - Create a bastion that is connected to both VPCs
    - Create a development Cloud SQL Instance and connect and prepare the WordPress environment
    - Create a Kubernetes cluster in the development VPC for WordPress
    - Prepare the Kubernetes cluster for the WordPress environment
    - Create a WordPress deployment using the supplied configuration
    - Enable monitoring of the cluster via stackdriver
    - Provide access for an additional engineer
  + Some Jooli Inc. standards you should follow:
    - Create all resources in the us-east1 region and us-east1-b zone, unless otherwise directed.
    - Use the project VPCs.
    - Naming is normally team-resource, e.g. an instance could be named kraken-webserver1.
    - unless directed, use n1-standard-1.
  + Your challenge
    - You need to help the team with some of their initial work on a new project. They plan to use WordPress and need you to set up a development environment. Some of the work was already done for you, but other parts require your expert skills.
  + Environment
    - Diagram

      Description automatically generated with medium confidence
  + Task 1: Create development VPC manually
    - Create a VPC called griffin-dev-vpc with the following subnets only:
      * griffin-dev-wp
        + IP address block: 192.168.16.0/20
      * griffin-dev-mgmt
        + IP address block: 192.168.32.0/20
    - Solution through Cmd Line
      * gcloud config set compute/region us-east1
      * gcloud config set compute/zone us-east1-b
      * gcloud compute networks create griffin-dev-vpc --subnet-mode=custom
      * gcloud compute networks subnets create griffin-dev-wp --network=griffin-dev-vpc --region=us-east1 --range=192.168.16.0/20
      * gcloud compute networks subnets create griffin-dev-mgmt --network=griffin-dev-vpc --region=us-east1 --range=192.168.32.0/20
    - Solution through Console
      * Navigation Menu > VPC network > VPC networks > Click on Create VPC network.
      * Enter griffin-dev-vpc to the Name field.
      * Select Custom for the Subnet creation mode.
      * Add griffin-dev-wp subnet with the following parameters:
        + Field Value
        + Name: griffin-dev-wp
        + Region: us-east1
        + IP address range: 192.168.16.0/20
      * Click + Add subnet and add griffin-dev-mgmt subnet with the following parameters
        + Field Value
        + Name: griffin-dev-mgmt
        + Region: us-east1
        + IP address range: 192.168.32.0/20
      * Click Create.
  + Task 2: Create production VPC manually
    - Create a VPC called griffin-prod-vpc with the following subnets only:
      * griffin-prod-wp
        + IP address block: 192.168.48.0/20
      * griffin-prod-mgmt
        + IP address block: 192.168.64.0/20
    - Solution through Cmd Line
      * gcloud config set compute/region us-east1
      * gcloud config set compute/zone us-east1-b
      * gcloud compute networks create griffin-prod-vpc --subnet-mode=custom
      * gcloud compute networks subnets create griffin-prod-wp --network=griffin-prod-vpc --region=us-east1 --range=192.168.48.0/20
      * gcloud compute networks subnets create griffin-prod-mgmt --network=griffin-prod-vpc --region=us-east1 --range=192.168.64.0/20
    - Solution through Console
      * Navigation Menu > VPC network > VPC networks > Click on Create VPC network.
      * Enter griffin- prod -vpc to the Name field.
      * Select Custom for the Subnet creation mode.
      * Add griffin- prod -wp subnet with the following parameters:
        + Field Value
        + Name: griffin- prod -wp
        + Region: us-east1
        + IP address range: 192.168. 48.0/20
      * Click + Add subnet and add griffin- prod -mgmt subnet with the following parameters
        + Field Value
        + Name: griffin- prod -mgmt
        + Region: us-east1
        + IP address range: 192.168. 64.0/20
      * Click Create.
* Task 3: Create bastion host
  + Create a bastion host with two network interfaces, one connected to griffin-dev-mgmt and the other connected to griffin-prod-mgmt. Make sure you can SSH to the host.
  + Solution
    - Navigation Menu > Compute Engine > VM instances > Click Create.
      * Use the following parameters to create the bastion host:
        + Field Value
        + Name: griffin-dev-db
        + Region: us-east1
      * Expand the Management, security, disks, networking, sole tenancy section.
        + In the Networking tab, add bastion to the Network tags.
        + Click Add network interface, make sure that you set up two Network interfaces,

griffin-dev-mgmt

griffin-prod-mgmt

* + - * + Click Create.
    - Navigation Menu > Navigate to VPC network > Firewall > Click CREATE FIREWALL RULE.
      * Configure the rule with the following parameters:
        + Field Value
        + Name: allow-bastion-dev-ssh
        + Network: griffin-dev-vpc
        + Targets: bastion
        + Source IP ranges: 192.168.32.0/20
        + Protocols and ports: tcp: 22
      * Click CREATE.
    - Click CREATE FIREWALL RULE again.
      * Configure another rule with the following parameters:
        + Field Value
        + Name: allow-bastion-prod-ssh
        + Network: griffin-prod-vpc
        + Targets: bastion
        + Source IP ranges: 192.168.48.0/20
        + Protocols and ports: tcp: 22
      * Click CREATE.
* Task 4: Create and configure Cloud SQL Instance
  + Create a MySQL Cloud SQL Instance called griffin-dev-db in us-east1. Connect to the instance and run the following SQL commands to prepare the WordPress environment:
    - CREATE DATABASE wordpress;
    - GRANT ALL PRIVILEGES ON wordpress.\* TO "wp\_user"@"%" IDENTIFIED BY "stormwind\_rules";
    - FLUSH PRIVILEGES;
  + These SQL statements create the worpdress database and create a user with access to the wordpress dataase.
  + You will use the username and password in task 6.
  + Solution
    - Navigation Menu > SQL > Click CREATE INSTANCE > Click Choose MySQL.
      * Use the following parameters to create the instance:
        + Field Value
        + Name: griffin-dev-db
        + Region: us-east1
        + Zone: us-east1-b
        + Root password: password
      * Click Create.
    - Click the griffin-dev-db in the SQL pane after it has been created.
      * Under Connect to this instance, click on Connect using Cloud Shell.
    - Go back to the Cloud Shell, run:
      * gcloud sql connect griffin-dev-db --user=root
        + Enter the Root password generated in Step 4 (password).
    - In the SQL console, run the following query to create the wordpress database:
      * CREATE DATABASE wordpress;
      * GRANT ALL PRIVILEGES ON wordpress.\* TO "wp\_user"@"%" IDENTIFIED BY "stormwind\_rules";
      * FLUSH PRIVILEGES;
* Task 5: Create Kubernetes cluster
  + Create a 2 node cluster (n1-standard-4) called griffin-dev, in the griffin-dev-wp subnet, and in zone us-east1-b.
  + Solution
  + Navigation Menu > Kubernetes Engine > Clusters > Click Create cluster.
    - In the Cluster basics tab, configure:
      * Name: griffin-dev
      * Zone: us-east1-b
      * Number of nodes: 2
      * Machine type: n1-standard-4
    - Go to the Network tab, set
      * Network: griffin-dev-vpc
      * Node subnet: griffin-dev-wp
    - Click CREATE.
* Task 6: Prepare the Kubernetes cluster
  + Use Cloud Shell and copy all files from gs://cloud-training/gsp321/wp-k8s.
  + The WordPress server needs to access the MySQL database using the username and password you created in task 4. You do this by setting the values as secrets. WordPress also needs to store its working files outside the container, so you need to create a volume.
  + Add the following secrets and volume to the cluster using wp-env.yaml. Make sure you configure the username to wp\_user and password to stormwind\_rules before creating the configuration.
  + You also need to provide a key for a service account that was already set up. This service account provides access to the database for a sidecar container. Use the command below to create the key, and then add the key to the Kubernetes environment.
    - gcloud iam service-accounts keys create key.json --iam-account=cloud-sql-proxy@$GOOGLE\_CLOUD\_PROJECT.iam.gserviceaccount.com
    - kubectl create secret generic cloudsql-instance-credentials --from-file key.json
  + Solution
    - The Cloud Shell, run the cmd
      * gsutil cp -r gs://cloud-training/gsp321/wp-k8s ~/
    - Open wp-k8s/wp-env.yaml with the Cloud Shell Editor.
      * vi wp-k8s/wp-env.yaml
        + Type “I”
        + Replace username\_goes\_here and password\_goes\_here to wp\_user and stormwind\_rules, respectively.
        + Enter “esc”, type “:wq” enter to save the file.
    - After the Kubernetes cluster has been created, click on the Connect button. Get the command from the pop up window.
      * Run the command to connect the cluster:
        + gcloud container clusters get-credentials griffin-dev --zone=us-east1
    - Deploy the configuration to the cluster using:
      * kubectl create -f wp-k8s/wp-env.yaml
    - Use the command below to create the key, and then add the key to the Kubernetes environment:
      * gcloud iam service-accounts keys create key.json --iam-account=cloud-sql-proxy@$GOOGLE\_CLOUD\_PROJECT.iam.gserviceaccount.com
      * kubectl create secret generic cloudsql-instance-credentials --from-file key.json
* Task 7: Create a WordPress deployment
  + Now you have provisioned the MySQL database, and set up the secrets and volume, you can create the deployment using wp-deployment.yaml. Before you create the deployment you need to edit wp-deployment.yaml and replace YOUR\_SQL\_INSTANCE with griffin-dev-db's Instance connection name. Get the Instance connection name from your Cloud SQL instance.
  + After you create your WordPress deployment, create the service with wp-service.yaml.
  + Once the Load Balancer is created, you can visit the site and ensure you see the WordPress site installer. At this point the dev team will take over and complete the install and you move on to the next task.
  + Solution
    - Run the cmd in cloud shell
    - vi wp-k8s/wp-deployment.yaml
      * Type “I”
      * Replace YOUR\_SQL\_INSTANCE with griffin-dev-db’s Instance connection name.
      * Enter “esc”, type “:wq” enter to save the file.
    - kubectl create -f wp-k8s/wp-deployment.yaml
    - kubectl create -f wp-k8s/wp-service.yaml
    - kubectl get service
    - Copy the External endpoints of the deployed wordpress service and open it in your browser.
* Task 8: Enable monitoring
  + Create an uptime check for your WordPress development site.
  + Solution
    - Navigation Menu > Monitoring > click Uptime checks > Click CREATE UPTIME CHECK.
    - Configure using the following parameters:
      * Field Value
      * Title WordPress Uptime
      * Check Type HTTP
      * Resource Type URL
      * Hostname YOUR-WORDPRESS\_ENDPOINT
      * Path /
    - Click Test and Click Save if there is no error
* Task 9: Provide access for an additional engineer
  + You have an additional engineer starting and you want to ensure they have access to the project, so please go ahead and grant them the editor role to the project.
  + The second user account for the lab represents the additional engineer.
  + Solution
    - Navigation > IAM & Admin > IAM > Click +ADD > Add email id > Role > select Editor > Click SAVE.