## Getting Started with Google Kubernetes Engine

* Introduction to Google Cloud
  + After a review of fundamentals cloud computing, learn about how GC's resources around the world are organized into regions and zones. learn how you can organize the resources you use in GC, so that you can manage them. Finally, the tools that let you connect to GC and allocate, change, and release resources.
* Cloud Computing and Google Cloud 4 minutes - https://youtu.be/s1OIH6En8i0
  + Which of these GC compute services provides environments for execution of code, in which users don't have to worry about infrastructure management? Choose all that are correct (2 correct answers).
    - App Engine, Cloud Functions
  + Which statements are true about cloud computing? Mark all that are true (2 correct answers).
    - Customers pay for the resources they use or reserve.
    - Customers who need more resources can get them rapidly
* Resource Management 7 minutes - https://youtu.be/puGMO6XszPM
  + Within which of these GC geographic scopes are network latencies generally less than 1 millisecond? Choose all that are correct (2 correct answers).
    - Zone, Region
  + What is the base-level organizing entity for creating and using Google Cloud resources and services?
    - Project
  + What type of resource is a Compute Engine virtual machine?
    - Zonal
* Billing 4 minutes - https://youtu.be/1ThUMciWb0w
  + Which type of quota resets at regular intervals?
    - Rate quotas
  + At what level in the Google Cloud resource hierarchy is billing set up?
    - Project
* Interacting with Google Cloud 4 minutes - https://youtu.be/fXJKfXmBazI
  + Which of these ways to interact with give you access to the gcloud and kubectl commands? 2 correct ans.
    - Cloud SDK, Cloud Shell
* Accessing the Google Cloud Console and Cloud Shell 1 hour - <https://www.cloudskillsboost.google/course_sessions/809399/labs/190654>
  + Overview
    - you become familiar with the GC web-based interface. Two integrated environments are available:
      * A GUI environment called the Google Cloud Console
      * A CLI called Cloud Shell, which has the commands from the Cloud SDK pre-installed
  + You need to know a few things about the Google Cloud Console:
    - You can perform most common GC actions in the GC Console. Sometimes new features are implemented in the Cloud SDK before they are made available in the GC Console.
    - The GC is extremely fast for some activities. The Google Cloud Console can perform multiple actions on your behalf that might require many command-line actions.
    - The commands in the Cloud SDK are valuable tools for automation.
  + Objectives
    - In this lab, you learn how to perform the following tasks:
      * Learn how to access the Google Cloud Console and Cloud Shell
      * Become familiar with the Google Cloud Console
      * Become familiar with Cloud Shell features, including the Cloud Shell code editor
      * Use the GC Console and Cloud Shell to create buckets and VMs and service accounts
      * Perform other commands in Cloud Shell
  + Navigate to Google Cloud Storage and create a bucket
    - Navigation menu > Cloud Storage > Browser> Click Create bucket.
    - Click on Choose how to control access to objects and uncheck Enforce public access prevention on this bucket, now select Fine-grained. Click Create.
  + Create a virtual machine (VM) instance
    - For Name, type first-vm as the name for your instance.
    - For Region, select us-central1. For Zone, select us-central1-c.
    - For Machine type, click e2-micro (2 shared vCPU).
      * The micro type is a shared-core VM that is inexpensive.
  + Create an IAM service account
    - Navigation > IAM & admin > Service accounts. > + Create service account.
    - On the Grant this service account access to project page, specify the role as Project > Editor.
    - Click Continue. Click Done.
    - click on the three dots >Manage keys > ADD KEY > Create new key > Select JSON > Click Create.
      * A JSON key file is downloaded.
  + In Cloud Shell, use the following commands to define the environment variables used in this task.
    - Replace [BUCKET\_NAME] with the name of the first bucket from task 1.
    - Replace [BUCKET\_NAME\_2] with a globally unique name.
    - MY\_BUCKET\_NAME\_1=[BUCKET\_NAME]
    - MY\_BUCKET\_NAME\_2=[BUCKET\_NAME\_2]
    - MY\_REGION=us-central1
  + Move the credentials file you created earlier into Cloud Shell
    - On your local workstation, locate the downloaded the JSON key & rename to credentials.json.
    - In Cloud Shell, click the three dots icon, Click Upload the credentials.json file from your local machine to the Cloud Shell VM.
  + Create a second Cloud Storage bucket and verify it in the Google Cloud Console
    - gsutil mb gs://$MY\_BUCKET\_NAME\_2
  + Use the gcloud command line to create a second virtual machine
    - gcloud compute zones list | grep $MY\_REGION
    - MY\_ZONE=[ZONE]
    - gcloud config set compute/zone $MY\_ZONE
    - MY\_VMNAME=second-vm
    - gcloud compute instances create $MY\_VMNAME --machine-type "e2-standard-2" --image-project "debian-cloud" --image-family "debian-9" --subnet "default"
  + Use the gcloud command line to create a second service account
    - gcloud iam service-accounts create test-service-account2 --display-name "test-service-account2"
    - In Cloud Shell, run the cmd to grant the second service account the Project viewer role:
      * gcloud projects add-iam-policy-binding $GOOGLE\_CLOUD\_PROJECT --member serviceAccount:test-service-account2@${GOOGLE\_CLOUD\_PROJECT}.iam.gserviceaccount.com --role roles/viewer
  + Download a file to Cloud Shell and copy it to Cloud Storage
    - Copy a picture of a cat from a Google-provided Cloud Storage bucket to your Cloud Shell.
      * gsutil cp gs://cloud-training/ak8s/cat.jpg cat.jpg
    - Copy the file into one of the buckets that you created earlier.
      * gsutil cp cat.jpg gs://$MY\_BUCKET\_NAME\_1
    - Copy the file from the first bucket into the second bucket:
      * gsutil cp gs://$MY\_BUCKET\_NAME\_1/cat.jpg gs://$MY\_BUCKET\_NAME\_2/cat.jpg
  + To get the default access list that's been assigned to cat.jpg (storage bucket), run the commands:
    - gsutil acl get gs://$MY\_BUCKET\_NAME\_1/cat.jpg > acl.txt
    - cat acl.txt
    - The output .

[

{

"entity": "project-owners-560255523887",

"projectTeam": {

"projectNumber": "560255523887",

"team": "owners"

},

"role": "OWNER"

},

{

"entity": "project-editors-560255523887",

"projectTeam": {

"projectNumber": "560255523887",

"team": "editors"

},

"role": "OWNER"

},

{

"entity": "project-viewers-560255523887",

"projectTeam": {

"projectNumber": "560255523887",

"team": "viewers"

},

"role": "READER"

},

{

"email": "google12345678\_student@qwiklabs.net",

"entity": "user-google12345678\_student@qwiklabs.net",

"role": "OWNER"

}

]

* + To change the object to have private access, execute the following command:
    - gsutil acl set private gs://$MY\_BUCKET\_NAME\_1/cat.jpg
  + To verify the new ACL that's been assigned to cat.jpg, execute the following two commands:
    - gsutil acl get gs://$MY\_BUCKET\_NAME\_1/cat.jpg > acl-2.txt
    - cat acl-2.txt
    - The output, the original creator of the object (your lab account) has Owner access.

[

{

"email": "google12345678\_student@qwiklabs.net",

"entity": "user-google12345678\_student@qwiklabs.net",

"role": "OWNER"

}

]

* + In Cloud Shell, execute the following command to view the current configuration:
    - gcloud config list
    - output.

[component\_manager]

disable\_update\_check = True

[compute]

gce\_metadata\_read\_timeout\_sec = 5

zone = us-central1-a

[core]

account = google12345678\_student@qwiklabs.net

disable\_usage\_reporting = False

project = qwiklabs-Google Cloud-1aeffbc5d0acb416

[metrics]

environment = devshell

Your active configuration is: [cloudshell-16441]

* + In Cloud Shell, change the authenticated user to the first service account (which you created in an earlier task) through the credentials that you downloaded to your local machine and then uploaded into Cloud Shell (credentials.json).
    - gcloud auth activate-service-account --key-file credentials.json
  + Cloud Shell is now authenticated as test-service-account. To verify the active account:
    - gcloud config list
    - The account is now set to the test-service-account service account.

[component\_manager]

disable\_update\_check = True

[compute]

gce\_metadata\_read\_timeout\_sec = 5

zone = us-central1-a

[core]

account = test-service-account@qwiklabs-Google Cloud-1aeffbc5d0acb416.iam.gserviceaccount.com

disable\_usage\_reporting = False

project = qwiklabs-Google Cloud-1aeffbc5d0acb416

[metrics]

environment = devshell

Your active configuration is: [cloudshell-16441]

* + To verify the list of authorized accounts in Cloud Shell, execute the following command:
    - gcloud auth list
    - You should see output that looks like the following example.

Credentialed Accounts

ACTIVE ACCOUNT

google12345678\_student@qwiklabs.net

\* test-service-account@qwiklabs-Google Cloud-1aeffbc5d0acb416.iam.gserviceaccount.com

To set the active account, run:

$ gcloud config set account `ACCOUNT`

* + To verify that the current account (test-service-account) cannot access the cat.jpg file in the 1st bucket
    - * gsutil cp gs://$MY\_BUCKET\_NAME\_1/cat.jpg ./cat-copy.jpg
      * Because you restricted access to this file to the owner earlier in this task you should see output that looks like the following example.
      * Output (do not copy)

Copying gs://test-bucket-123/cat.jpg...

AccessDeniedException: 403 KiB]

* + Verify that the current account (test-service-account) can access the cat.jpg file in the second bucket:
    - gsutil cp gs://$MY\_BUCKET\_NAME\_2/cat.jpg ./cat-copy.jpg
    - Because access has not been restricted to this file, see output.

Copying gs://test-bucket-123/cat.jpg...

- [1 files][ 81.7 KiB/ 81.7 KiB]

Operation completed over 1 objects/81.7 KiB.

* + To switch to the lab account, execute the following command.
    - gcloud config set account [USERNAME]
  + To verify that you can access the cat.jpg file in the [BUCKET\_NAME] first bucket .
    - gsutil cp gs://$MY\_BUCKET\_NAME\_1/cat.jpg ./copy2-of-cat.jpg
  + see output, The lab account created the bucket and object and remained an Owner when the object access control list (ACL) was converted to private, so the lab account can still accss the object.

Copying gs://test-bucket-123/cat.jpg...

- [1 files][ 81.7 KiB/ 81.7 KiB]

Operation completed over 1 objects/81.7 KiB.

* + Make the first Cloud Storage bucket readable by everyone, including unauthenticated users.
    - gsutil iam ch allUsers:objectViewer gs://$MY\_BUCKET\_NAME\_1
    - Note: this is an appropriate setting for hosting public website content in Cloud Storage.
  + in Cloud Shell, execute the following command to clone a git repository:
    - git clone https://github.com/googlecodelabs/orchestrate-with-kubernetes.git
  + In the Cloud Shell code editor window, Add the following text as the last line of the cleanup.sh file:
    - echo Finished cleanup!
  + Create index.html.
    - <html><head><title>Cat</title></head>
    - <body>
    - <h1>Cat</h1>
    - <img src="REPLACE\_WITH\_CAT\_URL">
    - </body></html>
  + On the Navigation menu > Compute Engine > VM instances > first VM > SSH button.
    - In the SSH login window that opens on your VM, install the nginx Web server:
      * sudo touch /var/lib/man-db/auto-update
      * sudo apt-get update
      * sudo apt-get install nginx
    - gcloud compute scp index.html first-vm:index.nginx-debian.html --zone=us-central1-c
      * enter no password
  + In the SSH login , copy the HTML file from your home directory to the document root of the nginx Web server:
    - sudo cp index.nginx-debian.html /var/www/html
  + Introduction to Google Cloud
    - You are developing a new product for a customer and need to implement control structures in Google Cloud to help manage the Google Cloud resources consumed by the product and the billing for the customer account. What steps should you take to manage costs for this product and customer?
      * Configure the billing account at the product folder level in the resource hierarchy.
    - You are ready to start work building an application in Google Cloud. What Cloud IAM hierarchy should you implement for this project?
      * Create a new folder inside your organization and create projects inside that folder for the resources.
    - You need to write some automated scripts to run periodic updates to the resources in your Google Cloud environment. What tools can you install in your own computers to allow you to run those scripts?
      * The Cloud SDK
    - One of the key characteristics of cloud computing is the concept of measured service. What is the primary customer benefit of the measured service aspect of cloud computing?
    - You pay only for the resources you consume.
  + You are considering deploying a solution using containers on Google Cloud. What Google Cloud solutions are available to you that will provide a managed compute platform with native support for containers?
    - Google Kubernetes Engine Clusters
* Introduction to Google Cloud - https://www.cloudskillsboost.google/course\_sessions/809399/video/190655
* Introduction to Containers and Kubernetes
  + learn about software containers and their benefits are for application deployment. You'll configure and build containers. learn about the functions container that management solutions like K8s provide. You'll encounter the advantages of GKE compared to building your own container-management infrastructure.
* Introduction to Containers 5 minutes - https://youtu.be/DaQ8irwtq2o
  + Which of these problems are containers intended to solve? Mark all that are correct (3 correct answers),
    - Packaging applications in virtual machines can be wasteful.
    - Applications need a way to isolate their dependencies from one another.
    - It's difficult to troubleshoot applications when they work on a developer's laptop but fail in production.
* Containers and Container Images 7 minutes - <https://youtu.be/mJNbM2EzRnY>
  + What is significant about the topmost layer in a container? Choose all that are true (2 correct answers).
    - An application running in a container can only modify the topmost layer.
    - The topmost layer's contents are lost when the container is no longer running.
  + Why do Linux containers use union file systems?
    - To efficiently encapsulate applications and their dependencies into a set of clean, minimal layers
* Working with Cloud Build 1 hour - https://youtu.be/FYFrffeygBc
  + Overview
    - build a Docker container image from provided code and a Dockerfile using Cloud Build. You will then upload the container to Container Registry.
  + Objectives
    - In this lab, you learn how to perform the following tasks:
    - Use Cloud Build to build and push containers
    - Use Container Registry to store and deploy containers
  + the Navigation menu > APIs & Services >Enable Cloud Build & Container Registry API
  + Task 2. Building Containers with DockerFile and Cloud Build
    - write build configuration files to provide instructions to Cloud Build as to which tasks to perform when building a container. These build files can fetch dependencies, run unit tests, analyses and more. In this task, create a DockerFile and use it as a build configuration script with Cloud Build. create a simple shell script (quickstart.sh) which will represent an application inside the container.
  + In Activate Cloud Shell.
    - nano quickstart.sh
      * #!/bin/sh
      * echo "Hello, world! The time is $(date)."
    - Save the file and close nano by pressing the CTRL+X key, then press Y and Enter.

nano Dockerfile

* + - * FROM alpine
        + This instructs the build to use the Alpine Linux base image.
      * COPY quickstart.sh /
        + This adds the quickstart.sh script to the / directory in the image.
      * CMD ["/quickstart.sh"]
        + This configures the image to execute the /quickstart.sh script when the associated container is created and run.
      * The Dockerfile should now look like:

FROM alpine

COPY quickstart.sh /

CMD ["/quickstart.sh"]

* + - Save the file and close nano by pressing the CTRL+X key, then press Y and Enter.
      * chmod +x quickstart.sh
    - In Cloud Shell, run the following command to build the Docker container image in Cloud Build.
      * gcloud builds submit --tag gcr.io/${GOOGLE\_CLOUD\_PROJECT}/quickstart-image .
        + Don't miss the dot (".") at the end of the command. The dot specifies that the source code is in the current working directory at build time.
    - When the build completes, your Docker image is built and pushed to Container Registry.
      * on the Navigation > click Container Registry > Images.
  + Task 3. Building Containers with a build configuration file and Cloud Build
    - Cloud Build also supports custom build configuration files. In this task you will incorporate an existing Docker container using a custom YAML-formatted build file with Cloud Build.
      * git clone https://github.com/GoogleCloudPlatform/training-data-analyst
    - Create a soft link as a shortcut to the working directory.
      * ln -s ~/training-data-analyst/courses/ak8s/v1.1 ~/ak8s
    - Change to the directory that contains the sample files for this lab.
      * cd ~/ak8s/Cloud\_Build/a
    - A sample custom cloud build configuration file called cloudbuild.yaml & Dockerfile has been provided in this directory and the quickstart.sh script you created in the first task.
      * cat cloudbuild.yaml

steps:

- name: 'gcr.io/cloud-builders/docker'

args: [ 'build', '-t', 'gcr.io/$PROJECT\_ID/quickstart-image', '.' ]

images:

- 'gcr.io/$PROJECT\_ID/quickstart-image'

* + - This file instructs Cloud Build to use Docker to build an image using the Dockerfile specification in the current local directory, tag it with gcr.io/$PROJECT\_ID/quickstart-image and then push that image to Container Registry.
    - Cloud Shell, run the cmd to start a Cloud Build using cloudbuild.yaml as the build configuration file:
      * gcloud builds submit --config cloudbuild.yaml .
    - The build output to Cloud Shell should be the same as before. When the build completes, a new version of the same image is pushed to Container Registry.
    - Navigation > Container Registry > Images> click quickstart-image > 2 versions of image in the list.
    - Navigation > Cloud Build > History > 2 builds appear > Click the build ID.
      * The details of the build, including the build log, are displayed.
  + Task 4. Building and Testing Containers with a build configuration file and Cloud Build
    - The true power of custom build configuration files is their ability to perform other actions, in parallel or in sequence, in addition to simply building containers: running tests on your newly built containers, pushing them to various destinations, and even deploying them to Kubernetes Engine. In this lab, we will see a simple example: a build configuration file that tests the container it built and reports the result to its calling environment.
      * cd ~/ak8s/Cloud\_Build/b
    - cloudbuild.yaml, slightly modified to demonstrate Cloud Build's ability to test the containers it has build. Run the cmd
      * cat cloudbuild.yaml
        + You will see the following:

steps:

- name: 'gcr.io/cloud-builders/docker'

args: [ 'build', '-t', 'gcr.io/$PROJECT\_ID/quickstart-image', '.' ]

- name: 'gcr.io/$PROJECT\_ID/quickstart-image'

args: ['fail']

images:

- 'gcr.io/$PROJECT\_ID/quickstart-image

* + - * the quickstart.sh script has been modified so that it simulates a test failure when an argument ['fail'] is passed to it.
    - Cloud Shell, run the cmd to start a Cloud Build using cloudbuild.yaml as the build configuration file:
      * gcloud builds submit --config cloudbuild.yaml .
        + Output (do not copy)

Finished Step #1

ERROR

ERROR: build step 1 "gcr.io/ivil-charmer-227922klabs-gcp-49ab2930eea05/quickstart-image" failed: exit status 127

----------------------------------------------------------------------------------------------------------------------------------------------------------------

ERROR: (gcloud.builds.submit) build f3e94c28-fba4-4012-a419-48e90fca7491 completed with status "FAILURE"

* + - Confirm that your command shell knows that the build failed:
      * echo $?
* Introduction to Kubernetes 3 minutes - https://youtu.be/Z9MkNbujhBQ
  + When you use Kubernetes, you describe the desired state you want, and Kubernetes's job is to make the deployed system conform to your desired state and to keep it there in spite of failures. What is the name for this management approach?
    - Declarative configuration
  + What is a stateful application?
    - An application that requires data to be stored persistently required
* Introduction to Google Kubernetes Engine 3 minutes - https://youtu.be/80jCygkcwNM
  + What is the relationship between Kubernetes and Google Kubernetes Engine?
    - Google Kubernetes Engine is Kubernetes as a managed service.
  + What is the name for the computers in a Kubernetes cluster that can run your workloads?
    - Nodes
  + Which of the following supports scaling a Kubernetes cluster as a whole?
    - Google Kubernetes Engine
* Compute Options Detail 10 minutes - https://youtu.be/gDsi1XlIeik
* Containers and Kubernetes in Google Cloud
  + You are developing a new solution and want to explore serverless application solutions. Which Google Cloud compute services provide serverless compute resources that you can use with containers?
    - check
  + Google Compute Engine provides fine-grained control of costs. Which Compute Engine features provide this level of control?
    - Per-second billing
    - Fully customizable virtual machines
  + You are deploying a containerized application, and you want maximum control over how containers are configured and deployed. You want to avoid the operational management overhead of managing a full container cluster environment yourself. Which Google Cloud compute solution should you choose?
    - Google Kubernetes Engine
  + You are classifying a number of your applications into workload types. Select the stateful applications in this list of applications. Choose all responses that are correct (2 correct responses).
    - A gaming application that keeps track of user state persistently.
    - A shopping application that saves user shopping cart data between sessions.
  + You are choosing a technology for deploying applications, and you want to deliver them in lightweight, standalone, resource-efficient, portable packages. Which choice best meets those goals?
    - Containers
* Kubernetes Architecture
  + learn the components of a K8s cluster and how they work together. You'll deploy a K8s cluster using GKE and deploy Pods to a GKE cluster. You'll also view and manage several very useful kinds of K8s objects.
* Kubernetes Concepts 4 minutes - <https://youtu.be/E80GnxIH0yY>
  + What is the difference between a pod and a container?
    - A pod contains one or more containers.
* Kubernetes Control Plane 6 minutes - <https://youtu.be/JxNaMYY0j-I>
  + What is the role of the kubelet?
    - To serve as Kubernetes’s agent on each node
  + Which control plane component is the cluster's database?
    - etcd
  + Which control plane component is the only one with which clients interact directly?
  + kube-apiserver
* Google Kubernetes Engine Concepts 6 minutes - <https://youtu.be/6twk0vTNHrM>
  + What is the purpose of configuring a regional cluster in GKE?
    - To allow applications running in the cluster to withstand the loss of a zone
  + In GKE, how are control planes provisioned?
    - As abstract parts of the GKE service that are not exposed to Google Cloud customers
  + In GKE clusters, how are nodes provisioned?
    - As Compute Engine virtual machines
* Kubernetes Object Management 10 minutes - https://youtu.be/592WGkUQ1eU
  + In a manifest file for a Pod, in which field do you define a container image for the Pod?
    - spec
  + What is the purpose of the Deployment object?
    - To ensure that a defined set of Pods is running at any given time.
  + What are Kubernetes namespaces useful for? Choose all that are correct (2 correct answers).
    - Namespaces let you implement resource quotas across your cluster.
    - Namespaces allow you to use object names that would otherwise be duplicates of one another.
* A note about Services
* Controller objects to know about
* Kubernetes Controller Objects
  + What is the purpose of a Service? Choose all that are true (2 correct answers)
    - To allow you to choose how Pods are exposed
    - To provide a load-balancing network endpoint for Pods
  + If you are deploying applications in your Pods that need persistent storage, which controller type should you use?
    - StatefulSet
* Deploying Google Kubernetes Engine 1 hour
  + Overview
    - In this lab, you use the Google Cloud Console to build GKE clusters and deploy a sample Pod.
  + Objectives, In this lab, you learn how to perform the following tasks:
    - Use the Google Cloud Console to build and manipulate GKE clusters
    - Use the Google Cloud Console to deploy a Pod
    - Use the Google Cloud Console to examine the cluster and Pods
  + Task 1. Deploy GKE clusters, Use the Google Cloud Console to deploy a GKE cluster
    - Navigation > Kubernetes Engine > Clusters > Create > GKE Standard cluster > Configure
    - Cluster name: standard-cluster-1, zone : us-central1-a, Leave rest defaults and click Create.
    - Once it created, Click the cluster name standard-cluster-1 to view the cluster details
  + Task 2. Modify GKE clusters
    - Click NODES for standard-cluster-1.
    - Node Pools section >click default-pool > click RESIZE > Change the nodes from 3 to 4 > click RESIZE.
  + Task 3. Deploy a sample workload
    - In this task, you will deploy a Pod running the nginx web server as a sample workload.
    - Navigation > K8s Engine > Workloads > Deploy > Continue to accept the default container image, nginx:latest, which deploys 3 Pods each with a single container running the latest version of nginx.
    - click the Deploy button leaving the Configuration details at the defaults.
  + Task 4. View details about workloads in the Google Cloud Console
    - Kubernetes Engine > Workloads page, click nginx-1.
    - This displays the overview information for the workload showing details like resource utilization charts, links to logs, and details of the Pods associated with this workload.
    - The Details tab shows more details about the workload including the Pod specification, number and status of Pod replicas and details about the horizontal Pod autoscaler.
    - the Revision History tab displays a list of the revisions that have been made to this workload.
    - the Events tab lists events associated with this workload.
    - the YAML tab provides the complete YAML file that defines this components and full configuration of this sample workload.
    - Details tab for the nginx-1 workload > click the Overview tab > scroll down to the Managed Pods section > click the name of one of the Pods to view the details page for that Pod.
      * The Pod Details page provides information on the Pod configuration and resource utilization and the node where the Pod is running.
      * click the Events and Logs tabs to view event details and links to container logs in Cloud Operations.
      * Click the YAML tab to view the detailed YAML file for the Pod configuration.
* Lab solution 4 minutes - https://youtu.be/J2P6Kfbz2mc
* Migrate for Anthos introduction 1 minute
  + But what if you've got existing applications, that are not in containers, or perhaps not even in the cloud?
  + Migrate for Anthos, is our tool for getting workloads into containerized deployments on Google Cloud.
  + what Migrate for Anthos does.
    - moves your existing applications into a Kubernetes environment. The process is automated.
    - Your workloads can be on premises, or in other cloud providers.
    - migrating your applications data in one move, or stream it to the cloud until the application is live.
* Migrate for Anthos Architecture 1 minute - <https://youtu.be/RA25_qPuhsg>
  + Graphical user interface

    Description automatically generated
* Migration Path 1 minute - <https://youtu.be/orMIPjWjPdA>
  + A picture containing diagram

    Description automatically generated
  + First you need to create the processing cluster.
    - After that you install the Migrate For Anthos components onto that cluster.
  + Next you need to add a migration source.
    - You can migrate from VMware, AWS, Azure or Google cloud.
  + You will need to create a migration object with the details of the migration that you're performing.
    - This will generate a plan template for you in a YAML file.
    - You may need to alter this configuration file to create the level of customization that you desire.
  + When the plan is ready, you will need to generate the artifacts for the migration.
    - This means generation that container images of your applications on the YAML files required for the deployment.
  + After your migration artifacts have been generated, they need to be tested.
    - Both the container images and the deployments will be tested at this stage.
  + Finally, if the tests are successful, you can use the generative artifacts to deploy your application in to your production clusters.
* Migrate for Anthos Installation 3 minutes - <https://youtu.be/XyGSNfBTt-s>
  + - Graphical user interface, text

      Description automatically generated
  + set up the processing cluster run the cmd, you are a gke admin, you also must have firewall rules in place to allow communications between migrate for anthos and migrate for a compute engine, after all that's done you can create the processing cluster the example on screen enables a vpc native cluster
    - Graphical user interface

      Description automatically generated
  + when the processing cluster is up and running you need to install migrate for amphos using the mig ctl cmd. This cmd installs all of the required k8s resources onto the processing cluster for the migration
    - Graphical user interface, text, application

      Description automatically generated
    - the mig ctl source create command specifies the location of the application to migrate. The example on screen is for migrating from google compute engine. if you are migrating from a vmware backend or another cloud provider you need to install some additional packages Graphical user interface, text, application

      Description automatically generated
  + the next step is to create a migration plan. the mig ctl migration create command will create the migration plan. this command will define the migration resources to be created on the cluster, you identify the source vm and what data to exclude from the migration, you can also specify what migration intents you want, you can specify the following intents image image and data, data or pv based container the output of this command is a yaml file that can be further customized
    - Graphical user interface, application

      Description automatically generated
  + generate the artifacts for the migration. the mig ctl migration generate artifacts command on screen will start this process. this process will first copy files and directories representing the vm to a container image registry as images. migrate for amphos creates two images a runnable image for deployment into another cluster and a non-runnable image layer that can be used to update the container image in the future. next migrate for amthos will generate configuration yaml files that you can use to deploy the vm to another gke cluster these are copied into a cloud storage bucket as an intermediate location
    - Graphical user interface, text, application, email

      Description automatically generated
  + you run the mig ctl migration get artifacts command to download the yaml configuration files generated from the last step the yaml configuration defines the resources to deploy such as are you creating a deployment or a stateful set is the deployment a headless service are you using persistent volumes or persistent volume claims you can edit the yaml file to customize your deployment examples of customizations include enabling load balancing allowing ingress or defining this size
    - Table

      Description automatically generated
  + finally you run the cubectl apply command to deploy the defined specification
* Kubernetes Architecture
  + Which Kubernetes component does the kubectl command connect to in order to carry out operations on a cluster?
    - kube-apiserver
  + You have a new logging and auditing utility that you need to deploy on all of the nodes within your cluster. Which type of controller should you use to handle this task?
    - DaemonSet
  + You need to ensure that the production applications running on your Kubernetes cluster are not impacted by test and staging deployments. Which features should you implement and configure to ensure that the resources for your production applications can be prioritized?
    - Configure Namespaces for Test, Staging and Production and configure specific Kubernetes resource quotas for the test and staging Namespaces.
  + When configuring storage for stateful applications, what steps must you take to provide file system storage inside your containers for data from your applications that will not be lost or deleted if your Pods fail or are deleted for any reason?
    - You must create Volumes using network based storage to provide durable storage remote to the Pods and specify these in the Pods.
  + You want to deploy multiple copies of your application, so that you can load balance traffic across them. How should you deploy this application's Pods to the production Namespace in your cluster?
    - Create a Deployment manifest that specifies the number of replicas that you want to run.
  + You are designing an application, and you want to ensure that the containers are located as close to each other as possible, in order to minimize latency. Which design decision helps meet this requirement?
    - Place the containers in the same Pod.
    - That is correct. Placing containers in the same Pod ensures they are scheduled together on the same node, minimizing latency.
  + You have deployed a new Kubernetes Engine regional cluster with four machines in the default pool for the first zone and left the number of zones at the default. How many Compute Engine machines are deployed and billed against your account?
    - Twelve. (Four nodes are deployed in each of three zones. A control plane node is deployed in each zone but it is not billed against your account.)
    - That is correct. GKE Regional clusters are deployed across multiple zones in a single region. Google does not bill you for GKE control plane nodes.
* Introduction to Kubernetes Workloads
  + In this module you’ll learn the components that are used to manage Kubernetes workloads. You'll learn about the kubectl command and creating Kubernetes workloads called deployments. You'll examine networking services within Kubernetes and the different types of Kubernetes storage abstractions.
* The kubectl Command 6 minutes - https://youtu.be/0u47ptrk7kQ
* Deployments 4 minutes - https://youtu.be/JM6IZnh2z20
* Ways to Create Deployments 2 minutes - https://youtu.be/1NeUoatBXGs
* Services and Scaling 1 minute - https://youtu.be/o-PbICQvt68
* Updating Deployments 1 minute - https://youtu.be/U7su0JxRg8w
* Blue-Green Deployments 2 minutes - https://youtu.be/GubUG49tXeI
* Canary Deployments 8 minutes - https://youtu.be/-EkEBtBCNuU
* Managing Deployments 1 minute - https://youtu.be/G\_BE9EItYpM
* Creating Google Kubernetes Engine Deployments 1 hour 8 minutes - https://www.cloudskillsboost.google/course\_sessions/809399/labs/190706
  + Overview
    - explore the basics of using deployment manifests. Manifests are files that contain configurations required for a deployment that can be used across different Pods. Manifests are easy to change.
  + Objectives
    - In this lab, you learn how to perform the following tasks:
    - Create deployment manifests, deploy to cluster, and verify Pod rescheduling as nodes are disabled
    - Trigger manual scaling up and down of Pods in deployments
    - Trigger deployment rollout (rolling update to new version) and rollbacks
    - Perform a Canary deployment
  + Task 1. Create deployment manifests and deploy to the cluster
    - In this task, you create a deployment manifest for a Pod inside the cluster.
    - Connect to the lab GKE cluster
      * In Cloud Shell, run the cmd to set the environment variable for the zone and cluster name.
        + export my\_zone=us-central1-a
        + export my\_cluster=standard-cluster-1
      * Configure kubectl tab completion in Cloud Shell.
        + source <(kubectl completion bash)
      * configure access to your cluster for the kubectl cmd-line tool, using the following cmd:
        + gcloud container clusters get-credentials $my\_cluster --zone $my\_zone
      * In Cloud Shell enter the following command to clone the repository to the lab Cloud Shell.
        + git clone https://github.com/GoogleCloudPlatform/training-data-analyst
      * Create a soft link as a shortcut to the working directory.
        + ln -s ~/training-data-analyst/courses/ak8s/v1.1 ~/ak8s
      * Change to the directory that contains the sample files for this lab.
        + cd ~/ak8s/Deployments/
    - Create a deployment manifest
      * This deployment is configured to run three Pod replicas with a single nginx container in each Pod listening on TCP port 80.

apiVersion: apps/v1

kind: Deployment

metadata:

name: nginx-deployment

labels:

app: nginx

spec:

replicas: 3

selector:

matchLabels:

app: nginx

template:

metadata:

labels:

app: nginx

spec:

containers:

- name: nginx

image: nginx:1.7.9

ports:

- containerPort: 80

* + - To deploy your manifest, execute the following command:
      * kubectl apply -f ./nginx-deployment.yaml
    - To view a list of deployments, execute the following command:
      * kubectl get deployments
  + Task 2. Manually scale up and down the number of Pods in deployments
    - Sometimes, you want to shut down a Pod instance. Other times, you want ten Pods running. In K8s, you can scale a specific Pod to the desired number of instances. To shut them down, you scale to 0.
    - In this task, you scale Pods up and down in the Google Cloud Console and Cloud Shell.
    - Scale Pods up and down in the console
      * Navigation > Kubernetes Engine > Workloads > Click nginx-deployment (your deployment) to open the Deployment details page > click ACTIONS > Scale > Type 1 and click SCALE.
    - To scale the Pod back up to three replicas, execute the following command:
      * kubectl scale --replicas=3 deployment nginx-deployment
  + Task 3. Trigger a deployment rollout and a deployment rollback
    - A deployment's rollout is triggered if and only if the deployment's Pod template (that is, .spec.template) is changed, for example, if the labels or container images of the template are updated. Other updates, such as scaling the deployment, do not trigger a rollout.
    - In this task, you trigger deployment rollout, and then you trigger deployment rollback.
    - Trigger a deployment rollout
      * To update the version of nginx in the deployment, execute the following command:
        + kubectl set image deployment.v1.apps/nginx-deployment nginx=nginx:1.9.1 --record
      * This updates the container image in your Deployment to nginx v1.9.1.
      * To view the rollout status, execute the following command:
        + kubectl rollout status deployment.v1.apps/nginx-deployment
      * View the rollout history of the deployment.
        + kubectl rollout history deployment nginx-deployment
    - Trigger a deployment rollback
      * To roll back an object's rollout, you can use the kubectl rollout undo command.
      * To roll back to the previous version of the nginx deployment, execute the following cmd:
        + kubectl rollout undo deployments nginx-deployment
      * View the updated rollout history of the deployment.
        + kubectl rollout history deployment nginx-deployment
      * View the details of the latest deployment revision
        + kubectl rollout history deployment/nginx-deployment --revision=3
  + Task 4. Define the service type in the manifest
    - In this task, you create and verify a service that controls inbound traffic to an application. Services can be configured as ClusterIP, NodePort or LoadBalancer types. In this lab, you configure a LoadBalancer.
    - Define service types in the manifest
      * A manifest file called service-nginx.yaml that deploys a LoadBalancer service type has been provided for you. This service is configured to distribute inbound traffic on TCP port 60000 to port 80 on any containers that have the label app: nginx.

apiVersion: v1

kind: Service

metadata:

name: nginx

spec:

type: LoadBalancer

selector:

app: nginx

ports:

- protocol: TCP

port: 60000

targetPort: 80

* + - * In the Cloud Shell, to deploy your manifest, execute the following command:
        + kubectl apply -f ./service-nginx.yaml
      * This manifest defines a service and applies it to Pods that correspond to the selector. In this case, the manifest is applied to the nginx container that you deployed in task 1. This service also applies to any other Pods with the app: nginx label, including any that are created after the service.
    - Verify the LoadBalancer creation
      * To view the details of the nginx service, execute the following command:
        + kubectl get service nginx
      * When the external IP appears, open http://[EXTERNAL\_IP]:60000/ in a new browser tab to see the server being served through network load balancing.
  + Task 5. Perform a canary deployment
    - A canary deployment is a separate deployment used to test a new version of your app. A single service targets both the canary and the normal deployments. And it can direct a subset of users to the canary version to mitigate the risk of new releases. The manifest file nginx-canary.yaml that is provided for you deploys a single pod running a newer version of nginx than your main deployment. In this task, you create a canary deployment using this new deployment file.

apiVersion: apps/v1

kind: Deployment

metadata:

name: nginx-canary

labels:

app: nginx

spec:

replicas: 1

selector:

matchLabels:

app: nginx

template:

metadata:

labels:

app: nginx

track: canary

Version: 1.9.1

spec:

containers:

- name: nginx

image: nginx:1.9.1

ports:

- containerPort: 80

* + - The manifest for the nginx Service you deployed in the previous task uses a label selector to target the Pods with the app: nginx label. Both the normal deployment and this new canary deployment have the app: nginx label. Inbound connections will be distributed by the service to both the normal and canary deployment Pods. The canary deployment has fewer replicas (Pods) than the normal deployment, and thus it is available to fewer users than the normal deployment.
    - Create the canary deployment based on the configuration file.
      * kubectl apply -f nginx-canary.yaml
    - Switch back to the browser tab that is connected to the external LoadBalancer service ip and refresh the page. You should continue to see the standard Welcome to nginx page.
    - Switch back to the Cloud Shell and scale down the primary deployment to 0 replicas.
      * kubectl scale --replicas=0 deployment nginx-deployment
    - Verify that the only running replica is now the Canary deployment:
      * kubectl get deployments
    - Switch back to the browser tab that is connected to the external LoadBalancer service ip and refresh the page. You should continue to see the standard Welcome to nginx page showing that the Service is automatically balancing traffic to the canary deployment.
  + Session affinity
    - The service configuration used in the lab does not ensure that all requests from a single client will always connect to the same Pod. Each request is treated separately and can connect to either the normal nginx deployment or to the nginx-canary deployment. This potential to switch between different versions may cause problems if there are significant changes in functionality in the canary release. To prevent this you can set the sessionAffinity field to ClientIP in the specification of the service if you need a client's first request to determine which Pod will be used for all subsequent connections.
    - For example:

apiVersion: v1

kind: Service

metadata:

name: nginx

spec:

type: LoadBalancer

sessionAffinity: ClientIP

selector:

app: nginx

ports:

- protocol: TCP

port: 60000

targetPort: 80

* Pod Networking 4 minutes - https://youtu.be/nQqwHPKPzSo
* Volumes 2 minutes - https://youtu.be/Wl3sLS36wW4
* Volume Types 3 minutes - https://youtu.be/aiq1QeJowok
* The PersistentVolume abstraction 4 minutes - https://youtu.be/lhm6JhNSBHc
* Configuring Persistent Storage for Google Kubernetes Engine 1 hour 4 minutes
  + Overview
    - In this lab, you set up PersistentVolumes and PersistentVolumeClaims. PersistentVolumes are storage that is available to a Kubernetes cluster. PVCs enable Pods to access PVs. Without PVCs Pods are mostly ephemeral, so you should use PVCs for any data that you expect to survive Pod scaling, updating, or migrating.
  + Objectives
    - Create manifests for PersistentVolumes (PVs) and PersistentVolumeClaims (PVCs) for Google Cloud persistent disks (dynamically created or existing)
    - Mount Google Cloud persistent disk PVCs as volumes in Pods
    - Use manifests to create StatefulSets
    - Mount Google Cloud persistent disk PVCs as volumes in StatefulSets
    - Verify the connection of Pods in StatefulSets to particular PVs as the Pods are stopped and restarted
  + Task 1. Create PVs and PVCs
    - In this task, you create a PVC, which triggers Kubernetes to automatically create a PV.
      * export my\_zone=us-central1-a
      * export my\_cluster=standard-cluster-1
      * source <(kubectl completion bash)
      * gcloud container clusters get-credentials $my\_cluster --zone $my\_zone
    - Create and apply a manifest with a PVC
      * Most of the time, you don't need to directly configure PV objects or create Compute Engine persistent disks. Instead, you can create a PVC, and Kubernetes automatically provisions a persistent disk for you.
      * You create the PVC in this task using the pvc-demo.yaml manifest file that has been provided for you. This creates a 30 gigabyte PVC called hello-web-disk that can be mounted as read-write volume on a single node at a time.

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

name: hello-web-disk

spec:

accessModes:

- ReadWriteOnce

resources:

requests:

storage: 30Gi

* + - * git clone https://github.com/GoogleCloudPlatform/training-data-analyst
      * ln -s ~/training-data-analyst/courses/ak8s/v1.1 ~/ak8s
      * cd ~/ak8s/Storage/
      * To create the PVC, execute the following command:
        + kubectl apply -f pvc-demo.yaml
      * To show your newly created PVC, execute the following command:
        + kubectl get persistentvolumeclaim
  + Task 2. Mount and verify Google Cloud persistent disk PVCs in Pods
    - In this task, you attach your persistent disk PVC to a Pod. You mount the PVC as a volume as part of the manifest for the Pod.
    - Mount the PVC to a Pod
      * The manifest file pod-volume-demo.yaml deploys an nginx container, attaches the pvc-demo-volume to the Pod and mounts that volume to the path /var/www/html inside the nginx container. Files saved to this directory inside the container will be saved to the persistent volume and persist even if the Pod and the container are shutdown and recreated.

kind: Pod

apiVersion: v1

metadata:

name: pvc-demo-pod

spec:

containers:

- name: frontend

image: nginx

volumeMounts:

- mountPath: "/var/www/html"

name: pvc-demo-volume

volumes:

- name: pvc-demo-volume

persistentVolumeClaim:

claimName: hello-web-disk

* + - * To create the Pod with the volume, execute the following command:
        + kubectl apply -f pod-volume-demo.yaml
      * If you do this quickly after creating the Pod you will see the status listed as "ContainerCreating" while the volume is mounted before the status changes to "Running".
      * To verify the PVC is accessible within the Pod, you must gain shell access to your Pod. To start the shell session, execute the following command:
        + kubectl exec -it pvc-demo-pod -- sh
      * To create a simple text message as a web page in the Pod enter the following commands:
        + echo Test webpage in a persistent volume!>/var/www/html/index.html
        + chmod +x /var/www/html/index.html
      * Verify the text file contains your message.
        + cat /var/www/html/index.html
      * Enter the following command to leave the interactive shell on the nginx container.
        + exit
    - Test the persistence of the PV
      * You will now delete the Pod from the cluster, confirm that the PV still exists, then redeploy the Pod and verify the contents of the PV remain intact.
      * Delete the pvc-demo-pod.
        + kubectl delete pod pvc-demo-pod
      * To show your PVC, execute the following command:
        + kubectl get persistentvolumeclaim
      * Your PVC still exists, and was not deleted when the Pod was deleted.
      * Redeploy the pvc-demo-pod.
        + kubectl apply -f pod-volume-demo.yaml
      * The Pod will deploy and the status will change to "Running" faster this time because the PV already exists and does not need to be created.
      * To verify the PVC is is still accessible within the Pod, you must gain shell access to your Pod. To start the shell session, execute the following command:
        + kubectl exec -it pvc-demo-pod -- sh
        + cat /var/www/html/index.html
      * The contents of the persistent volume were not removed, even though the Pod was deleted from the cluster and recreated.
  + Task 3. Create StatefulSets with PVCs
    - In this task, you use your PVC in a StatefulSet. A StatefulSet is like a Deployment, except that the Pods are given unique identifiers.
    - Release the PVC
      * Before you can use the PVC with the statefulset, you must delete the Pod that is currently using it. Execute the following command to delete the Pod:
        + kubectl delete pod pvc-demo-pod
    - Create a StatefulSet
      * The manifest file statefulset-demo.yaml creates a StatefulSet that includes a LoadBalancer service and three replicas of a Pod containing an nginx container and a volumeClaimTemplate for 30 gigabyte PVCs with the name hello-web-disk. The nginx containers mount the PVC called hello-web-disk at /var/www/html as in the previous task.

kind: Service

apiVersion: v1

metadata:

name: statefulset-demo-service

spec:

ports:

- protocol: TCP

port: 80

targetPort: 9376

type: LoadBalancer

---

apiVersion: apps/v1

kind: StatefulSet

metadata:

name: statefulset-demo

spec:

selector:

matchLabels:

app: MyApp

serviceName: statefulset-demo-service

replicas: 3

updateStrategy:

type: RollingUpdate

template:

metadata:

labels:

app: MyApp

spec:

containers:

- name: stateful-set-container

image: nginx

ports:

- containerPort: 80

name: http

volumeMounts:

- name: hello-web-disk

mountPath: "/var/www/html"

volumeClaimTemplates:

- metadata:

name: hello-web-disk

spec:

accessModes: [ "ReadWriteOnce" ]

resources:

requests:

storage: 30Gi

* + - * To create the StatefulSet with the volume, execute the following command:
        + kubectl apply -f statefulset-demo.yaml
      * You now have a statefulset running behind a service named statefulset-demo-service.
    - Verify the connection of Pods in StatefulSets
      * Use "kubectl describe" to view the details of the StatefulSet:
        + kubectl describe statefulset statefulset-demo
      * To list the PVCs, execute the following command:
        + kubectl get pvc
      * The original hello-web-disk is still there and you can now see the individual PVCs that were created for each Pod in the new statefulset Pod.
      * Use "kubectl describe" to view the details of the first PVC in the StatefulSet:
        + kubectl describe pvc hello-web-disk-statefulset-demo-0
  + Task 4. Verify the persistence of Persistent Volume connections to Pods managed by StatefulSets
    - In this task, you verify the connection of Pods in StatefulSets to particular PVs as the Pods are stopped and restarted.
    - To verify that the PVC is accessible within the Pod, you must gain shell access to your Pod. To start the shell session, execute the following command:
      * kubectl exec -it statefulset-demo-0 -- sh
    - Verify that there is no index.html text file in the /var/www/html directory.
      * cat /var/www/html/index.html
      * echo Test webpage in a persistent volume!>/var/www/html/index.html
      * chmod +x /var/www/html/index.html
      * exit
    - Delete the Pod where you updated the file on the PVC.
      * kubectl delete pod statefulset-demo-0
    - List the Pods in the cluster.
      * kubectl get pods
    - You will see that the StatefulSet is automatically restarting the statefulset-demo-0 Pod.
    - Note: You need to wait until the Pod status shows that it is running again.
    - Connect to the shell on the new statefulset-demo-0 Pod.
      * kubectl exec -it statefulset-demo-0 -- sh
    - Verify that the text file still contains your message.
      * cat /var/www/html/index.html
        + Test webpage in a persistent volume!
    - The StatefulSet restarts the Pod and reconnects the existing dedicated PVC to the new Pod ensuring that the data for that Pod is preserved.
      * exit