Google Cloud

Partner Certification Academy


# Associate Cloud Engineer

The information in this presentation is classified:

## Google confidential & proprietary

⚠ This presentation is shared with you under NDA.

- Do **not** record or take screenshots of this presentation.
- Do **not** share or otherwise distribute the information in this presentation with anyone **inside** or **outside** of your organization.

Thank you!

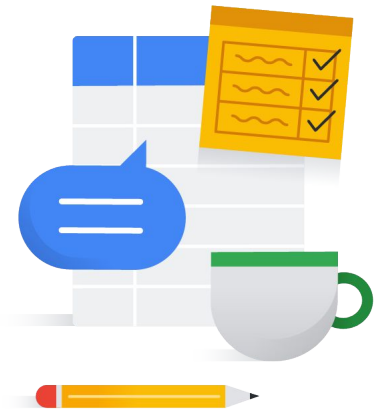
Google Cloud

# Session logistics

- When you have a question, please:
  - Click the Raise hand button in Google Meet.
  - Or add your question to the Q&A section of Google Meet.
  - Please note that answers may be deferred until the end of the session.
- These slides are available in the Student Lecture section of your Qwiklabs classroom.
- The session is **not recorded.**
- Google Meet does not have persistent chat.
  - If you get disconnected, you will lose the chat history.
  - Please copy any important URLs to a local text file as they appear in the chat.

# Program issues or concerns?

- Problems with **accessing** Cloud Skills Boost for Partners
  - partner-training@google.com
- Problems with **a lab** (locked out, etc.)
  - support@qwiklabs.com
- Problems with accessing Partner Advantage
  - https://support.google.com/googlecloud/topic/9198654




# Associate Cloud Engineer

## The Google Cloud Certified

**Associate Cloud Engineer** exam assesses your ability to:

- Setup a cloud solution environment
- Plan and configure a cloud solution
- Deploy and implement a cloud solution
- Ensure successful operation of a cloud solution
- Configure access and security

For more information:

https://cloud.google.com/certification/cloud-engineer



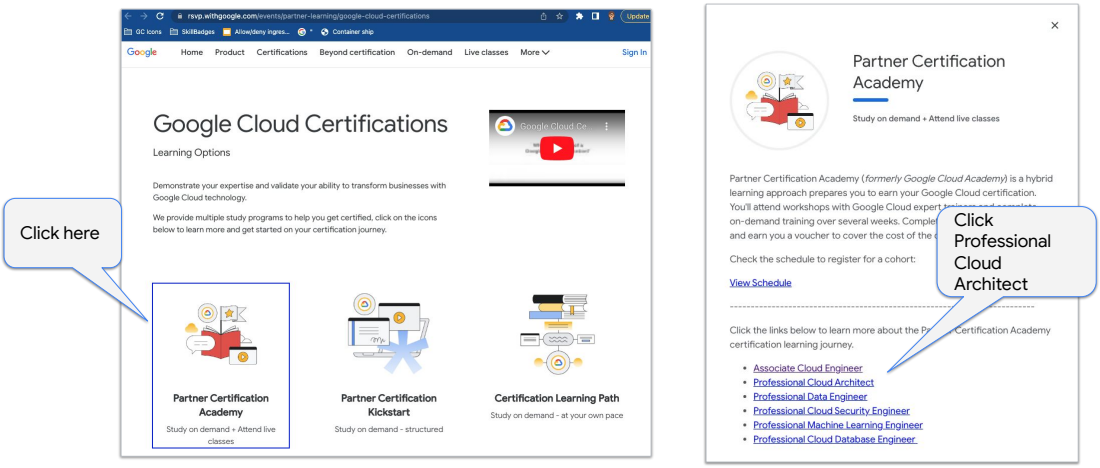
Associate Cloud Engineer
https://cloud.google.com/certification/cloud-engineer

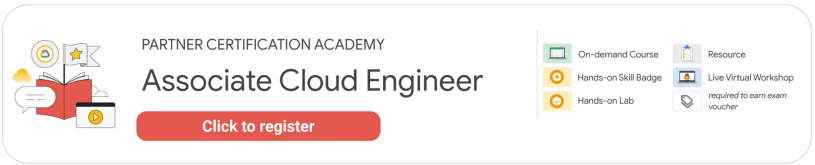
Exam Guide
https://cloud.google.com/certification/guides/cloud-engineer

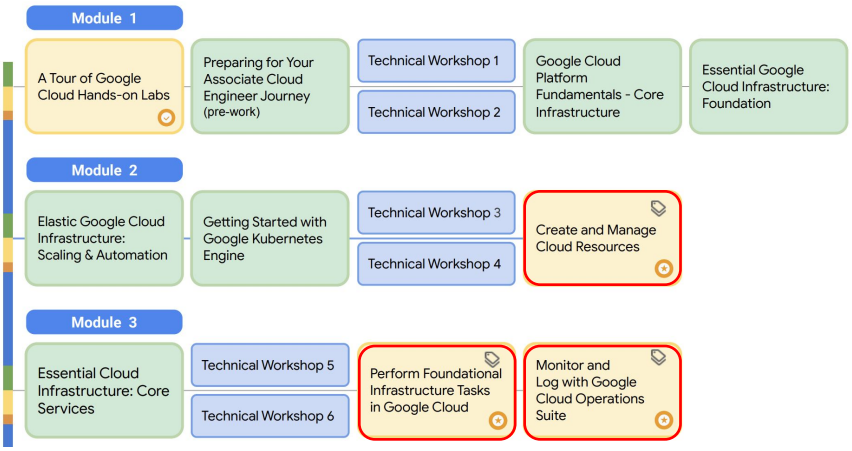
Sample Questions
https://docs.google.com/forms/d/e/1FAIpQLSfexWKtXT2OSFJ-obA4iT3GmzgiOCGvjrT9OfxilWC1yPtmfQ/viewform

# Learning Path - Partner Certification Academy Website

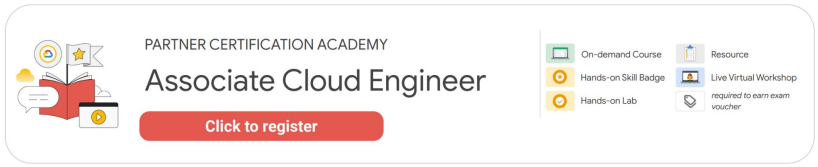
Go to: https://rsvp.withgoogle.com/events/partner-learning/google-cloud-certifications

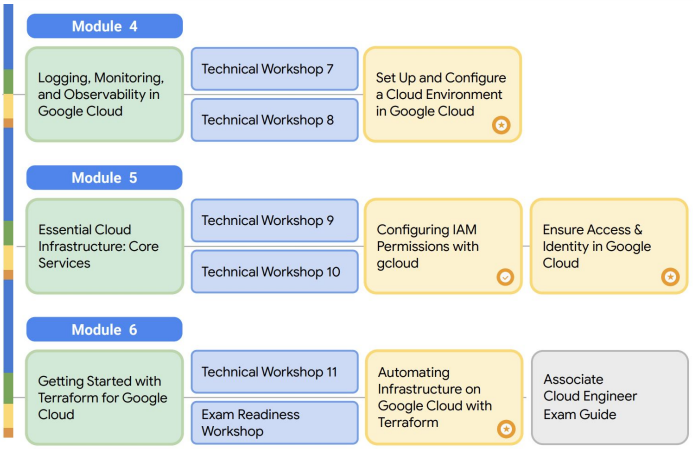





Needed for Exam Voucher



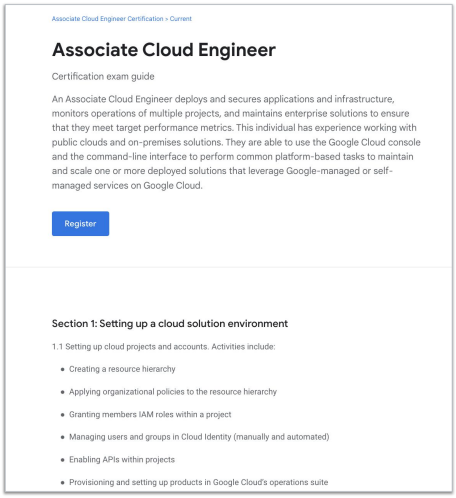
# Associate Cloud Engineer (ACE) Exam Guide

Each module of this course covers Google Cloud services based on the topics in the ACE Exam Guide

The primary topics are:

- Compute Engine
- VPC Networks
- Google Kubernetes Engine
- Cloud Run, Cloud Functions and App Engine
- Cloud Storage and database options
- Resource Hierarchy/Identity and Access Management (IAM)
- Logging and Monitoring

Next discussion



Associate Cloud Engineer Certification > Current

**Associate Cloud Engineer**

Certification exam guide

An Associate Cloud Engineer deploys and secures applications and infrastructure, monitors operations of multiple projects, and maintains enterprise solutions to ensure that they meet target performance metrics. This individual has experience working with public clouds and on-premises solutions. They are able to use the Google Cloud console and the command-line interface to perform common platform-based tasks to maintain and scale one or more deployed solutions that leverage Google-managed or self-managed services on Google Cloud.

Register

**Section 1: Setting up a cloud solution environment**

1.1 Setting up cloud projects and accounts. Activities include:

- Creating a resource hierarchy
- Applying organizational policies to the resource hierarchy
- Granting members IAM roles within a project
- Managing users and groups in Cloud Identity (manually and automated)
- Enabling APIs within projects
- Provisioning and setting up products in Google Cloud's operations suite

https://cloud.google.com/certification/guides/cloud-engineer/


# Google Kubernetes Engine, Cloud Run, Cloud Functions, App Engine

# Exam Guide Overview - Google Kubernetes Engine

Google Kubernetes Engine

**2.2 Planning and configuring compute resources. Considerations include:**

2.2.1 Selecting appropriate compute choices for a given workload (e.g., Compute Engine, Google Kubernetes Engine, Cloud Run, Cloud Functions)

2.2.2 Using preemptible VMs and custom machine types as appropriate

**3.2 Deploying and implementing Google Kubernetes Engine resources.**

3.2.1 Installing and configuring the command line interface (CLI) for Kubernetes (kubectl)

3.2.2 Deploying a Google Kubernetes Engine cluster with different configurations including AutoPilot, regional clusters, private clusters, etc.

3.2.3 Deploying a containerized application to Google Kubernetes Engine

3.2.4 Configuring Google Kubernetes Engine monitoring and logging

**4.2 Managing Google Kubernetes Engine resources. Tasks include:**

4.2.1 Viewing current running cluster inventory (nodes, pods, services)

4.2.2 Browsing Docker images and viewing their details in the Artifact Registry

4.2.3 Working with node pools (e.g., add, edit, or remove a node pool)

4.2.4 Working with pods (e.g., add, edit, or remove pods)

4.2.5 Working with services (e.g., add, edit, or remove a service)

4.2.6 Working with stateful applications (e.g. persistent volumes, stateful sets)

4.2.7 Managing Horizontal and Vertical autoscaling configurations

4.2.8 Working with management interfaces (e.g., Google Cloud console, Cloud Shell, Cloud SDK, kubectl)

# Exam Guide - Kubernetes Engine



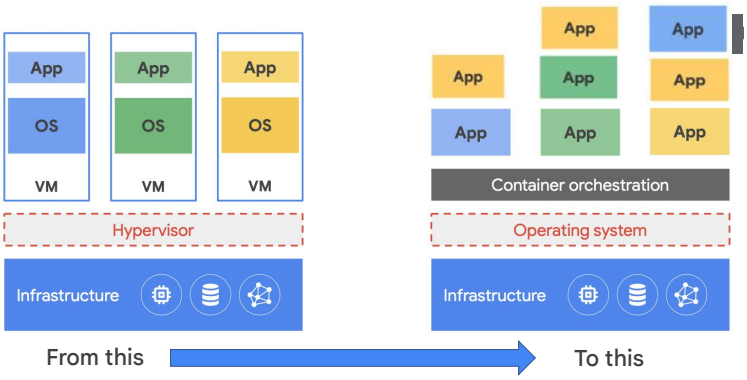
4.2 Managing Google Kubernetes Engine resources. Tasks include:

- 4.2.1 Viewing current running cluster inventory (nodes, pods, services)
- 4.2.2 Browsing Docker images and viewing their details in the Artifact Registry
- 4.2.3 Working with node pools (e.g., add, edit, or remove a node pool)
- 4.2.4 Working with pods (e.g., add, edit, or remove pods)
- 4.2.5 Working with services (e.g., add, edit, or remove a service)
- 4.2.6 Working with stateful applications (e.g. persistent volumes, stateful sets)
- 4.2.7 Managing Horizontal and Vertical autoscaling configurations
- 4.2.8 Working with management interfaces (e.g., Google Cloud console, Cloud Shell, Cloud SDK, kubectl)

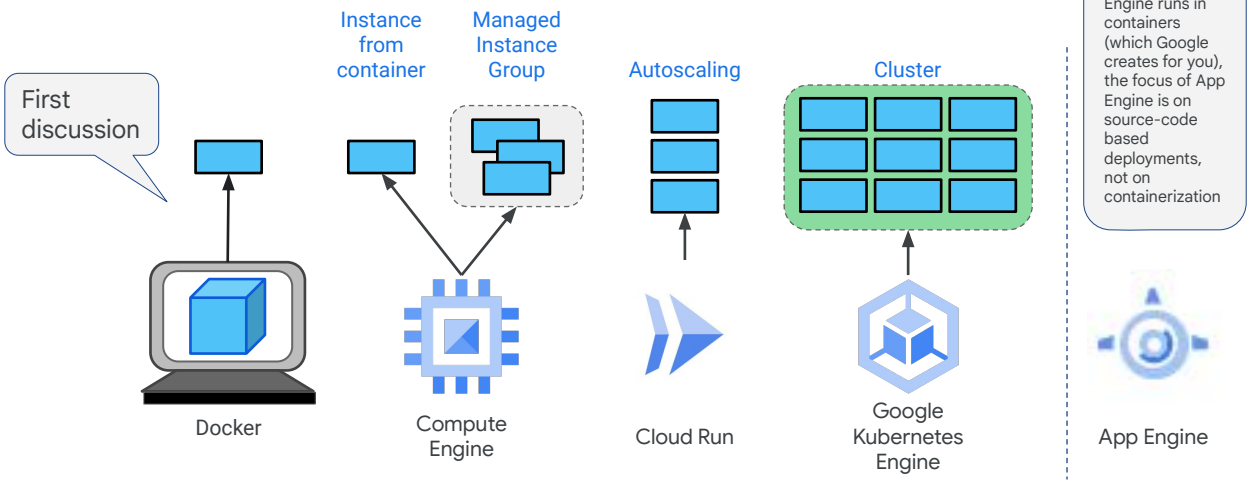
# Containers are an approach to app isolation

Containers typically deployed on VMs

- Allows better utilization of VMs resources
- A VM that ran one application before can now run many containers



# Where can you run containers?



# Docker

- Provides the ability to package and run an application in an environment called a container
- Images are very light-weight, pre-configured virtual environments
  - Contain everything needed to run the application, so do not need to rely on what is currently installed on the host
- Docker images will run on any platform that has a container runtime installed
  - Google Kubernetes Engine uses the containerd runtime on all GKE nodes as GKE Version 1.24

*Docker designed `containerd`, which is now a part of the CNCF, an organization that supports Kubernetes and Docker-based deployments. Docker is still an independent project that uses `containerd` as its runtime.
https://blog.purestorage.com/purely-informational/containerd-vs-docker-whats-the-difference



Docker website
https://www.docker.com/

Docker is a container format used to run applications. They can be leveraged in a microservice architecture.

Docker images are lightweight, preconfigured virtual environments used to run our application. The images include the software required to run the containerized application. The applications are deployed inside the Docker image.

Docker images are versatile and will run on any platform that has Docker installed.

# How do you get an app into a container using Docker?

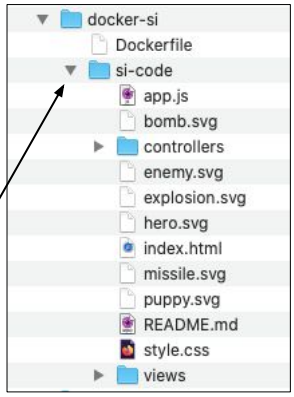
Create a `Dockerfile` which specifies such things as:

- The OS image to use
- Where to copy the code to inside the image
- And other items depending on what is being deployed
  - Libraries to load, etc.

Example Dockerfile content

```
#Minimalized debian container image
#Downloaded from https://github.com/bitnami/bitnami-docker-apache
FROM bitnami/apache:latest

#Copy the code in local directory to apache folder.
#Code is JavaScript
COPY ./si-code /app
```



# Then you build and run the container as an image

```
$> docker build -t space-invaders .
$> docker run -d space-invaders
```

- **docker build** builds a container and stores it locally as a runnable image.
- **docker run** starts the container image

# When local testing is complete

- Tag the image and upload it to a registry service (like Google Artifact Registry) for sharing

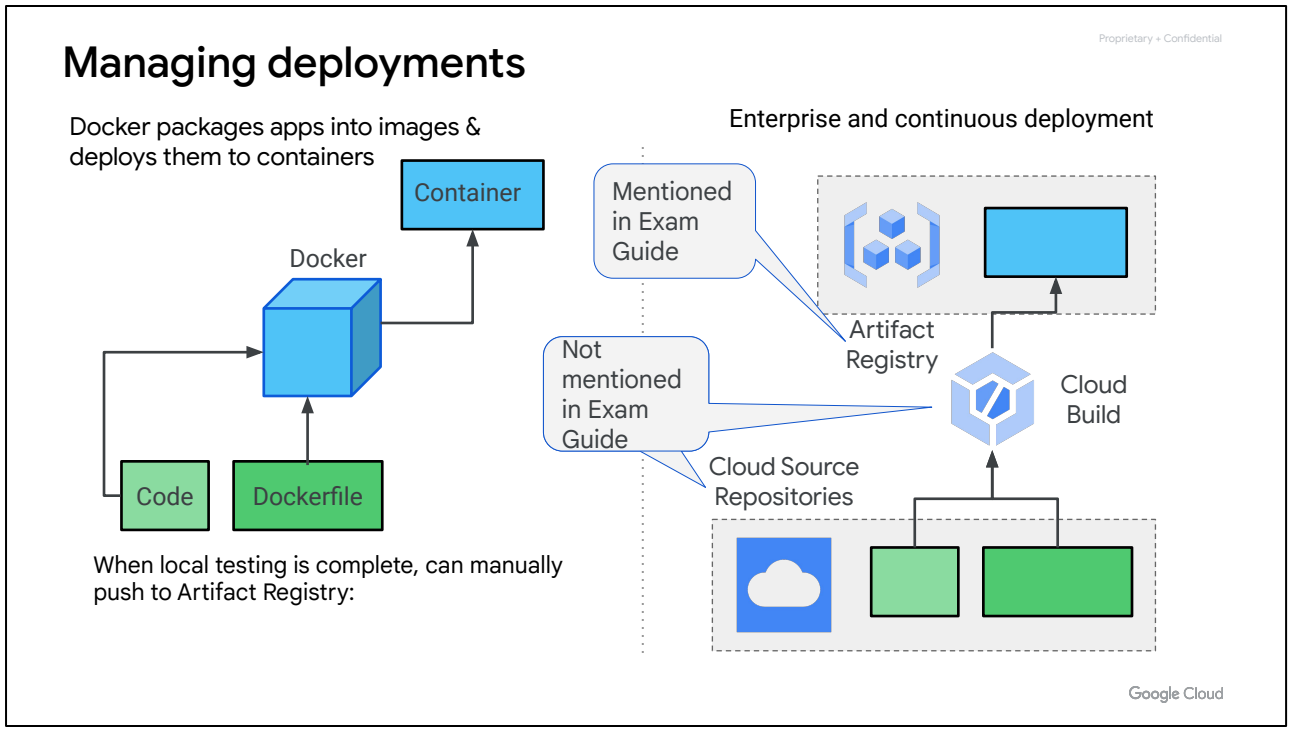
```
$> docker tag space-invaders
us-central1-docker.pkg.dev/bt-spaceinvaders-ke/space-invaders/spaceinvaders-
image

$> docker push
us-central1-docker.pkg.dev/bt-spaceinvaders-ke/space-invaders/spaceinvaders-
image:latest
```

Location in the Artifact Registry

Manually pushing the image

- At this point, the image can be deployed to Compute Engine, Cloud Run, Kubernetes Engine or App Engine



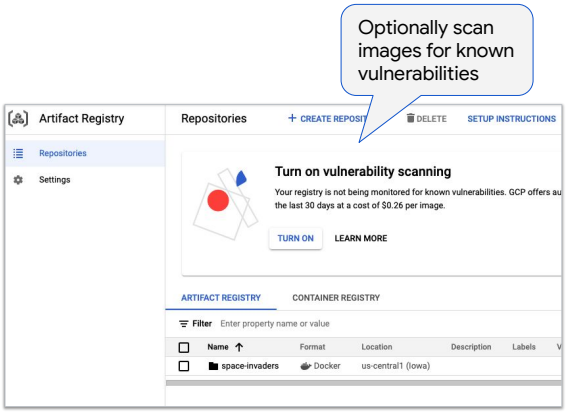

Developers can test their container locally. When done, they could manually push the image to the Artifact Registry as shown on the left in this slide.

If you had a hundred developers sharing source files, you would need a system for managing them, for tracking them, versioning them, and enforcing a check-in, review, and approval process. They typically push source code to a shared code repository, such as the Google Cloud Source Repositories, as shown on the right of this slide. Cloud Source Repositories is a cloud-based solution which integrates with Cloud Build.

Cloud Build functions similarly to Docker in that it accepts code and configuration and builds containers (among other things). Cloud Build offers many features and services that are geared towards professional development. It is designed to fit into a continuous development / continuous deployment workflow and can scale to handle many application developers working on and continuously updating a live global service.

# Artifact Registry

- Provides
  - Built-in vulnerability scanning of container images
  - Controlled access with fine-grained control (IAM)
  - Regional and multi-regional repositories
    - Can have multiple per project
- Store multiple artifact formats
  - Docker images
  - OS packages for Linux distributions
  - Language packages for Python, Java, and Node




https://cloud.google.com/artifact-registry



Overview of Artifact Registry: https://cloud.google.com/artifact-registry/docs/overview

Container concepts: https://cloud.google.com/artifact-registry/docs/container-concepts

Artifact Registry has its own IAM roles and can be deployed multi-regionally.

# Exam Guide - Kubernetes Engine



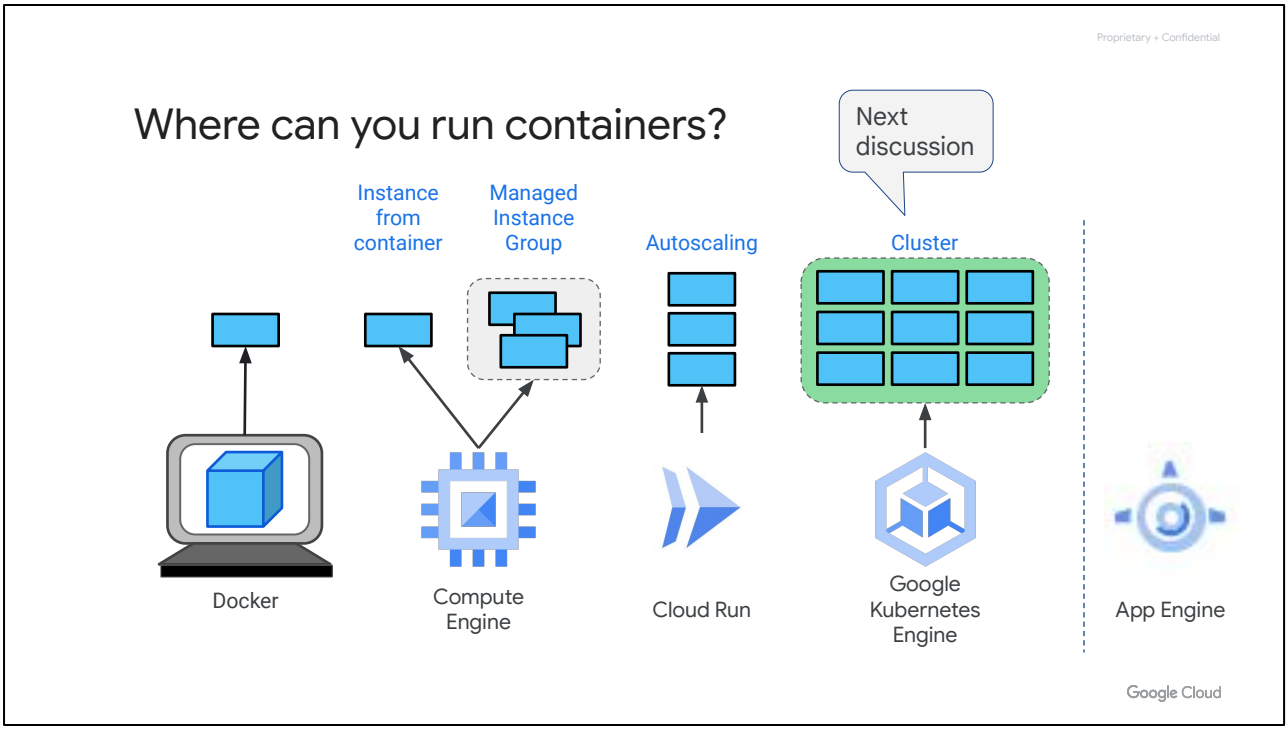
2.2 Planning and configuring compute resources. Considerations include:

- 2.2.1 Selecting appropriate compute choices for a given workload (e.g., Compute Engine, Google Kubernetes Engine, Cloud Run, Cloud Functions)
- 2.2.2 Using preemptible VMs and custom machine types as appropriate

Discussed Compute Engine in another module

3.2 Deploying and implementing Google Kubernetes Engine resources. Tasks include:

- 3.2.1 Installing and configuring the command line interface (CLI) for Kubernetes (kubectl)
- 3.2.2 Deploying a Google Kubernetes Engine cluster with different configurations including AutoPilot, regional clusters, private clusters, etc.
- 3.2.3 Deploying a containerized application to Google Kubernetes Engine
- 3.2.4 Configuring Google Kubernetes Engine monitoring and logging



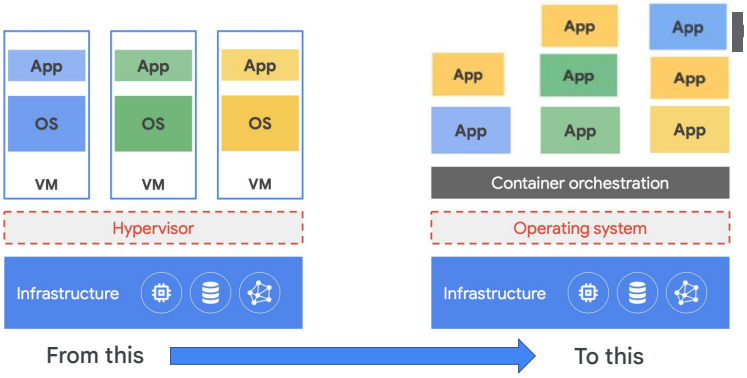

App Engine is covered later in this module. While App Engine runs in containers (which Google creates for you), the focus of App Engine is on source-code based deployments, not on containerization.

# Containers are an approach to app isolation

Repeat of an earlier slide

**Containers typically deployed on VMs**

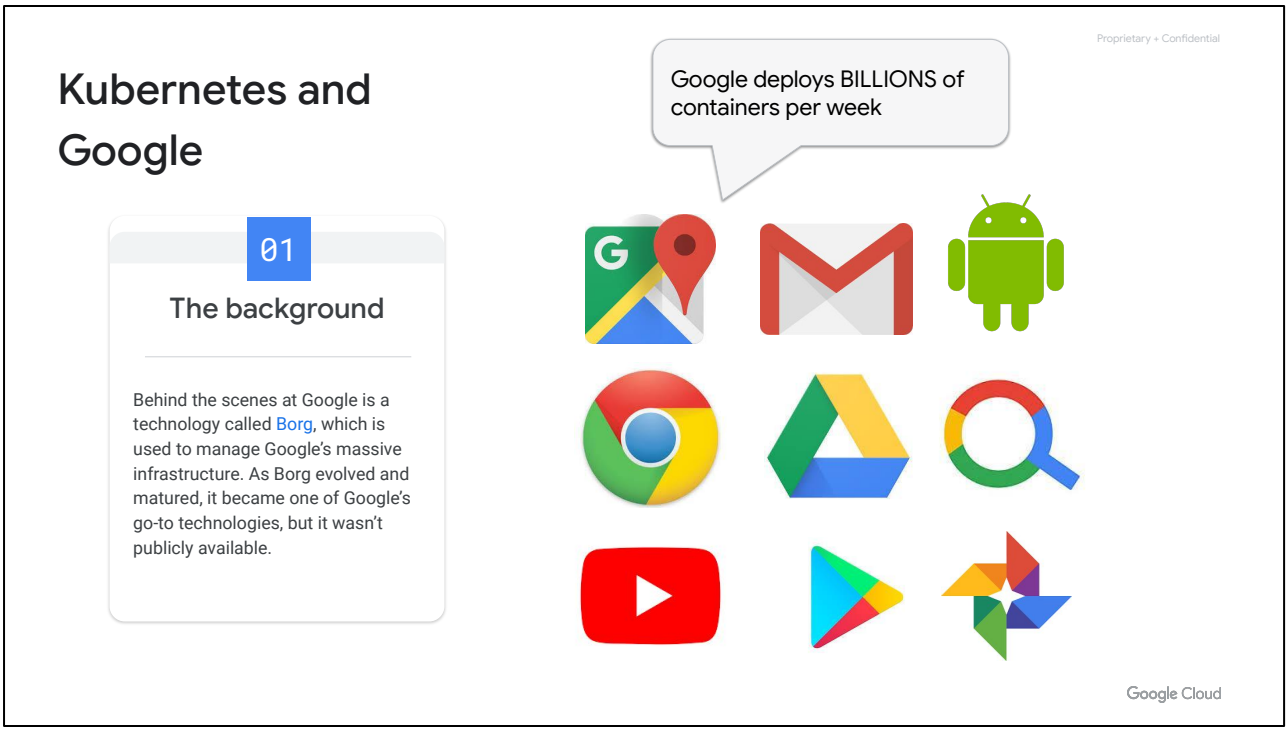
- Allows better utilization of VMs resources
- A VM that ran one application before can now run many containers



# The word Kubernetes comes from the Greek word for helmsman or pilot

- Just like a cargo ship, a VM (aka “node” in Kubernetes) can host multiple containers
- Each container is independent of the others
  - May have multiples of the same container running depending on the use case
    - Number can be scaled up/down as needed

Photo by Ian Taylor on Unsplash




Google is the pioneer when it comes to containerized services.  From Gmail to YouTube to Search, everything at Google runs in containers. Containerization allows development teams to move fast, deploy software efficiently, and operate at an unprecedented scale.

Behind the scenes at Google is a technology called Borg, which is used to manage Google's massive infrastructure. As Borg evolved and matured, it became one of Google's go-to technologies, but it wasn't publicly available.

Source: Kubernetes: Your Hybrid Cloud Strategy
https://cloud.google.com/files/kubernetes-your-hybrid-cloud-strategy.pdf

# Kubernetes and Google

01

## The background

Behind the scenes at Google is a technology called Borg, which is used to manage Google's massive infrastructure. As Borg evolved and matured, it became one of Google's go-to technologies, but it wasn't publicly available.

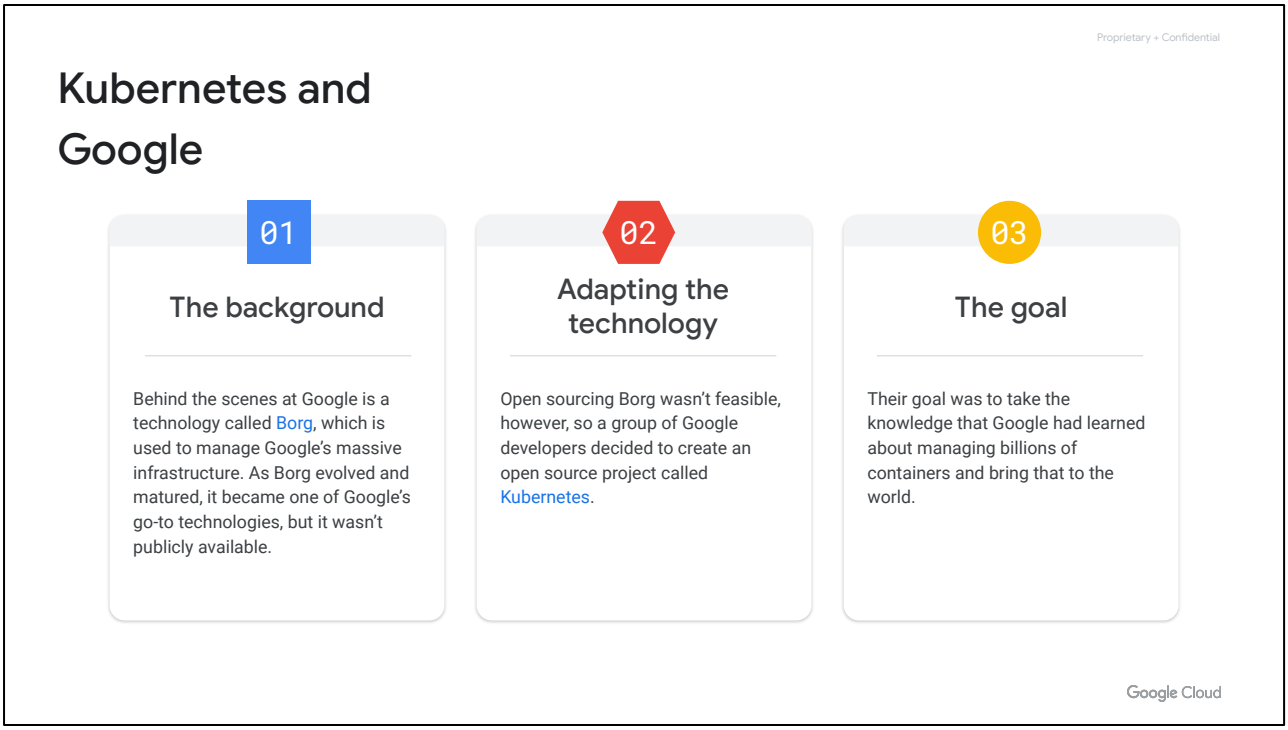
02

## Adapting the technology

Open sourcing Borg wasn't feasible, however, so a group of Google developers decided to create an open source project called Kubernetes.



Google wanted to open source the technology behind BORG. To do so, they created an open source project named Kubernetes in 2014




Their goal was to take the knowledge that Google had learned about managing billions of containers and bring that to the world.

A container story - Google Kubernetes Engine
https://cloud.google.com/blog/topics/developers-practitioners/container-story-google-kubernetes-engine

## Kubernetes

Today, Kubernetes is widely supported in the industry:

- Google Kubernetes Engine (GKE)
- Microsoft in Azure Container Service (AKS)
- AWS Elastic Kubernetes Service (EKS)
- Estimated that > 54% of Fortune 500 companies have adopted Kubernetes



Kubernetes, or K8s, can run on a variety of supported platforms including:

- Support on Google Cloud using Google Kubernetes Engine or GKE
- Support on Microsoft Azure using Azure Container service or AKS
- Support on AWS using Elastic Kubernetes Service or EKS
- Support on OpenStack
- Mesos and Pivotal Cloud Foundry also supports Kubernetes

# Exam Guide - Kubernetes Engine

2.2 Planning and configuring compute resources. Considerations include:
- 2.2.1 Selecting appropriate compute choices for a given workload (e.g., Compute Engine, Google Kubernetes Engine, Cloud Run, Cloud Functions)
- 2.2.2 Using preemptible VMs and custom machine types as appropriate

3.2 Deploying and implementing Google Kubernetes Engine resources. Tasks include:
- 3.2.1 Installing and configuring the command line interface (CLI) for Kubernetes (kubectl)
- 3.2.2 Deploying a Google Kubernetes Engine cluster with different configurations including AutoPilot, regional clusters, private clusters, etc.
- 3.2.3 Deploying a containerized application to Google Kubernetes Engine
- 3.2.4 Configuring Google Kubernetes Engine monitoring and logging

## Installing and configuring the command line interface

- `kubectl` is the command-line tool used to interact with GKE clusters
  - To install

```
gcloud components install kubectl
```

- Authorization plug-ins must be installed to provide authentication tokens to communicate with GKE clusters

```
gcloud components install gke-gcloud-auth-plugin
```

- Update the `kubectl` configuration to use the plugin

```
gcloud container clusters get-credentials CLUSTER_NAME
```



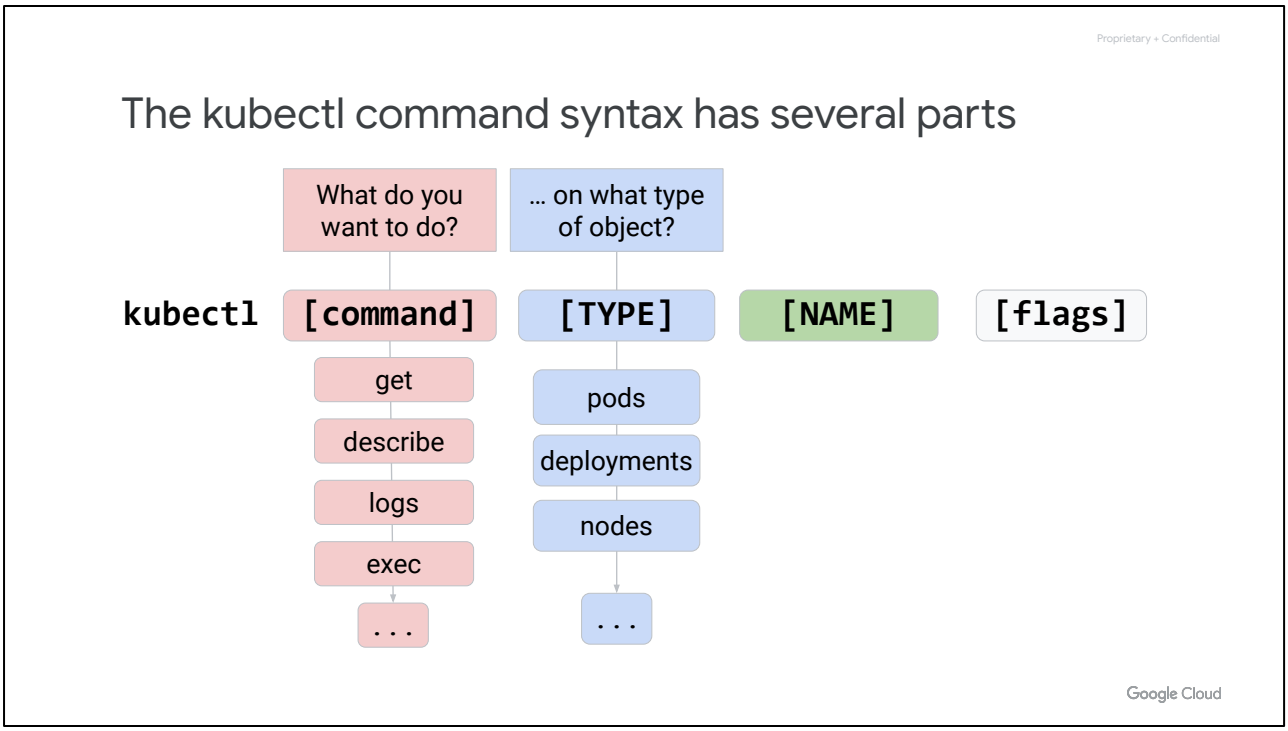
Install kubectl and configure cluster access:
https://cloud.google.com/kubernetes-engine/docs/how-to/cluster-access-for-kubectl

The first command line example shows how to deploy a service based on a configuration file. The kubernetes-config.yaml file will hold the configuration.

To show running pods, you can use the kubectl get pods command.

The kubectl get deployments command will list all the deployments currently running.

The kubectl describe deployments command is used to display the details of a deployment.



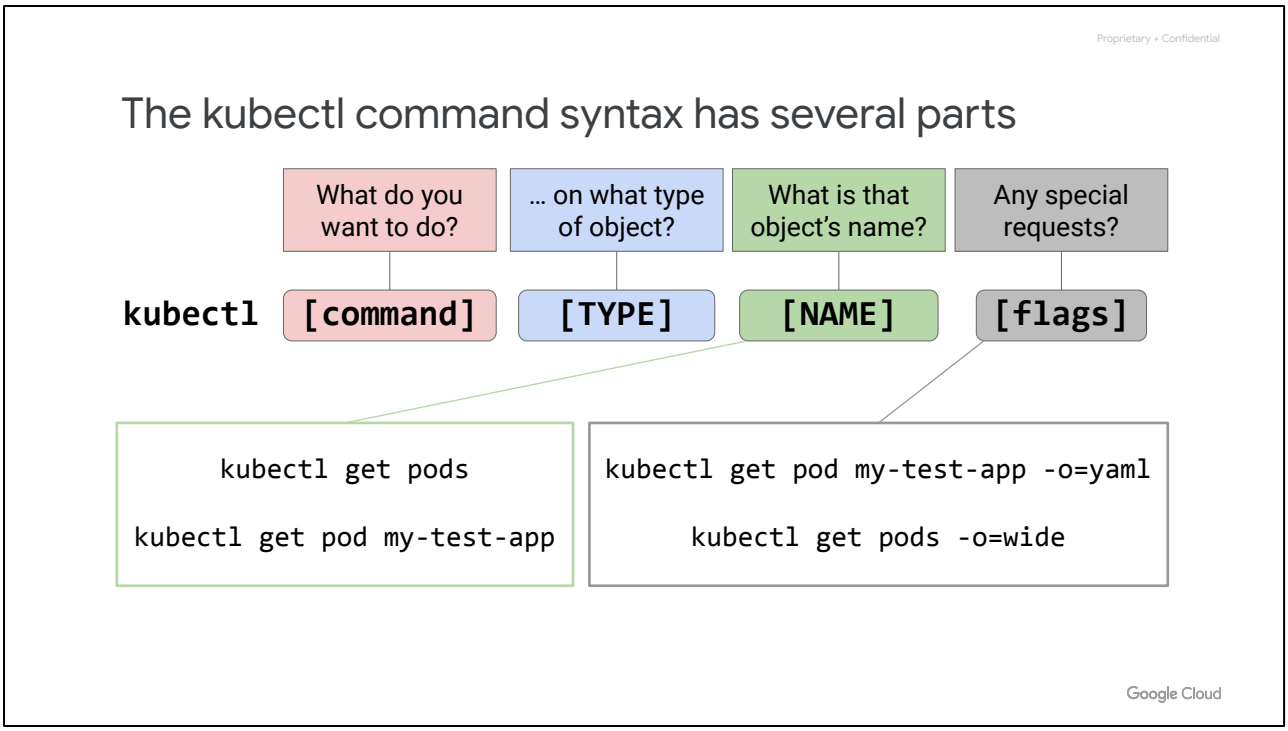

kubectl syntax:
https://kubernetes.io/docs/reference/kubectl/cheatsheet/

Once the config file in the .kube folder has been configured, the kubectl command automatically references this file and connects to the default cluster without prompting you for credentials. Now let's talk about how to use the kubectl command. Its syntax is composed of several parts: the command, the type, the name, and optional flags.

'Command' specifies the action that you want to perform, such as get, describe, logs, or exec. Some commands show you information, while others allow you to change the cluster's configuration.

'TYPE' defines the Kubernetes object that the 'command' acts upon. For example, you could specify Pods, Deployments, nodes, or other objects, including the cluster itself.

TYPE used in combination with 'command' tells kubectl what you want to do and the type of object you want to perform that action on.




'NAME' specifies the object defined in 'TYPE.' The Name field isn't always needed, especially when you're using commands that list or show you information.
For example, if you run the command "kubectl get pods" without specifying a name, the command returns the list of all Pods. To filter this list you specify a Pod's name, such as "kubectl get pod my-test-app ". kubectl then returns information only on the Pod named 'my-test-app'.

Some commands support additional optional flags that you can include at the end of the command.
Think of this as making a special request, like formatting the output in a certain way. You could view the state of a Pod by using the command "kubectl get pod my-test-app -o=yaml". By the way, telling kubectl to give you output in YAML format is a really useful tool. You'll often want to capture the existing state of a Kubernetes object in a YAML file so that, for example, you can recreate it in a different cluster.

You can also use flags to display more information than you normally see. For instance, you can run the command "kubectl get pods -o=wide" to display the list of Pods in "wide" format, which means you see additional columns of data for each of the Pods in the list. One noteworthy piece of extra information you get in wide format: which Node each Pod is running on.

# Exam Guide - Kubernetes Engine

2.2 Planning and configuring compute resources. Considerations include:

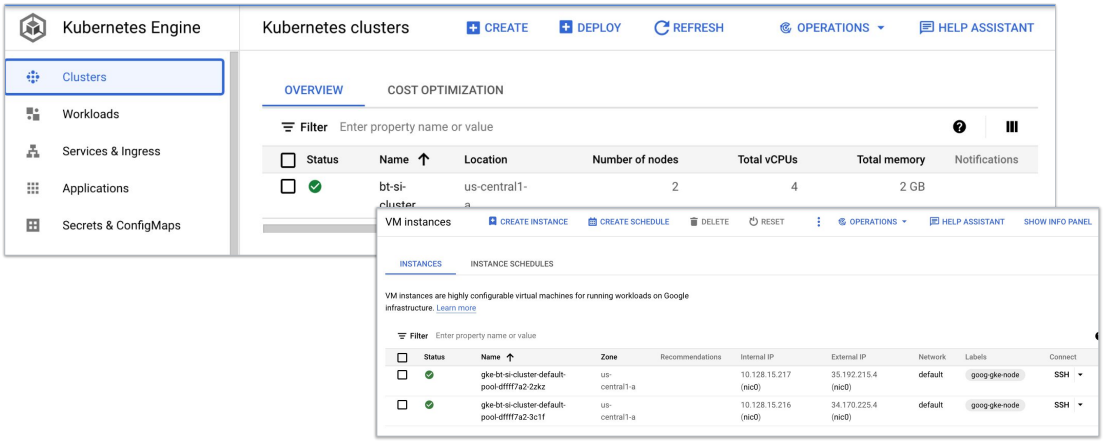
- 2.2.1 Selecting appropriate compute choices for a given workload (e.g., Compute Engine, Google Kubernetes Engine, Cloud Run, Cloud Functions)
- 2.2.2 Using preemptible VMs and custom machine types as appropriate

3.2 Deploying and implementing Google Kubernetes Engine resources. Tasks include:

- 3.2.1 Installing and configuring the command line interface (CLI) for Kubernetes (kubectl)
- 3.2.2 Deploying a Google Kubernetes Engine cluster with different configurations including AutoPilot, regional clusters, private clusters, etc.
- 3.2.3 Deploying a containerized application to Google Kubernetes Engine
- 3.2.4 Configuring Google Kubernetes Engine monitoring and logging

# Kubernetes Engine cluster is a collection of nodes (VMs)

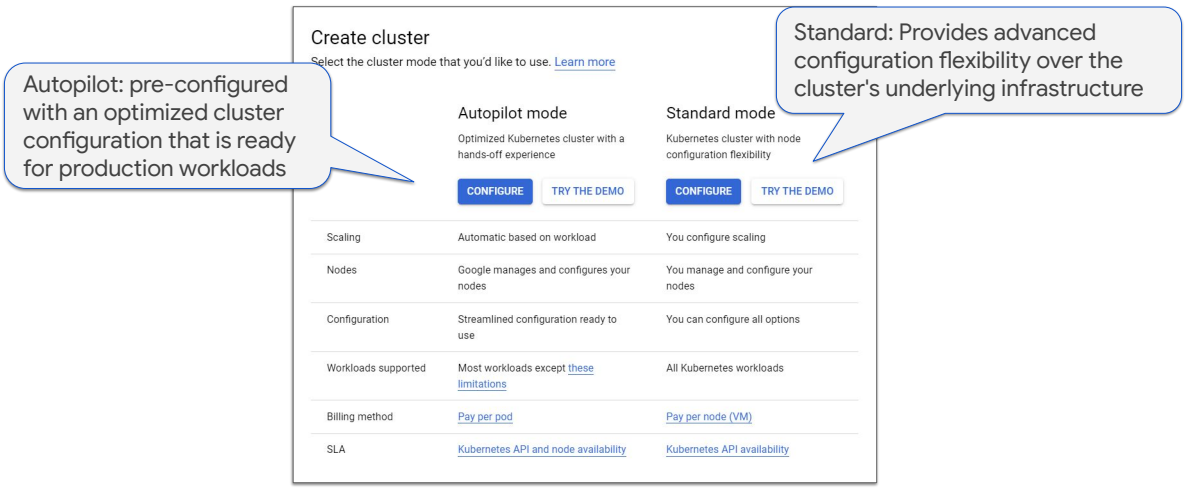
- Application containers are deployed to the cluster within “pods”
  - Kubernetes decides which nodes the pods will run on




Google Cloud

GKE automates the creation of a virtual machine cluster that provides resources to containerized applications. When deploying applications, you deploy them to a cluster and the location of the applications will be handled by Kubernetes.

# Creating Kubernetes Engine clusters



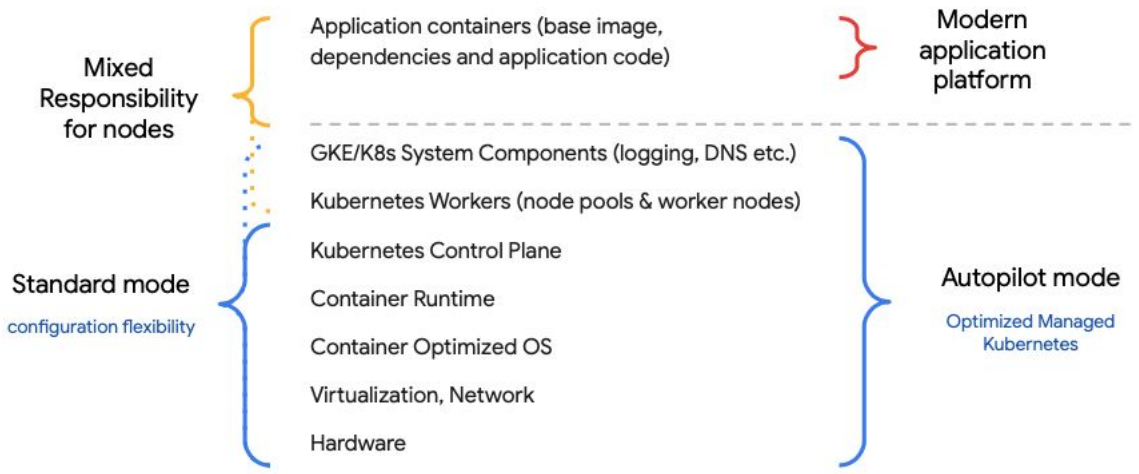



Types of clusters (Standard vs Autopilot, Regional vs Zonal)
https://cloud.google.com/kubernetes-engine/docs/concepts/types-of-cluster

Standard cluster architecture:
https://cloud.google.com/kubernetes-engine/docs/concepts/cluster-architecture

Creating an Autopilot cluster:
https://cloud.google.com/kubernetes-engine/docs/how-to/creating-an-autopilot-cluster

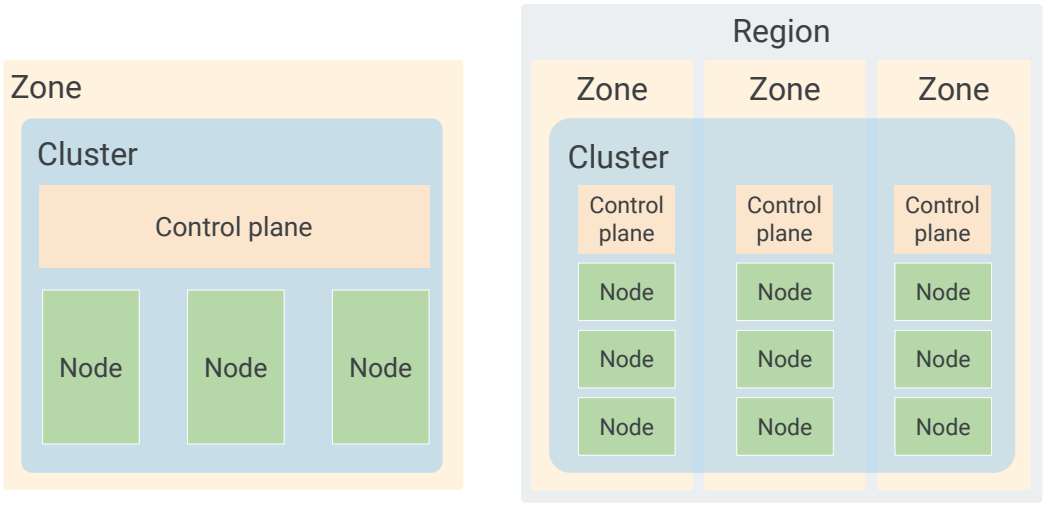
# GKE Autopilot mode - Shared Responsibility model






AutoPilot
https://cloud.google.com/kubernetes-engine/docs/concepts/autopilot-overview

## Zonal vs regional clusters






Creating a regional cluster:
https://cloud.google.com/kubernetes-engine/docs/how-to/creating-a-regional-cluster

Creating a zonal cluster:
https://cloud.google.com/kubernetes-engine/docs/how-to/creating-a-zonal-cluster

By default, a cluster launches in a single Google Cloud compute zone with three identical nodes, all in one node pool. The number of nodes can be changed during or after the creation of the cluster. Adding more nodes and deploying multiple replicas of an application will improve an application's availability. But only up to a point. What happens if the entire compute zone goes down?

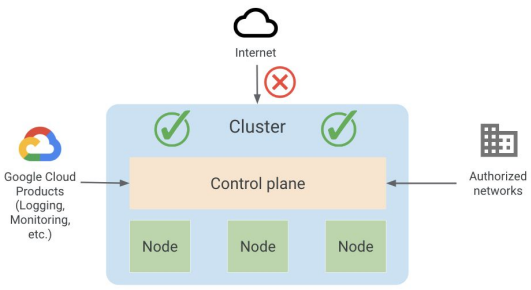
You can address this concern by using a GKE regional cluster. Regional clusters have a single API endpoint for the cluster. However, its control planes and nodes are spread across multiple Compute Engine zones within a region.

Regional clusters ensure that the availability of the application is maintained across multiple zones in a single region. In addition, the availability of the control plane is also maintained so that both the application and management functionality can withstand the loss of one or more, but not all, zones. By default, a regional cluster is spread across 3 zones, each containing 1 control plane and 3 nodes. These numbers

can be increased or decreased. For example, if you have five nodes in Zone 1, you will have exactly the same number of nodes in each of the other zones, for a total of 15 nodes. Once you build a zonal cluster, you can’t convert it into a regional cluster, or vice versa.

## A regional or zonal GKE cluster can also be set up as a private cluster

- A private cluster is a type of cluster that only has internal IP addresses.
  - A VPC subnet is created to host to the cluster worker nodes
    - Nodes, Pods, and Services receive unique subnet IP address ranges
  - The control plane communicates with the nodes via VPC Peering
    - Automatically setup by Google
- These are isolated from the internet by default
  - Can allow restricted access from certain IPs, e.g, CI/CD services






Creating a private cluster:
https://cloud.google.com/kubernetes-engine/docs/how-to/private-clusters

In a private cluster, the entire cluster (that is, the control plane and its nodes) are hidden from the public internet.

Cluster control planes can be accessed by Google Cloud products, such as Cloud Logging or Cloud Monitoring, through an internal IP address.

They can also be accessed by authorized networks through an external IP address. Authorized networks are basically IP address ranges that are trusted to access the control plane. In addition, nodes can have limited outbound access through Private Google Access, which allows them to communicate with other Google Cloud services. For example, nodes can pull container images from Container Registry without needing external IP addresses.

# Creating Kubernetes Engine Clusters - CLI

Standard

```
gcloud container clusters create "my-cluster" --project "my-project"
--region "us-central1"
```

Autopilot

```
gcloud container clusters create-auto "my-cluster" --project "my-project"
--region "us-central1"
```

Private

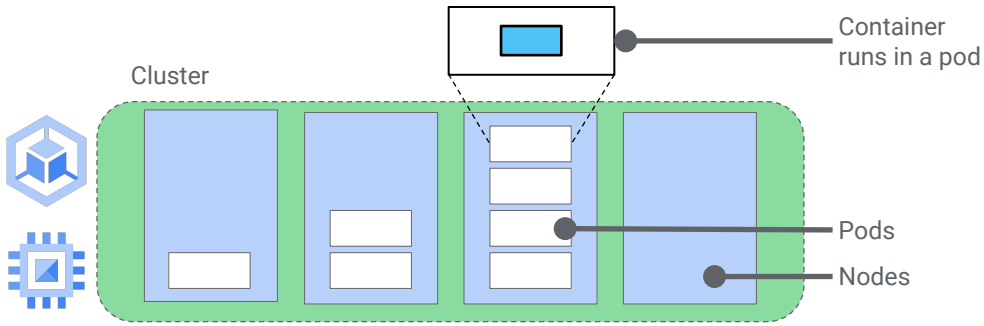
```
gcloud container clusters create private-cluster "my-cluster" --project
"my-project" --region "us-central1"
```

# Exam Guide - Kubernetes Engine

2.2 Planning and configuring compute resources. Considerations include:
- 2.2.1 Selecting appropriate compute choices for a given workload (e.g., Compute Engine, Google Kubernetes Engine, Cloud Run, Cloud Functions)
- 2.2.2 Using preemptible VMs and custom machine types as appropriate

3.2 Deploying and implementing Google Kubernetes Engine resources. Tasks include:
- 3.2.1 Installing and configuring the command line interface (CLI) for Kubernetes (kubectl)
- 3.2.2 Deploying a Google Kubernetes Engine cluster with different configurations including AutoPilot, regional clusters, private clusters, etc.
- 3.2.3 Deploying a containerized application to Google Kubernetes Engine
- 3.2.4 Configuring Google Kubernetes Engine monitoring and logging

## Kubernetes cluster has nodes, pods, and containers






Cluster nodes Auto-upgrading
https://cloud.google.com/kubernetes-engine/docs/how-to/node-auto-upgrades

Cluster nodes Auto-repair
https://cloud.google.com/kubernetes-engine/docs/how-to/node-auto-repair

A Kubernetes cluster is composed of nodes, which are a unit of hardware resources. Nodes in GKE are implemented as VMs in Compute Engine. Each node has pods. Pods are resource management units. A pod is how Kubernetes controls and manages resources needed by applications and how it executes code. Pods also give the system fine-grain control over scaling.

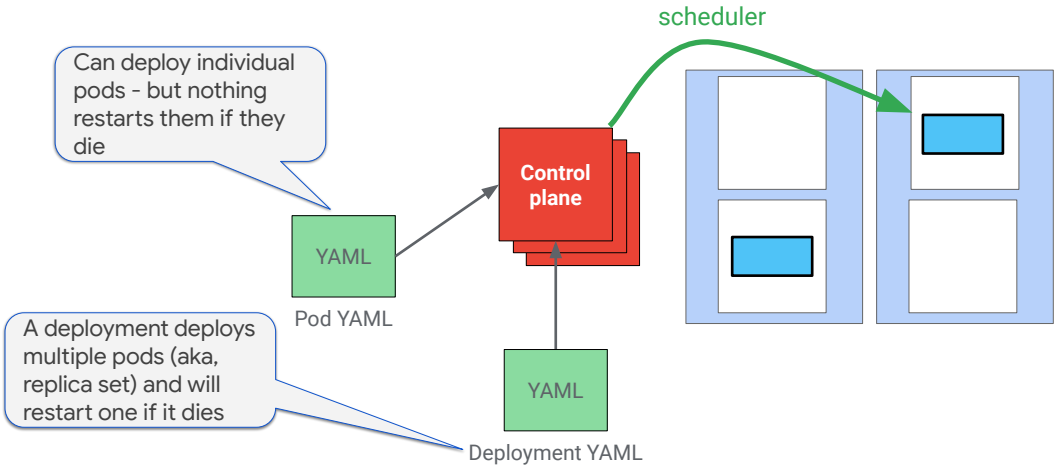
Each pod hosts, manages, and runs one or more containers. The containers in a pod share networking and storage.

So typically, there is one container per pod, unless the containers hold closely related applications. For example, a second container might contain the logging system for the application in the first container.

A pod can be moved from one node to another without reconfiguring or rebuilding anything.

This design enables advanced controls and operations that gives systems built on Kubernetes unique qualities.

## Kubernetes deploys containers to nodes






Pods: https://cloud.google.com/kubernetes-engine/docs/concepts/pod

Creating pods:
https://cloud.google.com/kubernetes-engine/docs/concepts/pod#creating_pods

Pods can be creating by a pod spec yaml file and using the “kubectl apply” command:
https://kubernetes.io/docs/concepts/workloads/pods/

Spec: https://kubernetes.io/docs/concepts/workloads/pods/

A pod template (which contains a pod spec) is included in a Deployment yaml file. This tells GKE how many pods to create and keep running at all times.
Pod template: https://kubernetes.io/docs/concepts/workloads/pods/#pod-templates

Deployments: https://kubernetes.io/docs/concepts/workloads/controllers/deployment/

Each cluster has a control plane node that determines what happens on the cluster. There are usually at least three of them for availability. And they can be located across zones. A Kubernetes job makes changes to the cluster.

For example a pod YAML file provides the information to start up and run a pod on a node. If for some reason a pod stops running or a node is lost, the pod will not automatically be replaced. The Deployment YAML tells Kubernetes how many pods

you want running. So the Kubernetes deployment is what keeps a number of pods running. The Deployment YAML also defines a Replica Set, which is how many copies of a container you want running. The Kubernetes scheduler determines on which node and in which pod the replica containers are to be run.

# Kubernetes Deployment Configuration Example

```
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: devops-deployment
  labels:
    <Some code omitted to save space>
spec:
  replicas: 3
  selector:
    <Some code omitted to save space>
  template:
    <Some code omitted to save space>
  spec:
      containers:
      - name: devops-demo
        image: us-central1-docker.pkg.dev/si/si-image:latest
        ports:
        - containerPort: 8080
```

Artifact Registry



Overview of deploying workloads
https://cloud.google.com/kubernetes-engine/docs/how-to/deploying-workloads-overview

Deploying a stateless Linux application:
https://cloud.google.com/kubernetes-engine/docs/how-to/stateless-apps

The illustration is an example of a deployment configuration.

The top portion of the bolded text depicts setting file as a deployment.

The middle bolded text section shows replicas set to 3. So this deployment will always have 3 pods running.

The bottom portion of the text sets the container image used for the pod.

# Deploying to Kubernetes via command line

Create cluster

```
gcloud container clusters create si-cluster \
  --zone us-central1-a --machine-type=e2-micro \
  --num-nodes 2
```

Connect and apply yaml file

```
gcloud container clusters get-credentials si-cluster --zone us-central1-a

kubectl apply -f devops-deployment.yaml
```

Show the running pods

```
kubectl get pods
```

Show all the deployments

```
kubectl get deployments
```



`gcloud container` syntax
https://cloud.google.com/sdk/gcloud/reference/container

# Creating a Load Balancer service via command line

Create a load balancer to route requests to the pods

```
kubectl expose deployment devops-deployment
--port=80 --target-port=8080 --type=LoadBalancer
```

To get the load balancer public IP address, use the following command:

```
kubectl get services
```

More about services later



GKE Services
https://cloud.google.com/kubernetes-engine/docs/concepts/service

We need to expose our application so users can connect to it. We use the kubectl expose deployment command for this. The command sets the port to 80, the target port to 8080, and the type of service to LoadBalancer.

Once the configuration is complete, we can run the kubectl get services command to see the details of the load balancer, including the public IP address.

## Scaling a Deployment

Scaling is discussed in more depth in the next section

- Use scale command to manually change the number of instances

```
kubectl scale deployment devops-deployment
--replicas=10
```

- To dynamically scale up and down, create an autoscaler
  - Specify min and max number of machines and some metric to monitor

```
kubectl autoscale deployment devops-deployment --min=5
--max=10 --cpu-percent=60
```



A deployment can be scaled using the Console or the command line. In the first example, the command scales the deployment to 10 replicas. Regardless of resource usage, there will always be 10.

To configure autoscaling via the command line, use the `kubectl autoscale` command. In the example, the deployment will run 5 replicas as a minimum. Based on CPU percentage, the replicas scale to a maximum of 10.

## Deleting Deployments and Resources

- Use the delete command to destroy anything previously created
  - Specifying a configuration file will delete everything created from it

```
kubectl delete -f devops-deployment.yaml
```

- Can also delete resources individually when created at the command line

```
kubectl delete services [name here]
```



Finally, here are a few examples on how to delete resources. Use the delete command to destroy anything previously created.

In the first example, you can delete resources based on changes made to the deployment file.

You can also delete resources individually. The second example shows how to do that.

## Summary of Kubernetes Terms

- **Pods** are the smallest unit of deployment
  - Usually pods represent a single container
- **Clusters** are collections of machines that will run containers
  - Each machine is a node in the cluster
- **Replica sets** are used to create multiple instances of a pod
  - Guarantee pods are healthy and the right number exist
- **Load balancers** route requests to pods
- **Autoscalers** monitor load and create or delete pods
- **Deployments** are configurations that define service resources



A few terms to understand when working with Kubernetes.

Pods are the smallest unit of deployment. A pod can be comprised of one or more containers, but typically it is one pod per container.

Clusters are a collection of instances the containers can run on. Each instance is a node in a cluster.

Replica sets are used to create multiple instances of a pod. The replica sets can guarantee the desired number is always running and only healthy pods are used.

Kubernetes can leverage cloud load balancers to distribute traffic to multiple pods. The load balancer is run as a network service.

To ensure the proper performance of your application, you can configure autoscalers to add and remove pods.

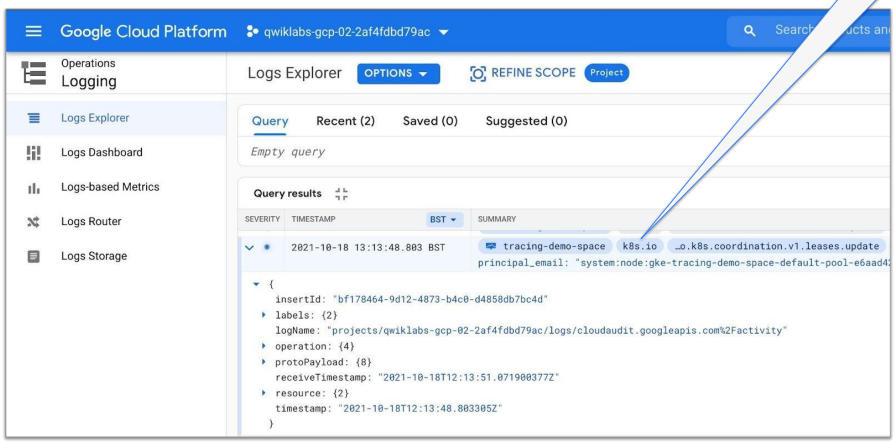
Deployments are configurations that define service resources.

# Exam Guide - Kubernetes Engine

2.2 Planning and configuring compute resources. Considerations include:
- 2.2.1 Selecting appropriate compute choices for a given workload (e.g., Compute Engine, Google Kubernetes Engine, Cloud Run, Cloud Functions)
- 2.2.2 Using preemptible VMs and custom machine types as appropriate

3.2 Deploying and implementing Google Kubernetes Engine resources. Tasks include:
- 3.2.1 Installing and configuring the command line interface (CLI) for Kubernetes (kubectl)
- 3.2.2 Deploying a Google Kubernetes Engine cluster with different configurations including AutoPilot, regional clusters, private clusters, etc.
- 3.2.3 Deploying a containerized application to Google Kubernetes Engine
- 3.2.4 Configuring Google Kubernetes Engine monitoring and logging

# Cloud Logging and GKE



# Configuring monitoring and logging support for a new cluster

- Logging and Monitoring are enabled by default in GKE
- Can configure which logs are sent to Cloud Logging

```
gcloud container clusters create [CLUSTER_NAME] \
  --zone=[ZONE] \
  --project=[PROJECT_ID] \
  --logging=SYSTEM, WORKLOAD
```

Options in both commands are SYSTEM, WORKLOAD or NONE

- Can also configure which metrics are sent to Cloud Monitoring

```
gcloud container clusters create [CLUSTER_NAME] \
  --zone=[ZONE] \
  --project=[PROJECT_ID] \
  --monitoring=SYSTEM
```

SYSTEM: audit logs for Admin activity, data access logs and events logs

WORKLOAD: logs produced from applications running in the containers



Configuring Cloud Operations for GKE:
https://cloud.google.com/stackdriver/docs/solutions/gke/installing

Managing GKE logs:
https://cloud.google.com/stackdriver/docs/solutions/gke/managing-logs

Introducing Kubernetes control plane metrics in GKE (Blog Sep 8, 2022):
https://cloud.google.com/blog/products/containers-kubernetes/kubernetes-control-plane-metrics-are-generally-available

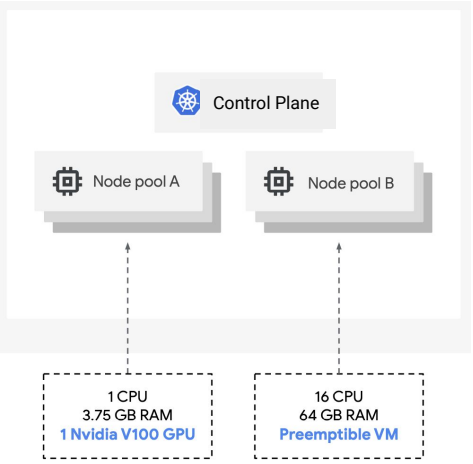
# Exam Guide - Kubernetes Engine

4.2 Managing Google Kubernetes Engine resources. Tasks include:

- 4.2.1 Viewing current running cluster inventory (nodes, pods, services)
- 4.2.2 Browsing Docker images and viewing their details in the Artifact Registry
- 4.2.3 Working with node pools (e.g., add, edit, or remove a node pool)
- 4.2.4 Working with pods (e.g., add, edit, or remove pods)
- 4.2.5 Working with services (e.g., add, edit, or remove a service)
- 4.2.6 Working with stateful applications (e.g. persistent volumes, stateful sets)
- 4.2.7 Managing Horizontal and Vertical autoscaling configurations
- 4.2.8 Working with management interfaces (e.g., Google Cloud console, Cloud Shell, Cloud SDK, kubectl)

# Node pools are cluster subsets with identical machine configurations

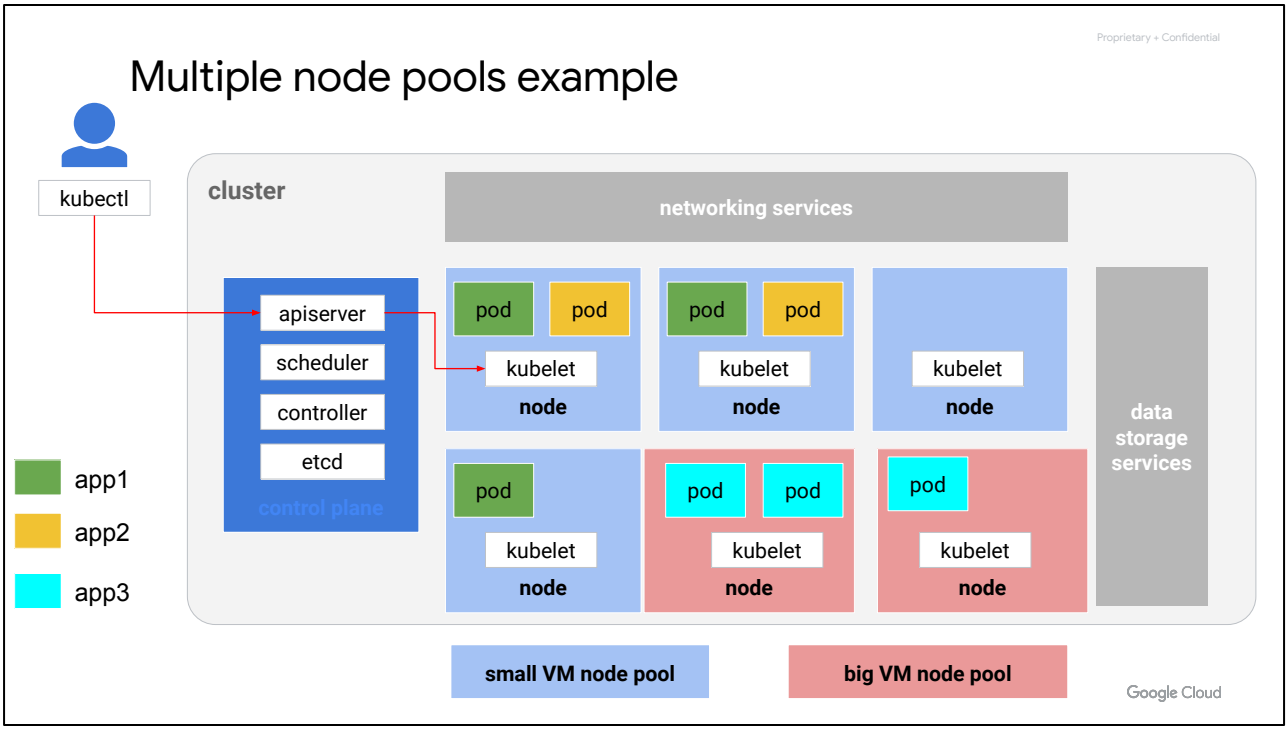
- Share the same hardware, OS, and GKE version
  - Sizing, scaling, and upgrades, operate per pool
  - Can cover a full region – or just select zones
- Additional pools of different sizes and types can be added after cluster creation
  - For example, a pool with local SSDs, or Spot VMs, or a specific image or a different machine type
- Common configurations:
  - Each stateful application gets their own node pool
  - Batch jobs in a node pool with Preemptible VMs






About node pools
https://cloud.google.com/kubernetes-engine/docs/concepts/node-pools

Running a GKE application on spot nodes with on-demand nodes as fallback:
https://cloud.google.com/blog/topics/developers-practitioners/running-gke-application-spot-nodes-demand-nodes-fallback




Here we have 2 node pools. The blue pool is running two apps and the red pool has one app deployed. Both pools are controlled by the same control plane.

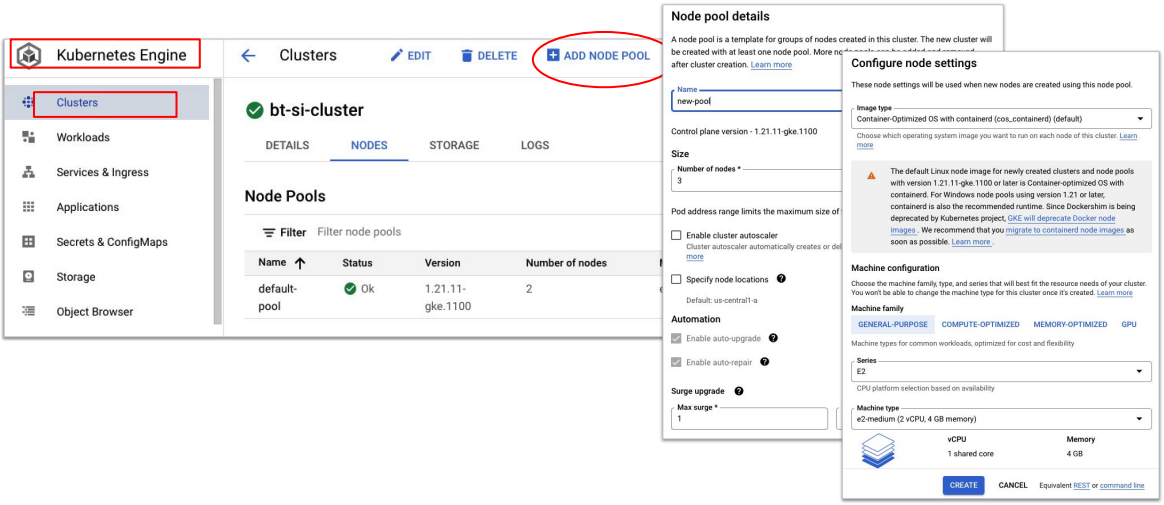
Cluster administrators configure the cluster by sending requests to **apiservers** on the Control Plane using a command-line tool called **kubectl**. Kubectl can be installed and run anywhere.

From there, the apiserver communicates with the cluster in two primary ways:

- To the **kubelet** process that runs on each node
- To any node, pod, or service through the apiserver's proxy functionality (not shown).

Then pods are started on various nodes. In this example, there are three types of pods running (shown in yellow, green and teal).

# Add a node pool in the Console



# Managing node pools with the CLI

- Create a pool

```
gcloud container node-pools create app3-pool --cluster bt-si-cluster
--zone us-central1-a --machine-type e2-medium
--disk-type pd-standard --disk-size 100  --num-nodes 3
```

- Resize a pool

```
gcloud container clusters resize bt-si-cluster \
    --node-pool app3-pool \
    --num-nodes 10
```

- Delete a pool

```
gcloud container node-pools delete app3-pool \
    --cluster bt-si-cluster
```



Add and manage node pools:
https://cloud.google.com/kubernetes-engine/docs/how-to/node-pools

Delete a node pool:
https://cloud.google.com/kubernetes-engine/docs/how-to/node-pools#deleting_a_node_pool

# Exam Guide - Kubernetes Engine



4.2 Managing Google Kubernetes Engine resources. Tasks include:

- 4.2.1 Viewing current running cluster inventory (nodes, pods, services)
- 4.2.2 Browsing Docker images and viewing their details in the Artifact Registry
- 4.2.3 Working with node pools (e.g., add, edit, or remove a node pool)
- 4.2.4 Working with pods (e.g., add, edit, or remove pods)
- 4.2.5 Working with services (e.g., add, edit, or remove a service)
- 4.2.6 Working with stateful applications (e.g. persistent volumes, stateful sets)
- 4.2.7 Managing Horizontal and Vertical autoscaling configurations
- 4.2.8 Working with management interfaces (e.g., Google Cloud console, Cloud Shell, Cloud SDK, kubectl)

# Services allow communication to/from pods

- Pods in a deployment are regularly created and destroyed, causing their IP addresses to change constantly
  - Makes it difficult for frontend applications to identify which pods to connect to
- Services consist of
  - A set of pods
  - A policy to access them
- The most common types of  Kubernetes services are
  - ClusterIP
  - NodePort
  - LoadBalancer
  - Ingress (not really a service)



Services
https://cloud.google.com/kubernetes-engine/docs/concepts/service

Another overview:
https://kubernetes.io/docs/concepts/services-networking/service/

Exposing applications using services (includes add, edit and remove):
https://cloud.google.com/kubernetes-engine/docs/how-to/exposing-apps

# Service details

- ClusterIP
  - Default service
  - Internal to the cluster and allows applications within the pods to communicate with each other
- NodePort
  - Opens a specific port on each nodes in the cluster
  - Traffic sent to that port is forwarded to the pods
- LoadBalancer
  - Standard way to expose a Kubernetes service externally so it can be accessed over the internet
  - In GKE this creates a external TCP Network Load Balancer with one IP address accessible to external users
- Ingress
  - In GKE, the ingress controller creates an HTTP(S) Load Balancer, which can route traffic to services in the Kubernetes cluster based on path or subdomain

# Creating a Load Balancer service via command line


Create a load balancer to route requests to the pods

```
kubectl expose deployment devops-deployment
--port=80 --target-port=8080 --type=LoadBalancer
```

To get the load balancer public IP address, use the following command:

```
kubectl get services
```

To delete the service

```
kubectl delete services [name here]
```



We need to expose our application so users can connect to it. We use the kubectl expose deployment command for this. The command sets the port to 80, the target port to 8080, and the type of service to LoadBalancer.

Once the configuration is complete, we can run the kubectl get services command to see the details of the load balancer, including the public IP address.

# Exam Guide - Kubernetes Engine



4.2 Managing Google Kubernetes Engine resources. Tasks include:

- 4.2.1 Viewing current running cluster inventory (nodes, pods, services)
- 4.2.2 Browsing Docker images and viewing their details in the Artifact Registry
- 4.2.3 Working with node pools (e.g., add, edit, or remove a node pool)
- 4.2.4 Working with pods (e.g., add, edit, or remove pods)
- 4.2.5 Working with services (e.g., add, edit, or remove a service)
- 4.2.6 Working with stateful applications (e.g. persistent volumes, stateful sets)
- 4.2.7 Managing Horizontal and Vertical autoscaling configurations
- 4.2.8 Working with management interfaces (e.g., Google Cloud console, Cloud Shell, Cloud SDK, kubectl)

# Kubernetes Engine StatefulSets

- StatefulSets represent a set of Pods with unique, persistent identities and stable hostnames that GKE maintains regardless of where they are scheduled
- Some examples of use cases include:
  - A Redis pod that needs to maintain access to the same storage volume
    - Even if it is redeployed or restarted
  - A Cassandra implementation with database sharding across Pods
  - A MongoDB database where multiple read replicas are needed to serve read-only traffic
- Stateful sets can take advantage of GKE health checks
  - E.g., if a pod containing a read-replica becomes non-responsive, a new pod will be created in its place
    - Its existing storage volume will be re-attached



To run or not to run a database on Kubernetes: What to consider
https://cloud.google.com/blog/products/databases/to-run-or-not-to-run-a-database-on-kubernetes-what-to-consider

Deploying a stateful application:
https://cloud.google.com/kubernetes-engine/docs/how-to/stateful-apps

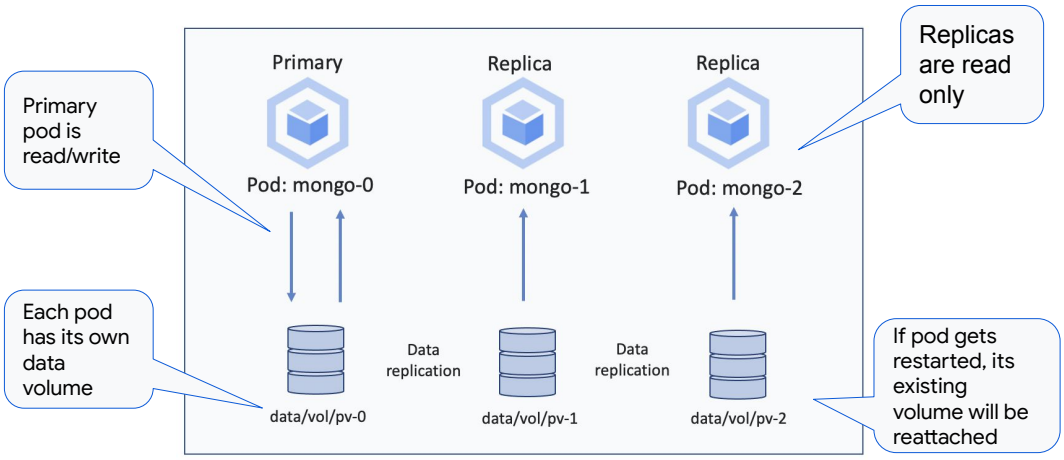
StatefulSet:
https://cloud.google.com/kubernetes-engine/docs/concepts/statefulset

Persistent volumes and dynamic provisioning:
https://cloud.google.com/kubernetes-engine/docs/concepts/persistent-volumes

Suggested tutorial:
https://kubernetes.io/docs/tutorials/stateful-application/basic-stateful-set/

Suggested Lab: Running a MongoDB Database in Kubernetes with StatefulSets
https://partner.cloudskillsboost.google/catalog_lab/327

# StatefulSet Example - MongoDB database




- Synching handled by MongoDB
- Connectivity to the correct pod is handled at the application level



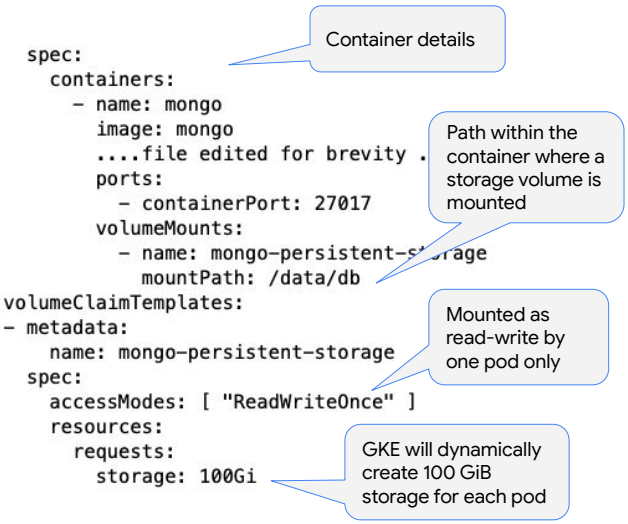
There are several MongoDB implementations in Cloud Marketplace. MongoDB automatically handles the replication.

# StatefulSet Pods are unique

- GKE provides guarantees about the ordering and uniqueness of each Pods
  - Pods are created one after the other, until the max specified is reached
    - Based on an identical container spec (same as a Deployment)
    - A “sticky” identity for each Pod is maintained (different from a Deployment)
      - For example, mypod-0, mypod-1, mypod-2
    - If a Pod fails, it is recreated, given the same identity and matched with its existing volume
  - State information is maintained in persistent disk storage
    - Each Pod gets its own dedicated volume

# PersistentVolume and PersistentVolumeClaim

- PersistentVolume (PV)
  - A piece of storage in the cluster that has been manually provisioned by an administrator, or dynamically provisioned by Kubernetes using a StorageClass
- PersistentVolumeClaim (PVC)
  - A request for storage by a user that can be fulfilled by a PV
  - The claim request
- Both are independent from Pod lifecycles and preserve data through restarting, rescheduling, and even deleting Pods.



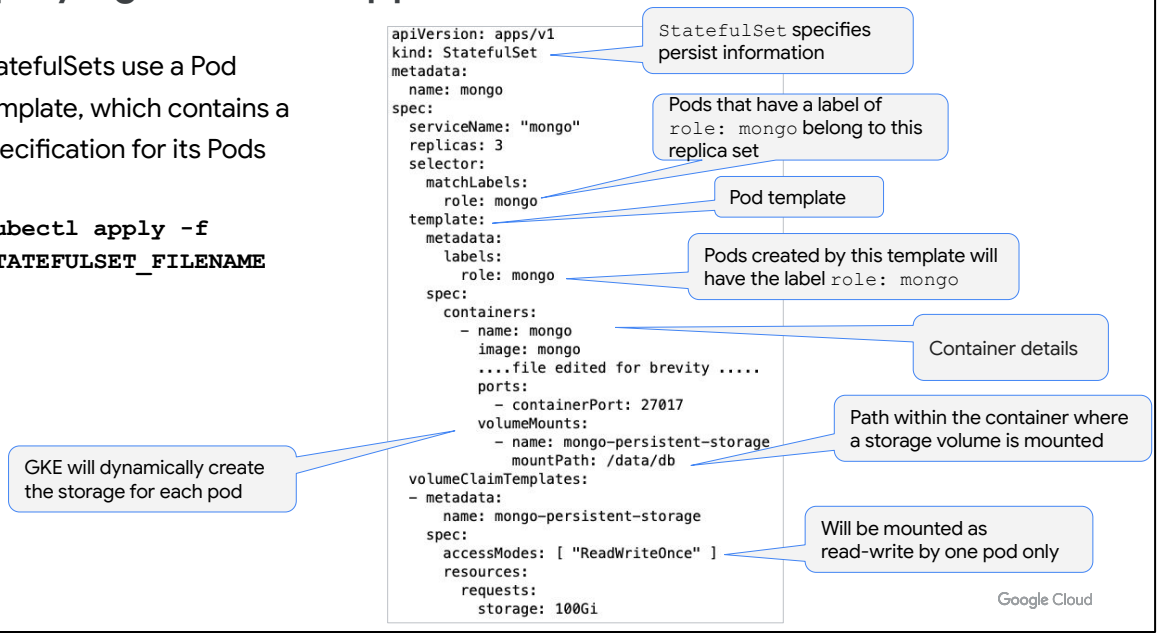



The prior page mentions volumes. This is an example of a manifest that creates a dynamic volume for a MongoDB container

# Deploying a stateful application

- StatefulSets use a Pod template, which contains a specification for its Pods

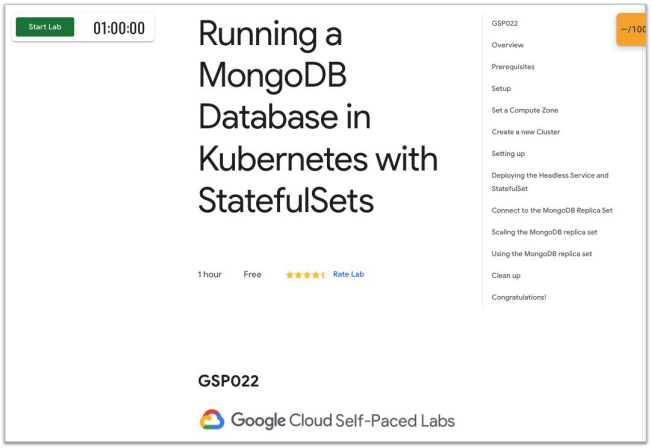
**`kubectl apply -f STATEFULSET_FILENAME`**




Tutorial: https://kubernetes.io/docs/tutorials/stateful-application/basic-stateful-set/

# Suggested Lab: Running a MongoDB Database in Kubernetes with StatefulSets (If time permits)

- Topics include
  - Deploy a Kubernetes cluster and a StatefulSet
  - Connect a Kubernetes cluster to a MongoDB replica set
  - Scale MongoDB replica set instances up and down.




https://partner.cloudskillsboost.google/catalog_lab/327

# Exam Guide - Kubernetes Engine

4.2 Managing Google Kubernetes Engine resources. Tasks include:

- 4.2.1 Viewing current running cluster inventory (nodes, pods, services)
- 4.2.2 Browsing Docker images and viewing their details in the Artifact Registry
- 4.2.3 Working with node pools (e.g., add, edit, or remove a node pool)
- 4.2.4 Working with pods (e.g., add, edit, or remove pods)
- 4.2.5 Working with services (e.g., add, edit, or remove a service)
- 4.2.6 Working with stateful applications (e.g. persistent volumes, stateful sets)
- 4.2.7 Managing Horizontal and Vertical autoscaling configurations
- 4.2.8 Working with management interfaces (e.g., Google Cloud console, Cloud Shell, Cloud SDK, kubectl)

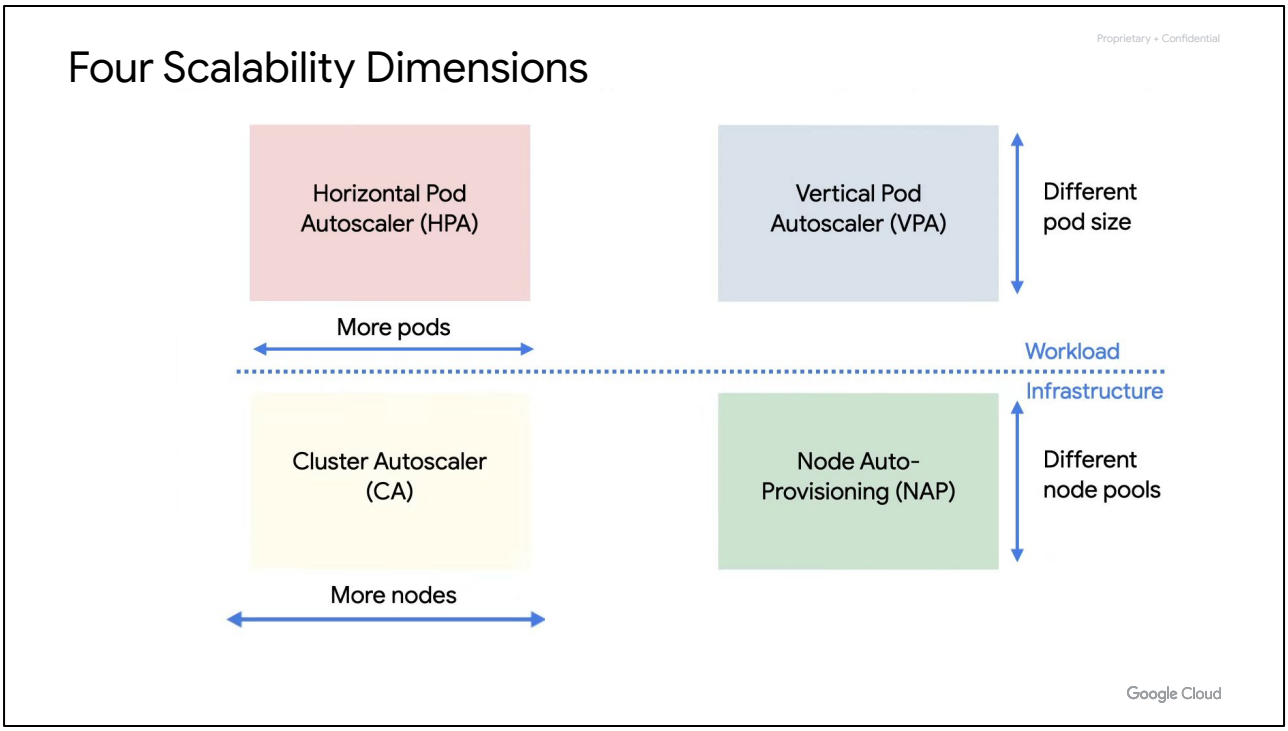
# Autoscaling Strategies

Horizontal Pod Autoscaler (HPA)

Vertical Pod Autoscaler (VPA)

Cluster Autoscaler (CA)

Node Auto Provisioning (NAP)




Configuring multidimensional Pod autoscaling
https://cloud.google.com/kubernetes-engine/docs/how-to/multidimensional-pod-autoscaling

# Resource Management for Pods and Containers

- A Pod spec (optionally) defines how much CPU and memory (RAM) a container needs
  - Used by the scheduler to determine which node to place the Pod on
- Requests
  - Minimum amount of CPU/Memory needed
- Limits
  - Maximum allowed
  - The system kernel terminates processes that attempt to allocate memory over the max allowed, with an out of memory (OOM) error

```yaml
---
apiVersion: v1
kind: Pod
metadata:
  name: frontend
spec:
  containers:
  - name: app
    image: images.my-company.example/app:v
    resources:
      requests:
        memory: "64Mi"
        cpu: "250m"
      limits:
        memory: "128Mi"
        cpu: "500m"
  - name: log-aggregator
    image: images.my-company.example/log-aggregator:v6
    resources:
      requests:
        memory: "64Mi"
        cpu: "250m"
      limits:
        memory: "128Mi"
        cpu: "500m"
```

Pod has 2 containers

Total Pod Requests:
CPU: 0.5 cores
Memory: 128 MiB

Total Pod Limits:
CPU: 1.0 core
Memory: 256 MiB

1 CPU unit i=1 physical CPU core, or 1 virtual core, usually measured in millicores (1,000 millicores - 1 vCPU)

Limits and requests for memory are measured in bytes

Units are case sensitive: m = millibyte, M = megabyte

Google Cloud

Image from:
https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/

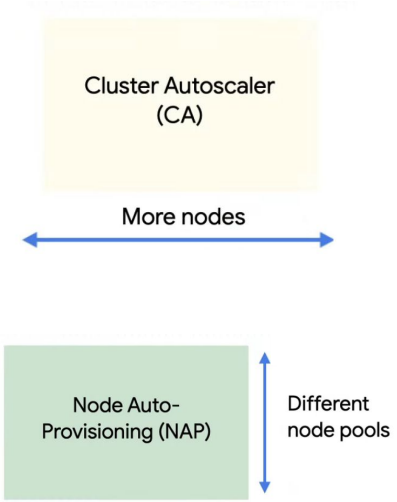
In Kubernetes, 1 CPU unit is equivalent to **1 physical CPU core**, or **1 virtual core**, usually measured in millicores (1,000 millicores - 1 vCPU)

Limits and requests for memory are measured in bytes

For more details on Requests and Limits, see:
https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/

# How Pods with resource requests are scheduled

- Kubernetes Scheduler selects a node for the Pod
  - Each node has a maximum capacity for the amount of CPU and memory it can provide for Pods
  - The Scheduler ensures that the sum of the resource requests of the scheduled containers is less than the capacity of the node
  - When total node capacity is reached, additional Pods are marked as “unschedulable”
    - Solutions:
      - Implement Cluster Autoscaler to add more nodes
      - Use Node Auto-Provisioning (NAP) to automatically create node pools on a as-needed basis
      - Manually scale # of nodes/add node pools



# Autoscaling nodes

- **Cluster autoscaler** -x add/remove nodes based on *resource requests* of pods running in the node pool

```
gcloud container clusters create example-cluster \
  --num-nodes 2 \
  --zone us-central1-a \
  --node-locations us-central1-a,us-central1-b,us-central1-f \
    --enable-autoscaling --min-nodes 1 --max-nodes 10
```

- **Node auto provisioning** - add/delete node pools as needed (managed by Google)
  - Feature is automatically enabled in Autopilot clusters
  - To enable it in Standard GKE

```
gcloud container clusters update dev-cluster
    --enable-autoprovisioning \
    --min-cpu 1 \
    --min-memory 1 \
    --max-cpu 10 \
    --max-memory 64
```

Scale between a total cluster size of 1 CPU / 1 GB memory to a maximum of 10 CPU / 64 GB memory



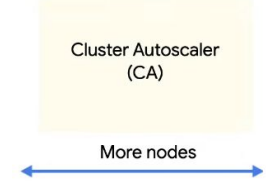
Cluster nodes scaling
https://cloud.google.com/kubernetes-engine/docs/concepts/cluster-autoscaler

Using node auto-provisioning
https://cloud.google.com/kubernetes-engine/docs/how-to/node-auto-provisioning

Enabling node auto-provisioning:
https://cloud.google.com/kubernetes-engine/docs/how-to/node-auto-provisioning#enable

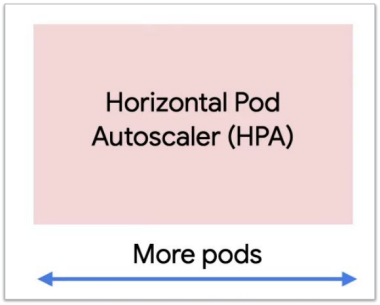
Youtube - Autoscaling with GKE: Clusters and nodes:
https://www.youtube.com/watch?v=VNAWA6NkoBs&t=106s

# Summary: Cluster Autoscaling vs Node Auto-provisioning

| | | |
|---|---|---|
|  | Automatically resizes node pools based on the workload demands<br><br>High demand: adds nodes to the node pool.<br>Low demand: scales back down to a minimum size | Requires active monitoring of node resource usage to ensure haven't over/under provisioned node resources<br>● Under provisioned: Extra overhead of adding additional nodes when needed (+ node cost)<br>● Over provisioned: Paying too much per node |
|  | Google optimizes the compute resources in each node pool to match workload requirements | ● Not best for sudden spikes in traffic<br>● Will take longer to create a new node pool vs adding a node to an existing pool |

# Horizontal Pod Autoscaler

- Automatically add more pods to a Deployment or StatefulSet when workload increases
  - A control loop runs intermittently (default = 15 seconds) and compares resource utilization against the metrics specified
- Metrics are based on
  - **Actual (or percentage) resource usage**: when a given Pod's CPU or memory usage exceeds a specified threshold.
  - **Custom metrics:** based on any metric reported by a Kubernetes object, such as the rate of client requests per second or I/O writes per second.
    - Useful if your application is prone to network or storage bottlenecks, rather than CPU or memory.
  - **External metrics:** based on a metric from an application or service external to your cluster, e.g., size of a Pub/Sub queue



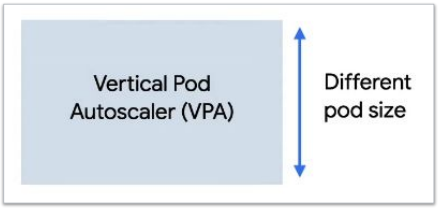



Horizontal pod scaling
https://cloud.google.com/kubernetes-engine/docs/concepts/horizontalpodautoscaler

Custom and external metrics for Horizontal autoscaling workloads:
https://cloud.google.com/kubernetes-engine/docs/concepts/custom-and-external-metrics

# Vertical Pod Autoscaler

- Provides recommendations for resource usage over time
  - Use horizontal pod autoscaler for sudden increases
- Vertical Pod autoscaling recommends values for CPU and memory requests for Pods
  - In lieu of you having to monitor and manually set CPU/Memory requests and limits for the containers in your Pods,
  - Recommendations can be used to manually update your Pods (delete and recreate), or can configure vertical Pod autoscaling to automatically update the values
- Vertical Pod autoscaling notifies the cluster autoscaler ahead of the update to provide the resources needed for the resized workload before recreating it

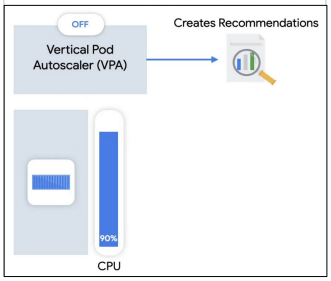


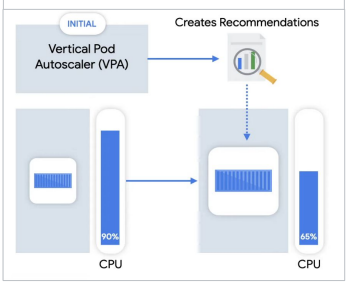


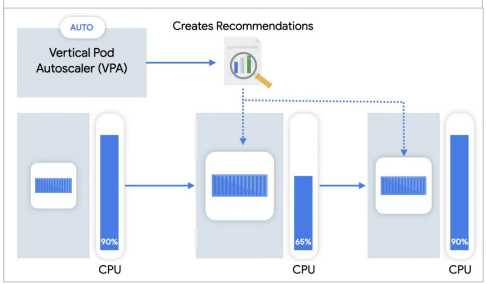

Vertical pod scaling
https://cloud.google.com/kubernetes-engine/docs/concepts/verticalpodautoscaler

# Vertical Pod Autoscaler Modes

| Off | Initial | Auto |
|---|---|---|
| ● No autoscaling; recommendations only | ● Recommendations used to create resized pods<br>● Won't resize them again | ● Recommendations used to regularly resize pods by deleting/recreating them as needed |



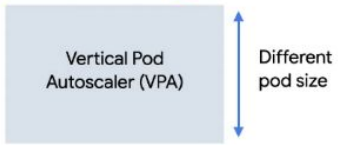
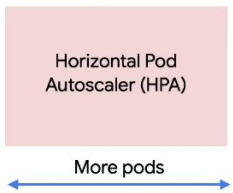




- Google recommends leaving VPA off for at least a week to accumulate recommendations
  - Then switch to Initial or Auto based on needs



Youtube - Autoscaling with GKE
https://www.youtube.com/watch?v=7naCIxIaV1M

# Summary: Vertical vs Horizontal Pod Autoscaling

| | | |
|---|---|---|
|  | Use when are unsure of the optimal resource requests the container application needs<br><br>Can use once or on a ongoing basis | If you do it manually:<br>• Over-estimating memory/cpu results in over-provisioning the # of nodes needed when pods scale up, increasing costs<br>• Under-estimating resources means that pods will be killed when the max limit is reached |
|  | Use when have sudden spikes in traffic | Can use in conjunction with Vertical Pod Autoscaler to ensure pod resource requests are optimized |

# Exam Guide - Kubernetes Engine



4.2 Managing Google Kubernetes Engine resources. Tasks include:

- 4.2.1 Viewing current running cluster inventory (nodes, pods, services)
- 4.2.2 Browsing Docker images and viewing their details in the Artifact Registry
- 4.2.3 Working with node pools (e.g., add, edit, or remove a node pool)
- 4.2.4 Working with pods (e.g., add, edit, or remove pods)
- 4.2.5 Working with services (e.g., add, edit, or remove a service)
- 4.2.6 Working with stateful applications (e.g. persistent volumes, stateful sets)
- 4.2.7 Managing Horizontal and Vertical autoscaling configurations
- 4.2.8 Working with management interfaces (e.g., Google Cloud console, Cloud Shell, Cloud SDK, kubectl)

No further discussion needed



Console:
https://cloud.google.com/cloud-console

Cloud Shell:
https://cloud.google.com/shel

Cloud SDK:
https://cloud.google.com/sdk

# Exam Guide - Cloud Run and Cloud Functions (and App Engine)

Cloud Run

Cloud Functions

App Engine

**2.2 Planning and configuring compute resources. Considerations include:**

- 2.2.1 Selecting appropriate compute choices for a given workload (e.g., Compute Engine, Google Kubernetes Engine, Cloud Run, Cloud Functions)
- 2.2.2 Using preemptible VMs and custom machine types as appropriate

**3.3 Deploying and implementing Cloud Run and Cloud Functions resources.**

- 3.3.1 Deploying an application and updating scaling configuration, versions, and traffic splitting
- 3.3.2 Deploying an application that receives Google Cloud events (e.g., Pub/Sub events, Cloud Storage object change notification events)

**4.3 Managing Cloud Run resources. Tasks include:**

- 4.3.1 Adjusting application traffic-splitting parameters
- 4.3.2 Setting scaling parameters for autoscaling instances
- 4.3.3 Determining whether to run Cloud Run (fully managed) or Cloud Run for Anthos

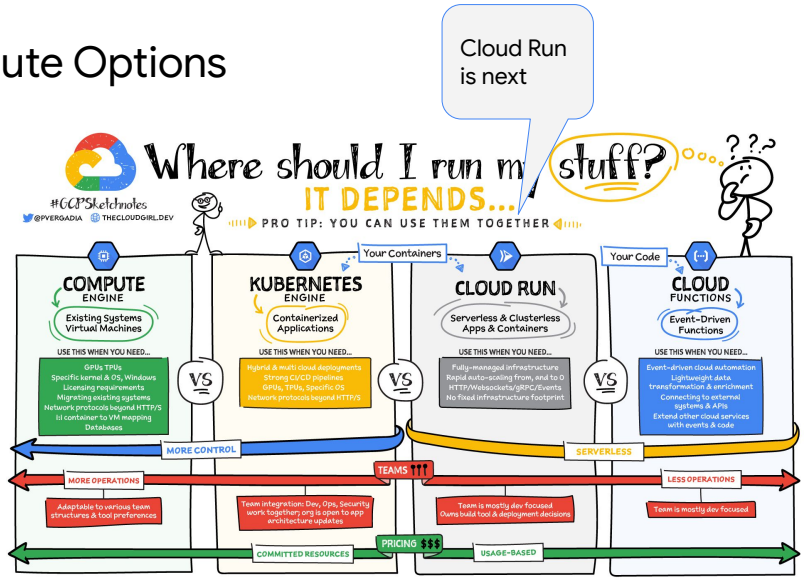
# Exam Guide - Cloud Run and Cloud Functions

2.2 Planning and configuring compute resources. Considerations include:
- 2.2.1 Selecting appropriate compute choices for a given workload (e.g., Compute Engine, Google Kubernetes Engine, Cloud Run, Cloud Functions)
- 2.2.2 Using preemptible VMs and custom machine types as appropriate

3.3 Deploying and implementing Cloud Run and Cloud Functions resources. Tasks include, where applicable:
- 3.3.1 Deploying an application and updating scaling configuration, versions, and traffic splitting
- 3.3.2 Deploying an application that receives Google Cloud events (e.g., Pub/Sub events, Cloud Storage object change notification events)

4.3 Managing Cloud Run resources. Tasks include:
- 4.3.1 Adjusting application traffic-splitting parameters
- 4.3.2 Setting scaling parameters for autoscaling instances
- 4.3.3 Determining whether to run Cloud Run (fully managed) or Cloud Run for Anthos

# Compute Options




[Where should I run my stuff? Choosing a Google Cloud compute option]

# Exam Guide - Cloud Run and Cloud Functions

3.3 Deploying and implementing Cloud Run and Cloud Functions resources. Tasks include, where applicable:
- 3.3.1 Deploying an application and updating scaling configuration, versions, and traffic splitting
- 3.3.2 Deploying an application that receives Google Cloud events (e.g., Pub/Sub events, Cloud Storage object change notification events)

4.3 Managing Cloud Run resources. Tasks include:
- 4.3.1 Adjusting application traffic-splitting parameters
- 4.3.2 Setting scaling parameters for autoscaling instances
- 4.3.3 Determining whether to run Cloud Run (fully managed) or Cloud Run for Anthos

# Cloud Run provides Containers-as-a-Service

- Serverless platform that runs individual containers.
- Based on the open-source knative project (https://knative.dev/)
- **Two deployment methods**
  - **Fully managed**
    - Use when want Google Cloud to manage autoscaling, connectivity, high availability, etc.
  - Cloud Run for Anthos
    - **Knative running on a GKE cluster**
      - Use when need to:
        - Access VPC network
        - Tune size of compute engine instance, use GPUs, etc.
        - Are running knative containers on-premise or in other clouds and want a single pane of glass (Anthos) to manage them



Cloud Run
https://cloud.google.com/run

Knative
https://knative.dev/
https://cloud.google.com/knative

Cloud Run: What no one tells you about Serverless (and how it's done)
https://cloud.google.com/blog/topics/developers-practitioners/cloud-run-story-serverless-containers

Choosing between Cloud Run and Cloud Run for Anthos:
https://cloud.google.com/anthos/run/docs/choosing-a-platform

# Exam Guide - Cloud Run and Cloud Functions

3.3 Deploying and implementing Cloud Run and Cloud Functions resources. Tasks include, where applicable:

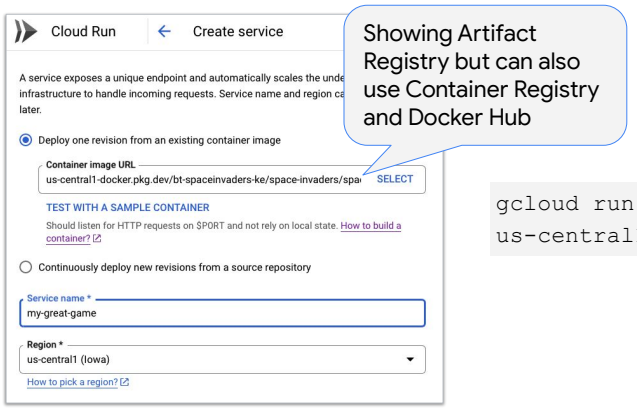
- 3.3.1 Deploying an application and updating scaling configuration, versions, and traffic splitting
- 3.3.2 Deploying an application that receives Google Cloud events (e.g., Pub/Sub events, Cloud Storage object change notification events)

4.3 Managing Cloud Run resources. Tasks include:

- 4.3.1 Adjusting application traffic-splitting parameters
- 4.3.2 Setting scaling parameters for autoscaling instances
- 4.3.3 Determining whether to run Cloud Run (fully managed) or Cloud Run for Anthos

# Deploying Cloud Run

- Can be deployed from the Console or the command line



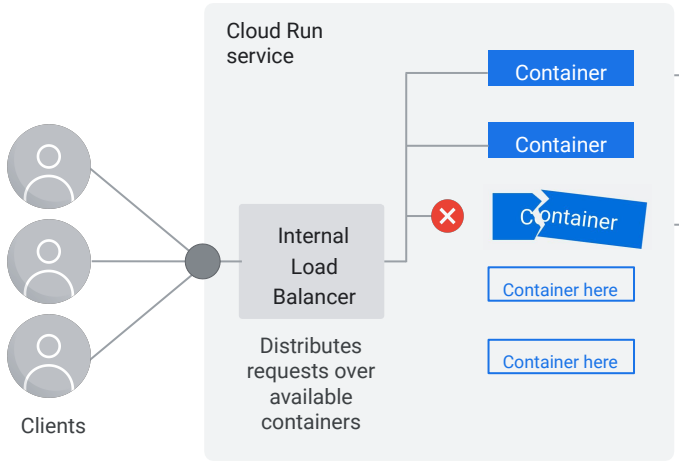

```
gcloud run deploy my-great-game --image
us-central1-docker.pkg.dev/bt-spaceinva...
```



Deploying to Cloud Run
https://cloud.google.com/run/docs/deploying

Deploying container images:
https://cloud.google.com/run/docs/deploying#command-line

# All incoming requests are handled with automatic scaling




Auto scaling: Add containers to handle requests and remove them if they are unused

Auto healing: Monitor and replace unhealthy containers



About container instance autoscaling:
https://cloud.google.com/run/docs/about-instance-autoscaling

# Cloud Run Scaling Parameters

- Specify a minimum and maximum number of instances

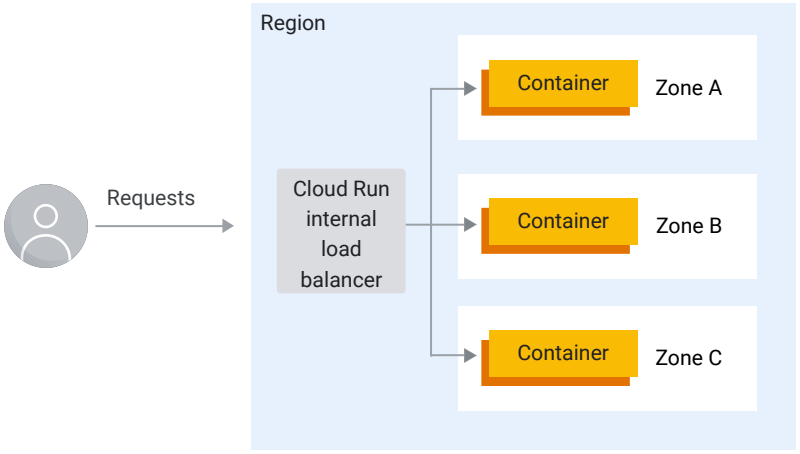

```
gcloud run deploy my-great-game --image
us-central1-docker.pkg.dev/bt-spaceinva…
--min-instances=0 --max-instances=100
```



Minimum instances (services)
https://cloud.google.com/run/docs/configuring/min-instances

Maximum number of container instances (services)
https://cloud.google.com/run/docs/configuring/max-instances

# Cloud Run automatically balances containers across zones in a region

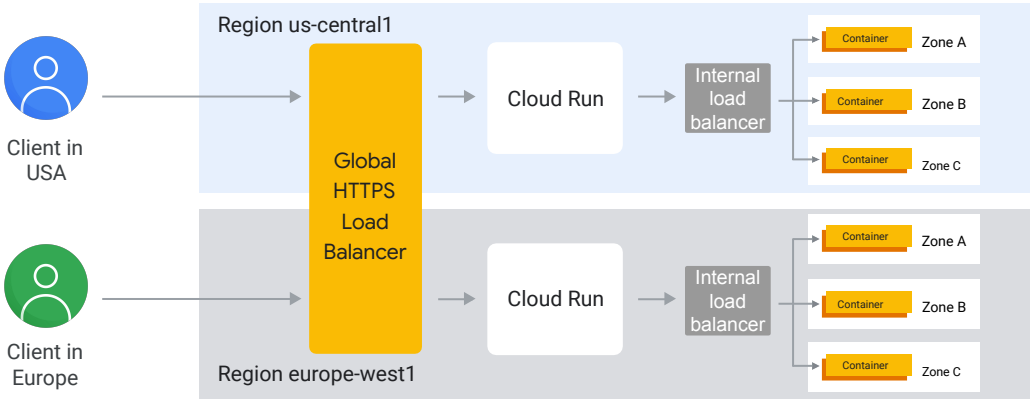



A region is a data center. For example: Council Bluffs (Iowa, North America)

Every region has three or more zones.

It's unlikely for a zone to go down, and a multi-zone failure is highly unlikely.

# Deploy Cloud Run to multiple regions and use a global load balancer to deliver lowest end-user latency






Set up a global external HTTP(S) load balancer with Cloud Run, App Engine, or Cloud Functions
https://cloud.google.com/load-balancing/docs/https/setup-global-ext-https-serverless

Google Cloud global HTTP(S) load balancer can load balancer to backend services, including Cloud Run

# Exam Guide - Cloud Run and Cloud Functions

3.3 Deploying and implementing Cloud Run and Cloud Functions resources. Tasks include, where applicable:

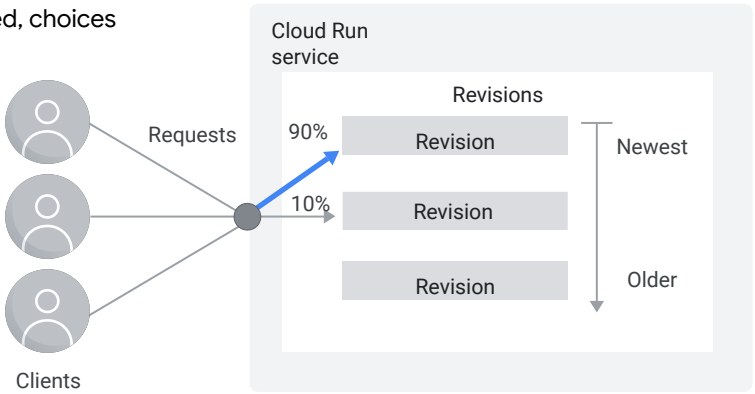
- 3.3.1 Deploying an application and updating scaling configuration, versions, and traffic splitting
- 3.3.2 Deploying an application that receives Google Cloud events (e.g., Pub/Sub events, Cloud Storage object change notification events)

4.3 Managing Cloud Run resources. Tasks include:

- 4.3.1 Adjusting application traffic-splitting parameters
- 4.3.2 Setting scaling parameters for autoscaling instances
- 4.3.3 Determining whether to run Cloud Run (fully managed) or Cloud Run for Anthos

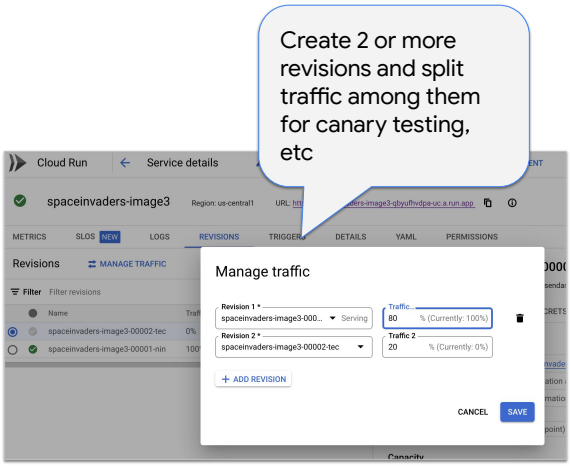
# Application updates are created as new revisions

- When a new revision is deployed, choices are
  - Serve 100% of traffic
  - Serve no traffic



# Cloud Run traffic splitting

- Split traffic across two or more revision
  - Great for canary deployments
  - Gradual rollouts for revisions
- Easily roll back to a previous revision




```
gcloud run services update-traffic my-great-game
--to-revisions LATEST=5
```

Send 5% of traffic to the latest revision



Rollbacks, gradual rollouts, and traffic migration:
https://cloud.google.com/run/docs/rollouts-rollbacks-traffic-migration

Traffic splitting:
https://cloud.google.com/run/docs/rollouts-rollbacks-traffic-migration

# Exam Guide - Cloud Run and Cloud Functions

3.3 Deploying and implementing Cloud Run and Cloud Functions resources. Tasks include, where applicable:

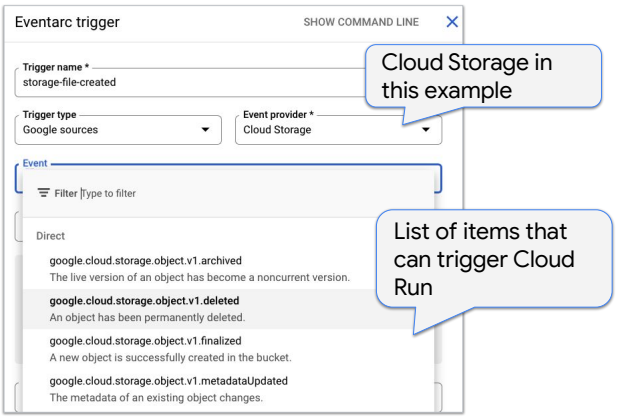
- 3.3.1 Deploying an application and updating scaling configuration, versions, and traffic splitting
- 3.3.2 Deploying an application that receives Google Cloud events (e.g., Pub/Sub events, Cloud Storage object change notification events)

4.3 Managing Cloud Run resources. Tasks include:

- 4.3.1 Adjusting application traffic-splitting parameters
- 4.3.2 Setting scaling parameters for autoscaling instances
- 4.3.3 Determining whether to run Cloud Run (fully managed) or Cloud Run for Anthos

# Invoking Cloud Run

- Can be invoked by
  - HTTPS endpoint
  - gRPC
  - Websockets
  - Pub/Sub
  - Eventarc triggers
    - Cloud Storage
    - BigQuery
    - And more




```
gcloud eventarc triggers create storage-file-created \
--more here shortened for brevity
--event-filters="type=google.cloud.storage.object.v1.finalized"
```



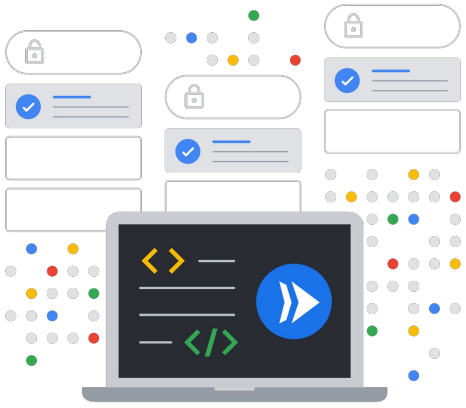
Eventarc overview
https://cloud.google.com/eventarc/docs/overview

Triggering from Pub/Sub push:
https://cloud.google.com/run/docs/triggering/pubsub-push

Cloud Storage (+ other types of events):
https://cloud.google.com/eventarc/docs/creating-triggers

# Cloud Run use cases

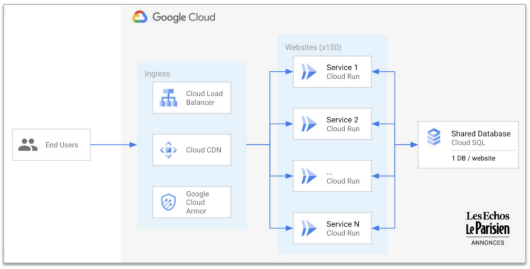
- Web applications, public APIs or websites
- Microservices-based applications that communicate using direct or asynchronous messages
- Event processing
- Scheduled tasks shorter than 60 minutes



Other example use cases found toward the bottom of this page: https://cloud.google.com/run/

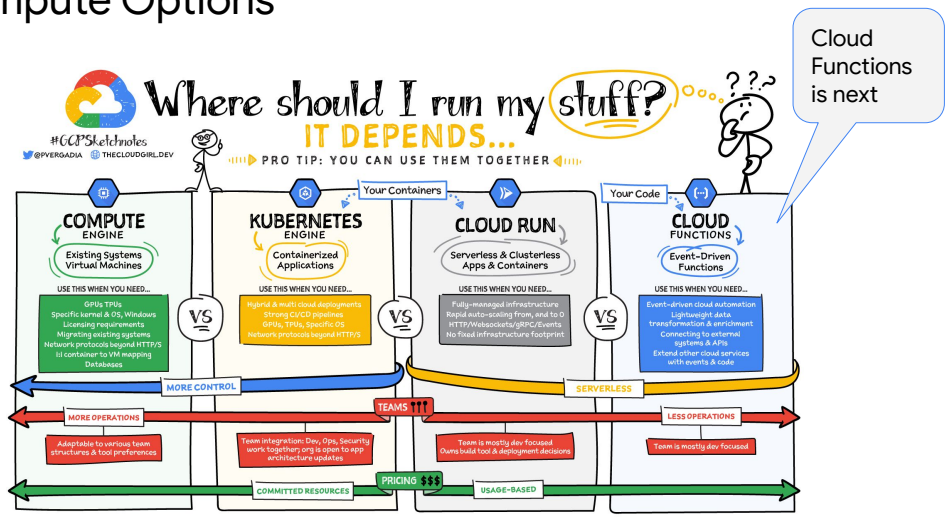
# Cloud Run - customer use case

- Les Echos Le Parisien Annonces publishes legal notices on behalf of various organizations
  - Supports a number of sites for different regions within France, as well as French readers in territories across the world
  - Each local site offers a variety of services and content
- Historically these sites were served from multiple, dedicated on-prem infrastructure
  - Over time, growth was constricted by their monolithic architecture
- Today, each site is containerized and deployed as its own Cloud Run service in Google Cloud




[Scaling quickly to new markets with Cloud Run—a web modernization story]

# Compute Options




[Where should I run my stuff? Choosing a Google Cloud compute option]

# Exam Guide - Cloud Run and Cloud Functions

3.3 Deploying and implementing Cloud Run and Cloud Functions resources. Tasks include, where applicable:

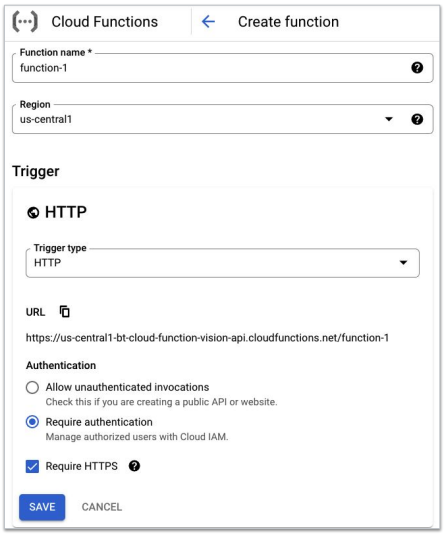
- 3.3.1 Deploying an application and updating scaling configuration, versions, and traffic splitting
- 3.3.2 Deploying an application that receives Google Cloud events (e.g., Pub/Sub events, Cloud Storage object change notification events)

4.3 Managing Cloud Run resources. Tasks include:

- 4.3.1 Adjusting application traffic-splitting parameters
- 4.3.2 Setting scaling parameters for autoscaling instances
- 4.3.3 ~~Determining whether to run Cloud Run (fully managed) or Cloud Run for Anthos~~

# Cloud Functions provide single purpose services

- Used to create specialized services for a single purpose e.g.
  - Process an image
  - Update a user profile in a database
- Scales to zero to save on costs
- Autoscales up/down based on load
- Various programming languages supported
  - Code can be maintained by different developers
- Great for microservices
  - Each scales independently
  - Updates to one doesn't affect the others
- Triggered several ways
  - Storage bucket changes
  - Pub/Sub (queue) topic
  - REST API call
  - Other events






Cloud Functions
https://cloud.google.com/functions

Cloud Functions Overview
https://cloud.google.com/functions/docs/concepts/overview

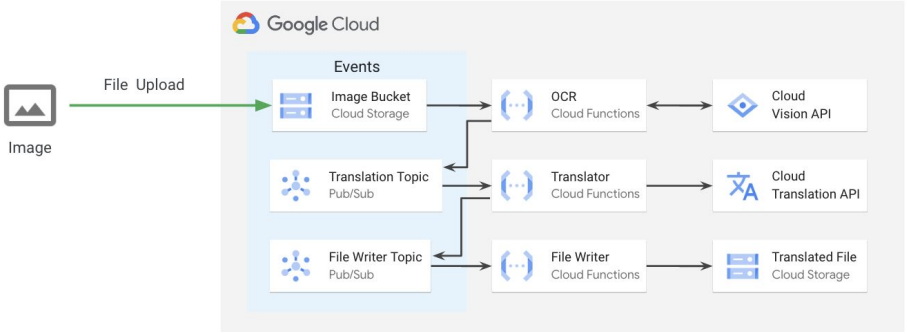
Google Cloud Pub/Sub Triggers:
https://cloud.google.com/functions/docs/calling/pubsub

Cloud Pub/Sub Tutorial:
https://cloud.google.com/functions/docs/tutorials/pubsub

Google Cloud Storage Triggers:
https://cloud.google.com/functions/docs/calling/storage

# Useful when need event-driven, highly scalable microservices

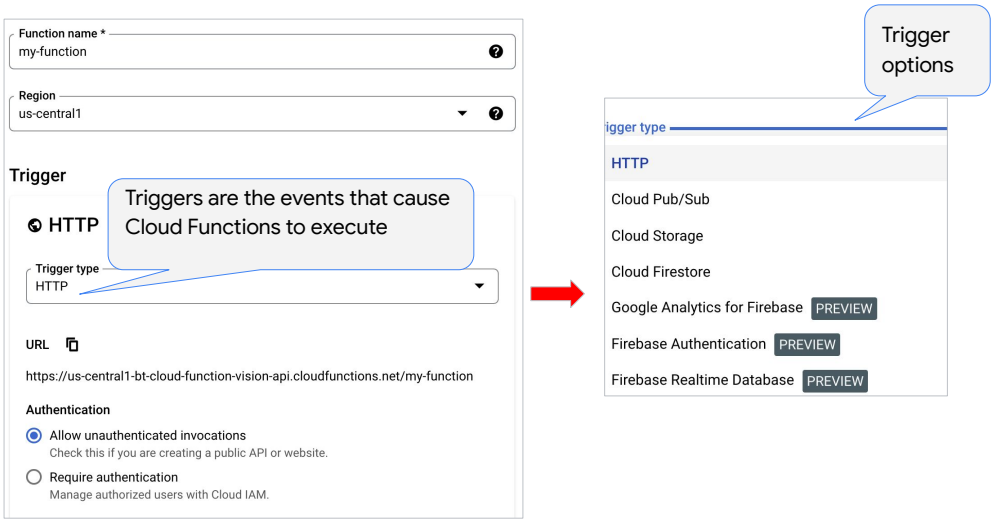
- Can be triggered by changes in a storage bucket, Pub/Sub messages, web requests and other types of events
- Completely managed, scalable, and inexpensive




Tutorial: https://cloud.google.com/functions/docs/tutorials/ocr

# Creating Cloud Functions



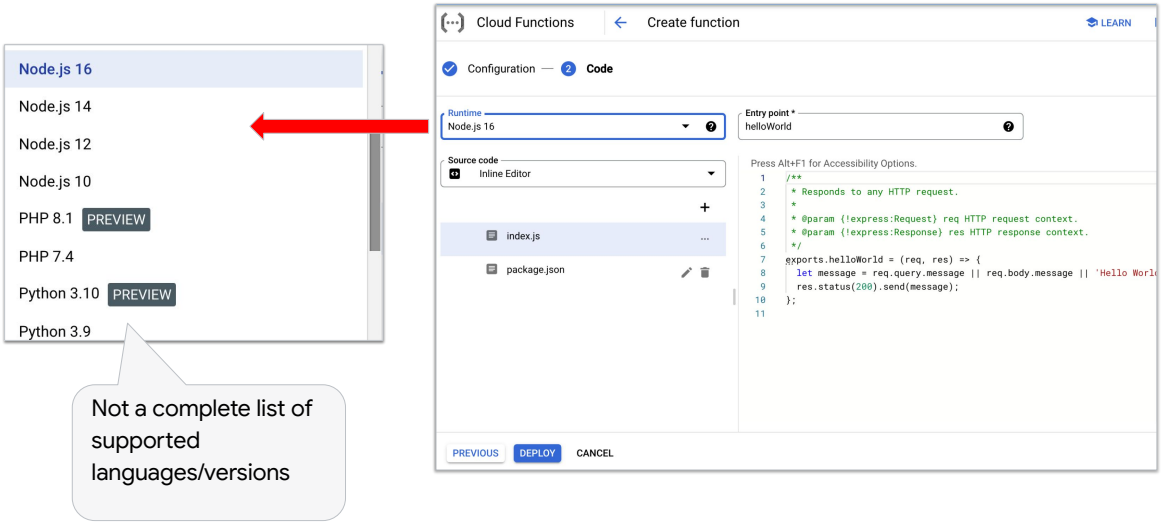





Using maximum instances:
https://cloud.google.com/functions/docs/configuring/max-instances#gcloud

Using minimum instances:
https://cloud.google.com/functions/docs/configuring/min-instances

# Creating Cloud Functions (continued)



# Cloud Function CLI

- Deploy a Cloud Function

```
gcloud functions deploy my-java-function --entry-point
com.example.MyFunction --runtime java11 --trigger-http
--allow-unauthenticated
```



Deploying Cloud Functions:
https://cloud.google.com/functions/docs/deploying

# Exam Guide - Cloud Run and Cloud Functions

3.3 Deploying and implementing Cloud Run and Cloud Functions resources. Tasks include, where applicable:

- 3.3.1 Deploying an application and updating scaling configuration, versions, and traffic splitting
- 3.3.2 Deploying an application that receives Google Cloud events (e.g., Pub/Sub events, Cloud Storage object change notification events)

4.3 Managing Cloud Run resources. Tasks include:

- 4.3.1 Adjusting application traffic-splitting parameters
- 4.3.2 Setting scaling parameters for autoscaling instances
- 4.3.3 ~~Determining whether to run Cloud Run (fully managed) or Cloud~~

Answers depend on which "generation" of Cloud Functions is used

# Cloud Functions 2nd Generation



- Cloud Functions 2nd generation was released in August 2022
  - Features are outlined in the table below

*Exam questions will probably be generic and not specific to Generation 1 or 2

| Feature | Cloud Functions (1st gen) | Cloud Functions (2nd gen) |
|---|---|---|
| Image registry | Container Registry or Artifact Registry | Artifact Registry only |
| Request timeout | Up to 9 minutes | • Up to 60 minutes for HTTP-triggered functions<br>• Up to 9 minutes for event-triggered functions |
| Instance size | Up to 8GB RAM with 2 vCPU | Up to 16GiB RAM with 4 vCPU |
| Concurrency | 1 concurrent request per function instance | Up to 1000 concurrent requests per function instance |
| Traffic splitting | Not supported | Supported |
| Event types | Direct support for events from 7 sources | Support for any event type supported by Eventarc, including 90+ event sources via Cloud Audit Logs |
| CloudEvents | Supported only in Ruby, .NET, and PHP runtimes | Supported in all language runtimes |

*Personal opinion of content developer; Others may disagree

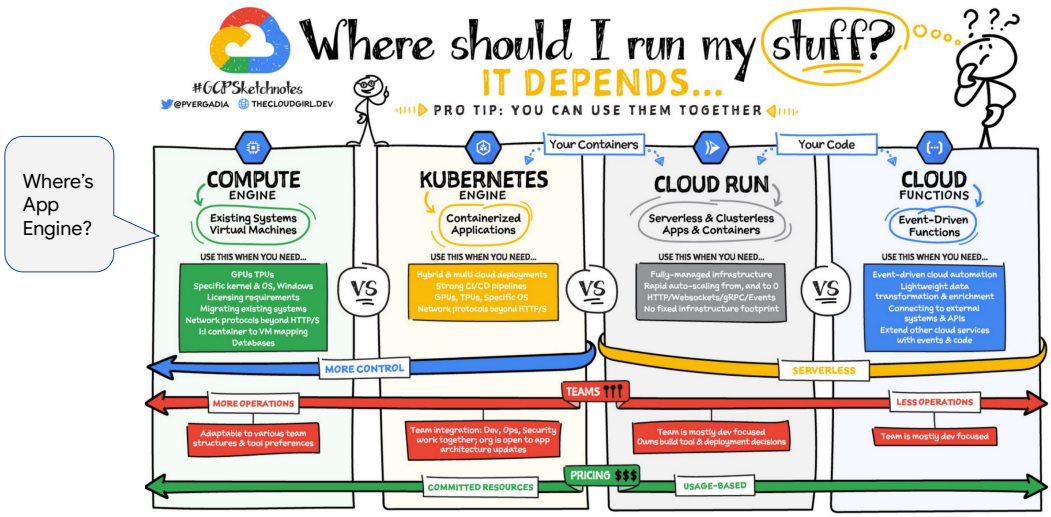


Cloud Functions version comparison
https://cloud.google.com/functions/docs/concepts/version-comparison

Cloud Functions vs. Cloud Run: when to use one over the other
https://cloud.google.com/blog/products/serverless/cloud-run-vs-cloud-functions-for-serverless

# Choosing a Google Cloud compute option

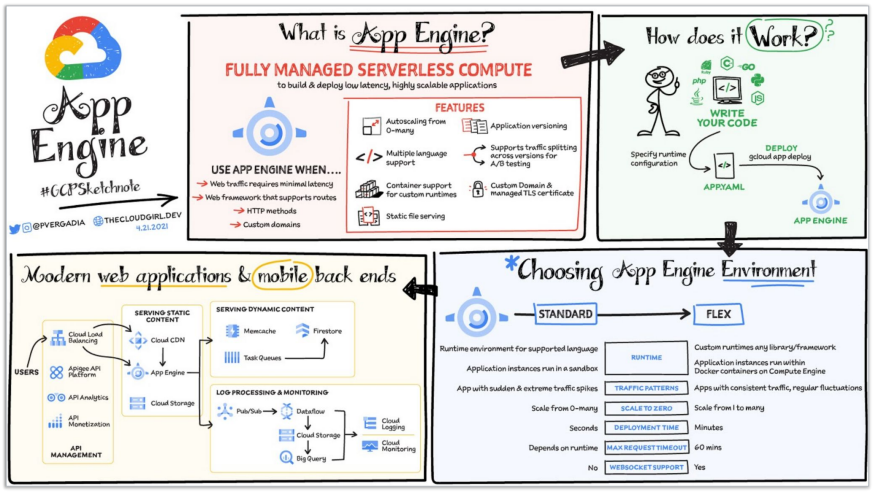







Source:
https://cloud.google.com/blog/topics/developers-practitioners/where-should-i-run-my-stuff-choosing-google-cloud-compute-option

# What is App Engine?



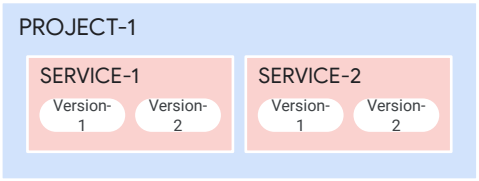

[The ultimate App Engine cheat sheet]



App Engine cheat sheet
https://cloud.google.com/blog/topics/developers-practitioners/ultimate-app-engine-cheat-sheet

## App Engine offers fully managed, serverless compute for low latency, highly scalable applications

- Developers focus on code, Google manages infrastructure
- Supports Node.js, Java, Ruby, C#, Go, Python, or PHP
- Autoscales automatically depending on load
- 2 Types
  - App Engine Standard
    - Limited language support
    - Free tier
  - App Engine Flexible
    - Google Cloud builds a Docker container
    - Runs on Compute Engine - no free tier






App Engine documentation
https://cloud.google.com/appengine/docs

App Engine is discussed for completeness, even though it was not explicitly mentioned in the exam guide

App Engine is a fully managed, serverless application platform supporting the building and deploying of applications. Applications can be scaled seamlessly from zero upward without having to worry about managing the underlying infrastructure. App Engine was designed for microservices. For configuration, each Google Cloud project can contain one App Engine application, and an application has one or more services. Each service can have one or more versions, and each version has one or more instances. App Engine supports traffic splitting so it makes switching between versions and strategies such as canary testing or A/B testing simple. The diagram on the right shows the high-level organization of a Google Cloud project with two services, and each service has two versions. These services are independently deployable and versioned.

# App Engine Standard

- Instances start in milliseconds
  - Can scale to zero instances - no charge when app is not running
  - Free tier of 28 instance hours per day
- Supports certain languages, including Python
- Runs your app in a restrictive sandbox environment
  - Has built-in APIs for tasks, queuing, memory store, etc.

https://cloud.google.com/appengine/docs/standard



App Engine standard
https://cloud.google.com/appengine/docs/standard

App Engine Standard environment runs your code in containers provided by Google.

Container instances can start in milliseconds. If there is no traffic coming to an application, it will turn all the containers off. When it scales to zero instances, you aren't charged anything for the application. If a request comes in, a container starts and handles the requests. If millions of requests start coming in, it will scale quickly to meet the demand. There is a free tier for App Engine standard that allows smaller apps to run without generating a bill.

App Engine Standard supports many languages including Python, Java, PHP, Go, and JavaScript.

# App Engine Flexible

- Supports multiple programming languages*, including:
  - C#, Go, Java, Node.js, PHP, Python, and Ruby
  - Python, Java, Node.js, Go, Ruby, PHP
- Runs Docker containers on Compute Engine VM instances using a container optimized OS
  - Two options
    - Upload code which will be containerized internally by Google before deployment
    - Provide a Dockerfile, along with the associated code, which Google will deploy
- One container per VM
  - Google manages the number of VMs
    - Can specify min/max number of VMs to deploy when scaling
- When additional capacity is needed, **may take > minute to scale** due to VM creation
- **Does not scale to zero - no free tier**

App Engine flexible environment

*Check the documentation for a complete list of supported programming languages



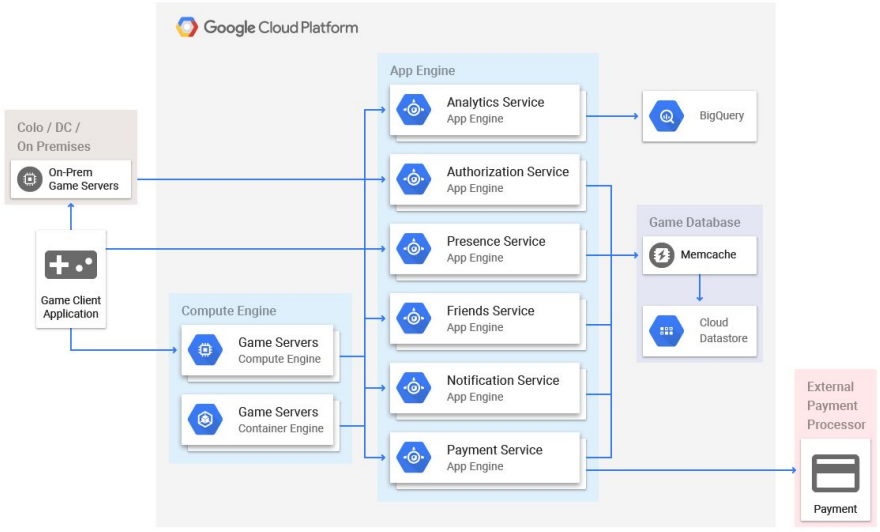
App Engine Flexible
https://cloud.google.com/appengine/docs/flexible/

# Choosing an App Engine environment

- When should you choose Standard vs Flexible?
  - https://cloud.google.com/appengine/docs/the-appengine-environments

# App Engine Example: Gaming Platform Services





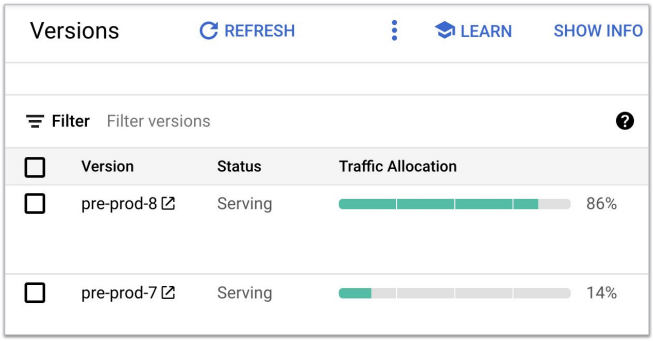



Since game platform services often need to be accessed by many different processes - including client apps, game servers, and websites - using RESTful HTTP endpoints is a very effective pattern. Google App Engine allows you to just write the code for these endpoints without worrying about scaling or downtime.

Game Database: Cloud Firestore in Datastore mode along with a dedicated Memcache can provide a fast, reliable, scalable App Engine native NoSQL database. This database pattern has been proven to scale seamless for games that started out serving thousands and ended up serving millions, such as Pokemon Go

# App Engine supports multiple applications/multiple versions

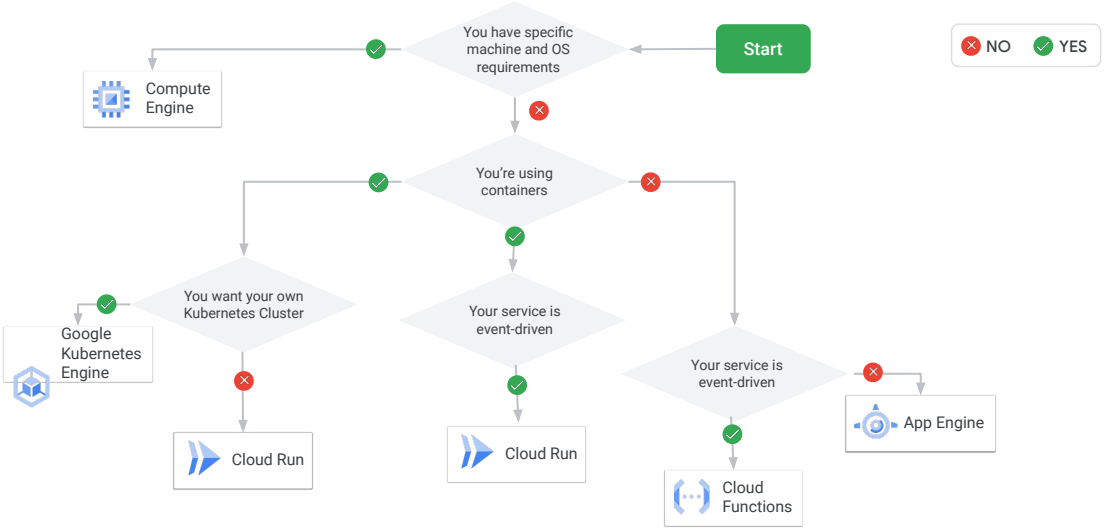
- Each Google Cloud project can contain 1 App Engine application.
- An application has 1 or more services.
- Each service has 1 or more versions.
- Versions have 1 or more instances.
- Automatic traffic splitting for switching versions






App Engine is a fully managed, serverless application platform supporting the building and deploying of applications. Applications can be scaled seamlessly from zero upward without having to worry about managing the underlying infrastructure. App Engine was designed for microservices. For configuration, each Google Cloud project can contain one App Engine application, and an application has one or more services. Each service can have one or more versions, and each version has one or more instances. App Engine supports traffic splitting so it makes switching between versions and strategies such as canary testing or A/B testing simple. The diagram on the right shows the high-level organization of a Google Cloud project with two services, and each service has two versions. These services are independently deployable and versioned.

# Choosing a Google Cloud deployment platform






Here is a high-level overview of how you could decide on the most suitable platform for your application.

First, ask yourself whether you have specific machine and OS requirements. If you do, then Compute Engine is the platform of choice.

If you have no specific machine or operating system requirements, then the next question to ask is whether you are using containers. If you are, then you should consider Google Kubernetes Engine or Cloud Run, depending on whether you want to configure your own Kubernetes cluster.

If you are not using containers, then you want to consider Cloud Functions if your service is event-driven and App Engine if it's not.

# Exam Guide - Install CLI, Calculator, Marketplace, IaC

Cloud SDK

Cloud Marketplace

Cloud Deployment Manager

**1.3 Installing and configuring the command line interface (CLI), specifically the Cloud SDK (e.g., setting the default project)**

**2.1 Planning and estimating Google Cloud product using the Pricing Calculator**

**3.6 Deploying a solution using Cloud Marketplace. Tasks include:**

3.6.1 Browsing the Cloud Marketplace catalog and viewing solution details

3.6.2 Deploying a Cloud Marketplace solution

**3.7 Implementing resources via infrastructure as code. Tasks include:**

3.7.1 Building infrastructure via Cloud Foundation Toolkit templates and implementing best practices

3.7.2 Installing and configuring Config Connector in Google Kubernetes Engine to create, update, delete, and secure resources

# Exam Guide - Install CLI, Calculator, Marketplace, IaC

1.3 Installing and configuring the command line interface (CLI), specifically the Cloud SDK (e.g., setting the default project)

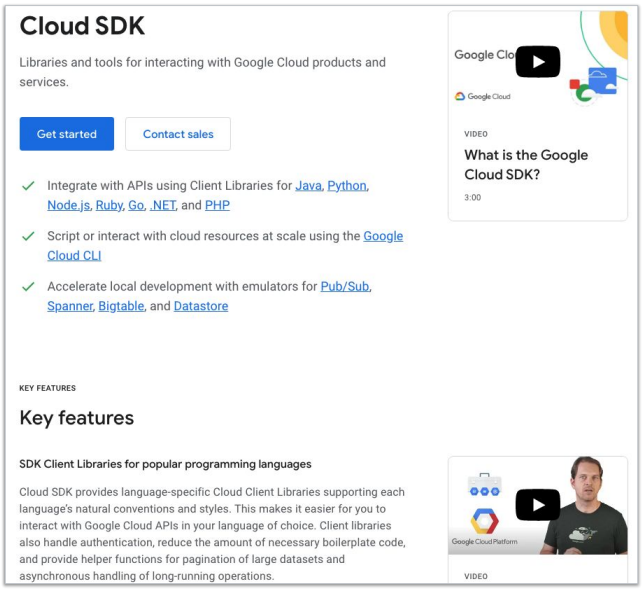
2.1 Planning and estimating Google Cloud product using the Pricing Calculator

3.6 Deploying a solution using Cloud Marketplace. Tasks include:
- 3.6.1 Browsing the Cloud Marketplace catalog and viewing solution details
- 3.6.2 Deploying a Cloud Marketplace solution

3.7 Implementing resources via infrastructure as code. Tasks include:
- 3.7.1 Building infrastructure via Cloud Foundation Toolkit templates and implementing best practices
- 3.7.2 Installing and configuring Config Connector in Google Kubernetes Engine to create, update, delete, and secure resources

# Google Cloud SDK

See: https://cloud.google.com/sdk/






Google Cloud SDK:
https://cloud.google.com/sdk/

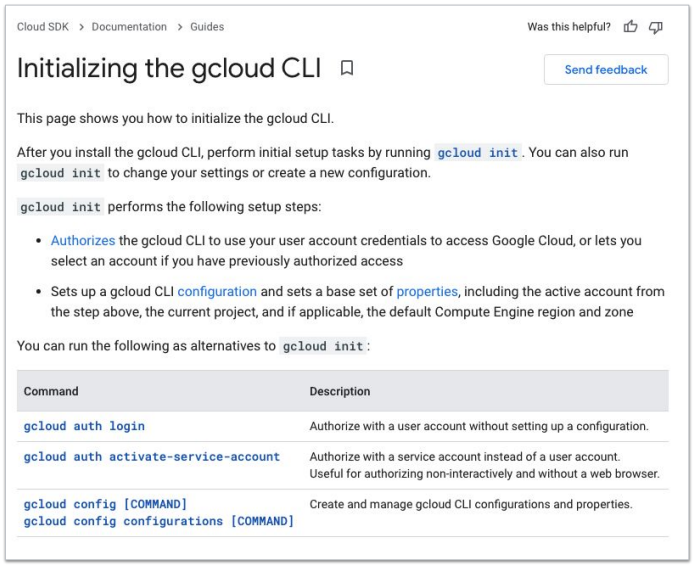
SDK Installation and quick start:
https://cloud.google.com/sdk/docs/install-sdk

To get started using the SDK you need to install it. Go to cloud.google.com/sdk to find information and instructions for installing it on your preferred operating system.

- Includes command-line tools for Google CLoud products and services.
  - gcloud, gsutil (Cloud Storage), bq (BigQuery)
- Access via the Cloud Shell button in the Cloud Console
- Can also be installed on local machines.
- Is also available as a Docker image.

# Initializing the gcloud CLI

See: https://cloud.google.com/sdk/docs/initializing



Cloud SDK > Documentation > Guides

Was this helpful?

## Initializing the gcloud CLI

Send feedback

This page shows you how to initialize the gcloud CLI.

After you install the gcloud CLI, perform initial setup tasks by running `gcloud init`. You can also run `gcloud init` to change your settings or create a new configuration.

`gcloud init` performs the following setup steps:

- Authorizes the gcloud CLI to use your user account credentials to access Google Cloud, or lets you select an account if you have previously authorized access
- Sets up a gcloud CLI configuration and sets a base set of properties, including the active account from the step above, the current project, and if applicable, the default Compute Engine region and zone

You can run the following as alternatives to `gcloud init`:

| Command | Description |
|---|---|
| `gcloud auth login` | Authorize with a user account without setting up a configuration. |
| `gcloud auth activate-service-account` | Authorize with a service account instead of a user account. Useful for authorizing non-interactively and without a web browser. |
| `gcloud config [COMMAND]`<br>`gcloud config configurations [COMMAND]` | Create and manage gcloud CLI configurations and properties. |



Initializing the gcloud CLI:
https://cloud.google.com/sdk/docs/initializing

gcloud command to set configuration:
https://cloud.google.com/sdk/gcloud/reference/config/set

gcloud tool guide:
https://cloud.google.com/sdk/gcloud/

After the CLI is installed, you need to do an initial setup.  That includes things such as setting a default region and zone.

# Exam Guide - Install CLI, Calculator, Marketplace, IaC

1.3 Installing and configuring the command line interface (CLI), specifically the Cloud SDK (e.g., setting the default project)

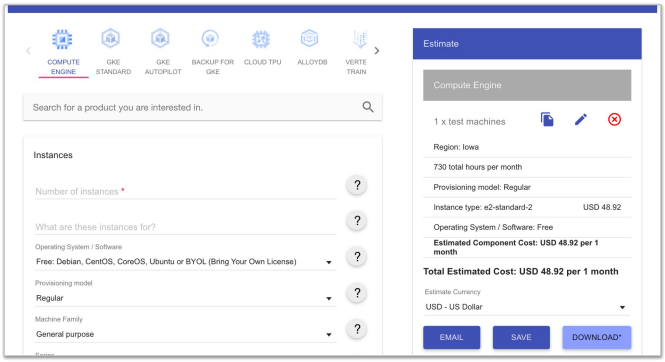
2.1 Planning and estimating Google Cloud product using the Pricing Calculator

3.6 Deploying a solution using Cloud Marketplace. Tasks include:
- 3.6.1 Browsing the Cloud Marketplace catalog and viewing solution details
- 3.6.2 Deploying a Cloud Marketplace solution

3.7 Implementing resources via infrastructure as code. Tasks include:
- 3.7.1 Building infrastructure via Cloud Foundation Toolkit templates and implementing best practices
- 3.7.2 Installing and configuring Config Connector in Google Kubernetes Engine to create, update, delete, and secure resources

# Use the Google Cloud Pricing Calculator to estimate costs

- Create cost estimates based on forecasting and capacity planning.
- The parameters entered will vary according to the service, e.g.,
  - Compute Engine - machine type, operating system, usage/day, disk size, etc
  - Cloud Storage - Location, storage class, storage amount, ingress and egress estimates
- Can save and email estimates for later use, e.g., presentations




https://cloud.google.com/products/calculator



Pricing calculator:
https://cloud.google.com/products/calculator/

The pricing calculator is the go-to resource for gaining cost estimates. Remember that the costs are just an estimate, and actual cost may be higher or lower. The estimates by default use the timeframe of one month. If any inputs vary from this, they will state this. For example, Firestore document operations read, write, and delete are asked for on a per day basis.

# Exam Guide - Install CLI, Calculator, Marketplace, IaC

1.3 Installing and configuring the command line interface (CLI), specifically the Cloud SDK (e.g., setting the default project)

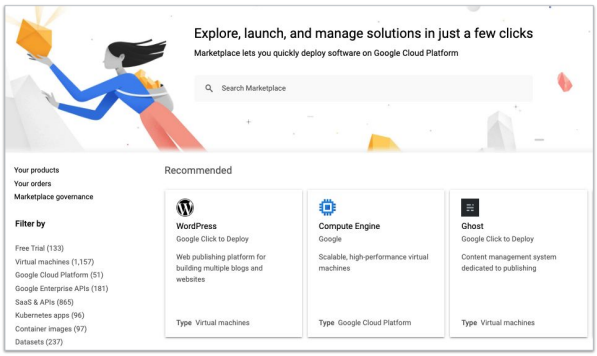
2.1 Planning and estimating Google Cloud product using the Pricing Calculator

3.6 Deploying a solution using Cloud Marketplace. Tasks include:
- 3.6.1 Browsing the Cloud Marketplace catalog and viewing solution details
- 3.6.2 Deploying a Cloud Marketplace solution

3.7 Implementing resources via infrastructure as code. Tasks include:
- 3.7.1 Building infrastructure via Cloud Foundation Toolkit templates and implementing best practices
- 3.7.2 Installing and configuring Config Connector in Google Kubernetes Engine to create, update, delete, and secure resources

# Deploying a Cloud Marketplace solution

- Provides access to approved deployments for common applications
- Some deployments use Deployment Manager
  - Others use Kubernetes
- Some are open source
  - Others require a license






GKE: Deploying an application from Cloud Marketplace:
https://cloud.google.com/kubernetes-engine/docs/how-to/deploying-marketplace-app

Suggested lab - Provision Services with Google Cloud Marketplace
https://partner.cloudskillsboost.google/catalog_lab/339

# Exam Guide - Install CLI, Calculator, Marketplace, IaC

1.3 Installing and configuring the command line interface (CLI), specifically the Cloud SDK (e.g., setting the default project)

2.1 Planning and estimating Google Cloud product using the Pricing Calculator

3.6 Deploying a solution using Cloud Marketplace. Tasks include:
- 3.6.1 Browsing the Cloud Marketplace catalog and viewing solution details
- 3.6.2 Deploying a Cloud Marketplace solution

3.7 Implementing resources via infrastructure as code. Tasks include:
- 3.7.1 Building infrastructure via Cloud Foundation Toolkit templates and implementing best practices
- 3.7.2 Installing and configuring Config Connector in Google Kubernetes Engine to create, update, delete, and secure resources

## Moving to the cloud requires a mindset change

**On-Premises**

- Buy machines.
- Keep machines running for years.
- Prefer fewer big machines.
- Machines are capital expenditures.

**Cloud**

- Rent machines.
- Turn machines off as soon as possible.
- Prefer lots of small machines.
- Machines are monthly expenses.



The on demand, pay-per-use model of cloud computing is a different model to traditional on-premises infrastructure provisioning. Resources can be allocated to best meet demand in a timely manner, and the cloud supports experimentation and innovation by providing immediate access to an ever-increasing range of services.

## Treat infrastructure as disposable in the cloud

- Don't fix broken machines.
- Don't install patches.
- Don't upgrade machines.
- If you need to fix a machine, delete it and re-create a new one.

- Many tools exist for automating infrastructure creation
  - Terraform
  - Deployment Manager on Google Cloud
  - CloudFormation on AWS
  - Resource Manager on Azure



The key term is infrastructure as code (IaC). The provisioning, configuration, and deployment activities should all be automated.

Having the process automated minimizes risks, eliminates manual mistakes, and supports repeatable deployments and scale and speed. Deploying one or one hundred machines is the same effort.

Costs can be reduced by provisioning ephemeral environments, such as test environments that replicate the production environment.

## Infrastructure as code (IaC) allows quick provisioning and removing of infrastructures

- Build an infrastructure when needed.
- Destroy the infrastructure when not in use.
- Create identical infrastructures for dev, test, and prod
  - Use templates to create different types of resources per group
- Can be part of a CI/CD pipeline.
- Templates are the building blocks for disaster recovery procedures.



In essence, infrastructure as code allows for the quick provisioning and removing of infrastructures.

The on-demand provisioning of a deployment is extremely powerful. This can be integrated into a continuous integration pipeline that smoothes the path to continuous deployment.

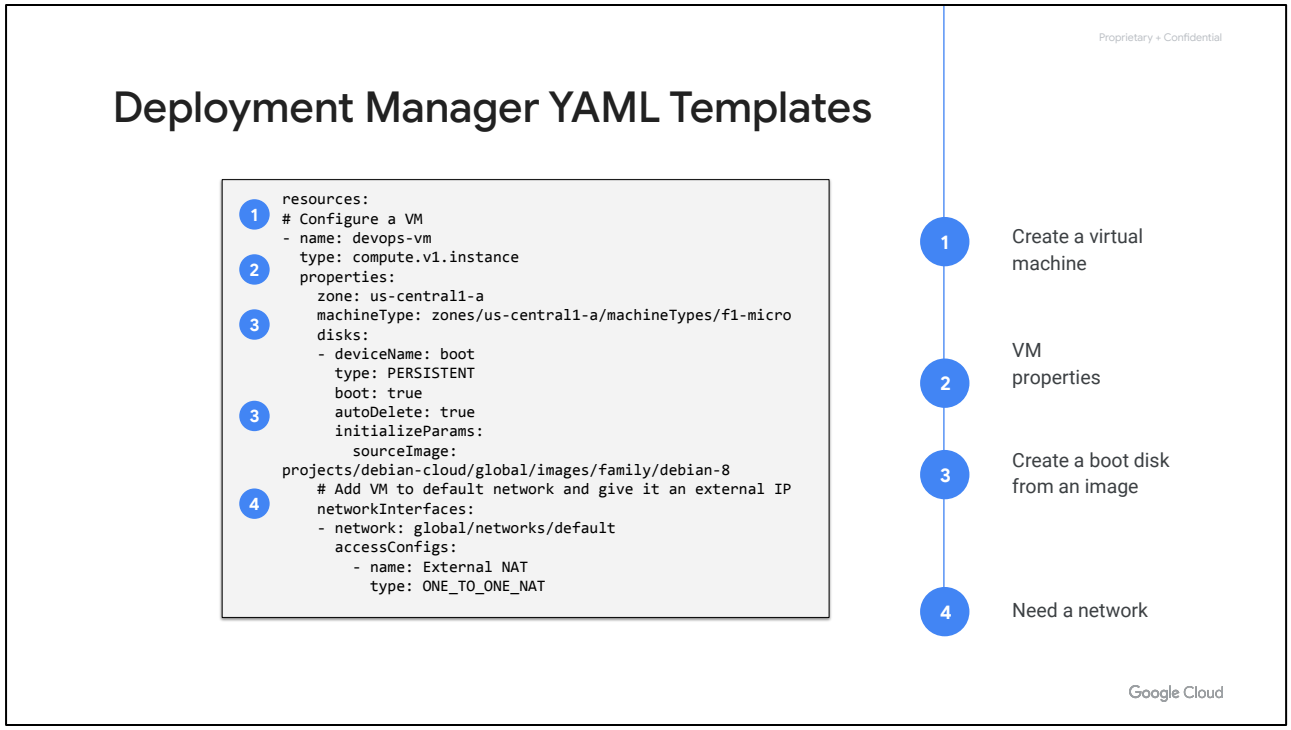
Automated infrastructure provisioning means that the infrastructure can be provisioned on demand, and the deployment complexity is managed in code. This provides the flexibility to change infrastructure as requirements change. And all the changes are in one place. Infrastructure for environments such as development and test can now easily replicate production and can be deleted immediately when not in use. All because of infrastructure as code.

Several tools can be used for IaC. Google Cloud supports Terraform, where deployments are described in a file known as a configuration. This details all the resources that should be provisioned. Configurations can be modularized using templates, which allows the abstraction of resources into reusable components across deployments.

In addition to Terraform, Google Cloud also provides support for other IaC tools, including:

- Deployment Manager
- Chef

- Puppet
- Ansible
- Packer




Deployment Manager is Google Cloud's IaC tool
https://cloud.google.com/deployment-manager/docs

Shown here is a simple Deployment Manager template written in YAML. The root element at the top is resources.

Resources is a collection. Notice the minus sign before the name element. In YAML, the minus sign denotes one item in a collection. In this case, we are creating a VM named devops-vm.

The VM has a collection of properties, one of which is a collection of disks. Lastly, at the bottom we are assigning the machine to a network.

## Managing deployments with `gcloud`

- Once the template is built, use `gcloud` to deploy it
  - Simple creation of infrastructure
  - Easy changes with update command
  - Delete will remove everything created in the reverse order
- Examples:
  - `gcloud deployment-manager deployments **create** devops-deployment --config deployment-manager-config.yaml`
  - `gcloud deployment-manager deployments **list**`
  - `cloud deployment-manager deployments **update** example-deployment --config deployment-manager-config.yaml --preview`
  - `gcloud deployment-manager deployments **delete** devops-deployment`



The gcloud commands used with Deployment Manager are pretty straightforward. Like all gcloud commands you specify the service, collection verb, name, and parameters.

So all the commands begin with gcloud deployment-manager deployments. The verbs are create, list, update, and delete. Lastly, specify the deployment name and the parameters.

When running an update, only resources that have changed in the template will be updated. When running a delete command, it is smart enough to delete resources in the reverse order in which they were created. Thus, you don't get an error when deleting a resource that is used by another resource. As an example, an Instance Group cannot be deleted if a Load Balancer back end is using it. Deployment Manager handles those types of issues for you.

# Terraform susports multi-cloud environments



- Pre-installed in Cloud Shell
- Supports both a native syntax named HCL plus JSON-compatible syntax as shown here

```
1  resource "google_compute_instance" "default" {
     name         = "flask-vm"
     machine_type = "f1-micro"
     zone         = "us-west1-a"
2    tags         = ["ssh"]

     boot_disk {
       initialize_params {
         image = "debian-cloud/debian-9"
       }
     }

3    # Install Flask
     metadata_startup_script = "sudo apt-get update; sudo
apt-get install -yq build-essential python-pip rsync; pip
install flask"

4    network_interface {
       network = "default"

       access_config {
         // Ephemeral public IP
       }}}
```

1. Create a virtual machine
2. Create a boot disk from an image
3. Run a script at startup
4. Network interface to default VPC & external IP



Terraform
https://www.terraform.io/

# Managing Terraform deployments

- Once the template is built, use terraform commands to make changes
  - `terraform init` : get plugins if needed
  - `terraform plan` : checks for errors in the terraform file
  - `terraform apply` : applies the file and creates the resources
  - `terraform delete` : deletes the resources

# Cloud Foundation Toolkit

- Ready made IaC templates which reflect best practices, in both
  - Deployment Manager
  - Terraform
- Can be used off-the-shelf to quickly build a repeatable enterprise-ready foundation in Google Cloud
  - Can easily update the foundation as needs change

Some of the example templates

| Resource | Description |
|---|---|
| Autoscaler | Create a Compute Engine autoscaler. |
| Backend Service | Create a global or regional backend service. |
| Bastion Host | Create a bastion host, which you can use to access other servers in the same network. |
| BigQuery | Create a BigQuery dataset and table. |
| Cloud Functions | Create a Cloud Functions function. |
| Cloud Router | Create a Cloud Router. |
| Cloud Spanner | Create a Spanner instance and database. |
| Cloud SQL | Create a Cloud SQL instance with databases and users. |
| Cloud Tasks | Create a Cloud Tasks task and task queue. |
| Dataproc | Create a Dataproc cluster. |
| Cloud DNS managed zone | Create a managed zone in Cloud DNS. |
| Cloud DNS records | Create Cloud DNS records using recordsets. |
| External Load Balancer | Create an HTTP(S), SSL Proxy, or TCP Proxy external load balancer. |
| Firewall rules | Create firewall rules for your network. |



Cloud Foundation Toolkit
https://cloud.google.com/foundation-toolkit

Example templates from the Cloud Foundation Toolkit:
https://cloud.google.com/deployment-manager/docs/reference/cloud-foundation-toolkit

Rapid cloud foundation buildout and workload deployment using Terraform:
https://cloud.google.com/blog/products/devops-sre/using-the-cloud-foundation-toolkit-with-terraform

Terraform with Google Cloud: https://cloud.google.com/docs/terraform

# Exam Guide - Install CLI, Calculator, Marketplace, IaC

1.3 Installing and configuring the command line interface (CLI), specifically the Cloud SDK (e.g., setting the default project)

2.1 Planning and estimating Google Cloud product using the Pricing Calculator

3.6 Deploying a solution using Cloud Marketplace. Tasks include:
- 3.6.1 Browsing the Cloud Marketplace catalog and viewing solution details
- 3.6.2 Deploying a Cloud Marketplace solution

3.7 Implementing resources via infrastructure as code. Tasks include:
- 3.7.1 Building infrastructure via Cloud Foundation Toolkit templates and implementing best practices
- 3.7.2 Installing and configuring Config Connector in Google Kubernetes Engine to create, update, delete, and secure resources

# Config Connector

- Part of the Anthos toolset
- Lets you manage more than 120 Google Cloud resources the same way you manage other Kubernetes resources
  - Use YAML files

YAML creating a Pub/Sub topic

```
apiVersion: pubsub.cnrm.cloud.google.com/v1beta1
kind: PubSubTopic
metadata:
  annotations:
    cnrm.cloud.google.com/project-id: my_project_id
  labels:
    environment:production
  name: my_pub_sub_topic
```

To deploy and create the topic

```
kubectl apply -f pubsub-topic.yaml
```



Config Connector overview:
https://cloud.google.com/config-connector/docs/overview

YouTube video:
https://www.youtube.com/watch?v=3lAOr2XdAh4

Choosing an installation type:
https://cloud.google.com/config-connector/docs/concepts/installation-types

Getting Started w/ config connector:
https://cloud.google.com/config-connector/docs/how-to/getting-started

# Exam Guide - Install CLI, Calculator, Marketplace, IaC

1.3 Installing and configuring the command line interface (CLI), specifically the Cloud SDK (e.g., setting the default project)

2.1 Planning and estimating Google Cloud product using the Pricing Calculator

3.6 Deploying a solution using Cloud Marketplace. Tasks include:
- 3.6.1 Browsing the Cloud Marketplace catalog and viewing solution details
- 3.6.2 Deploying a Cloud Marketplace solution

3.7 Implementing resources via infrastructure as code. Tasks include:
- 3.7.1 Building infrastructure via Cloud Foundation Toolkit templates and implementing best practices
- 3.7.2 Installing and configuring Config Connector in Google Kubernetes Engine to create, update, delete, and secure resources


Google Cloud