Google Cloud

Partner Certification Academy

Professional Cloud Architect

Google Cloud

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!

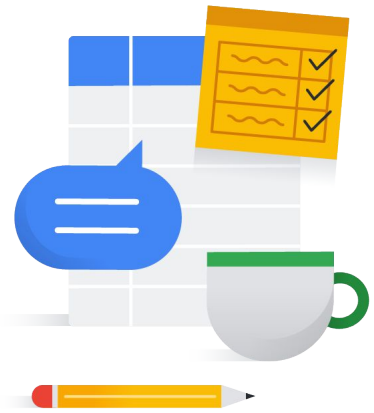
Google Cloud

Session logistics

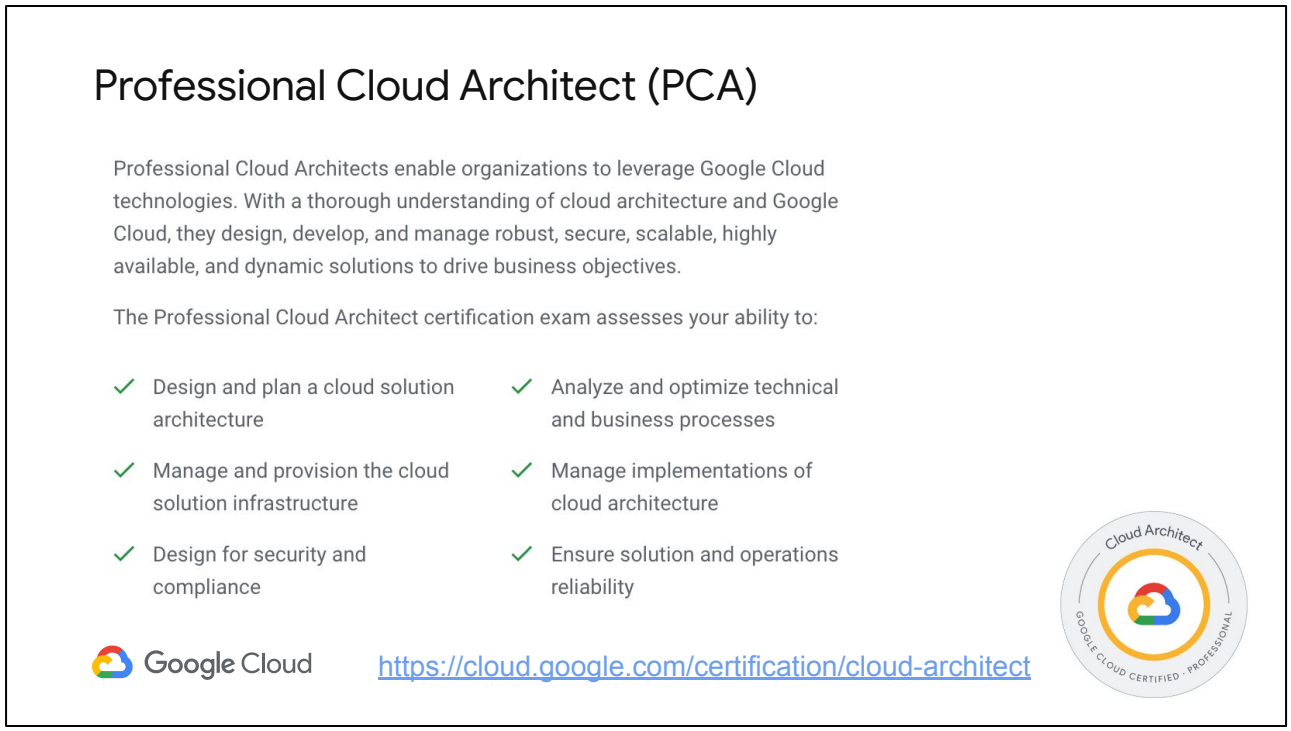
- When you have a question, please:
  - Click the Raise hand button in Google Meet.
  - Or add your question to the Q&A section of Google Meet.
  - Please note that answers may be deferred until the end of the session.
- These slides are available in the Student Lecture section of your Qwiklabs classroom.
- 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



Google Cloud



What is a PCA supposed to know/do?

Professional Cloud Architect Certification Page
https://cloud.google.com/certification/cloud-architect

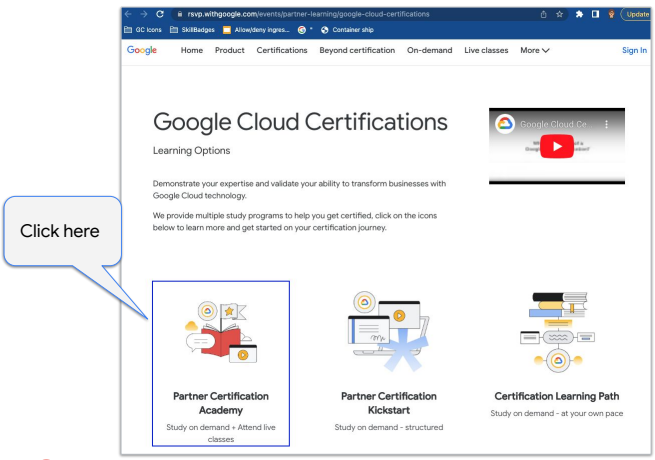
Professional Cloud Architect Exam Guide
https://cloud.google.com/certification/guides/professional-cloud-architect/

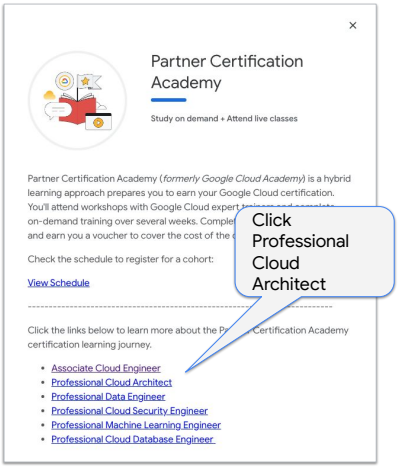
Professional Cloud Architect Sample Questions
https://docs.google.com/forms/d/e/1FAIpQLSdvf8Xq6m0kvyIoysdr8WZYCG32WHENStftiHTSdtW4ad2-0w/viewform

Google Cloud Adoption Framework + Whitepaper
https://cloud.google.com/adoption-framework
https://services.google.com/fh/files/misc/google_cloud_adoption_framework_whitepaper.pdf

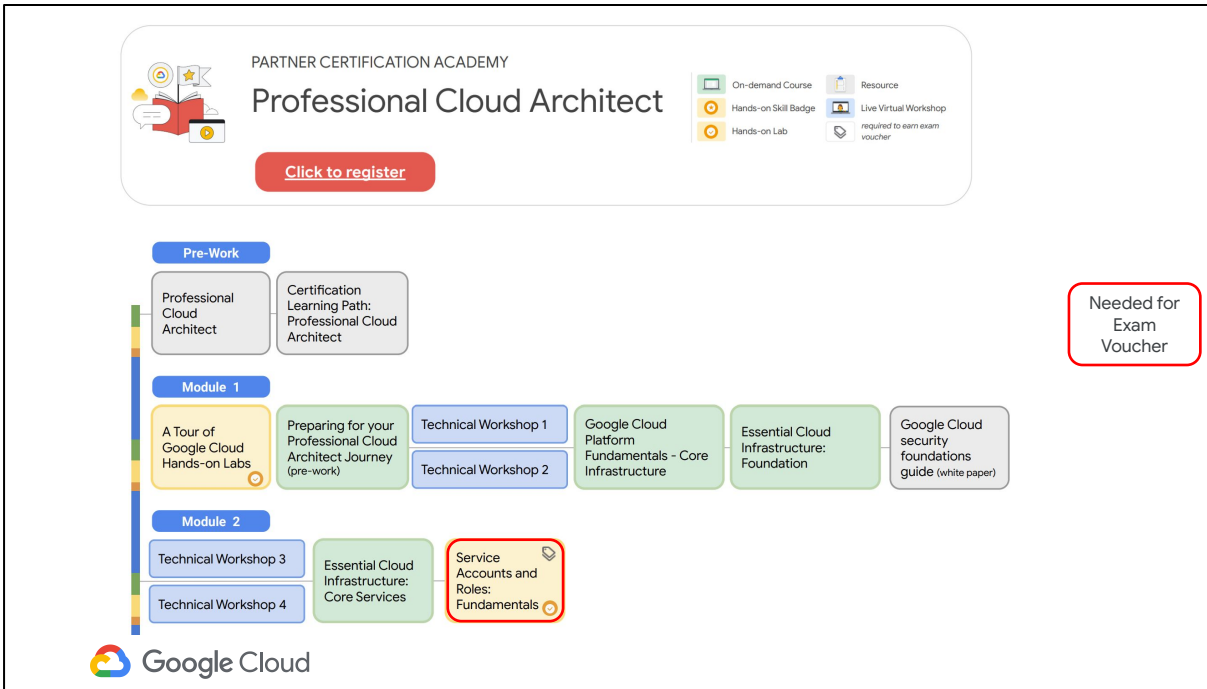
Learning Path - Partner Certification Academy Website

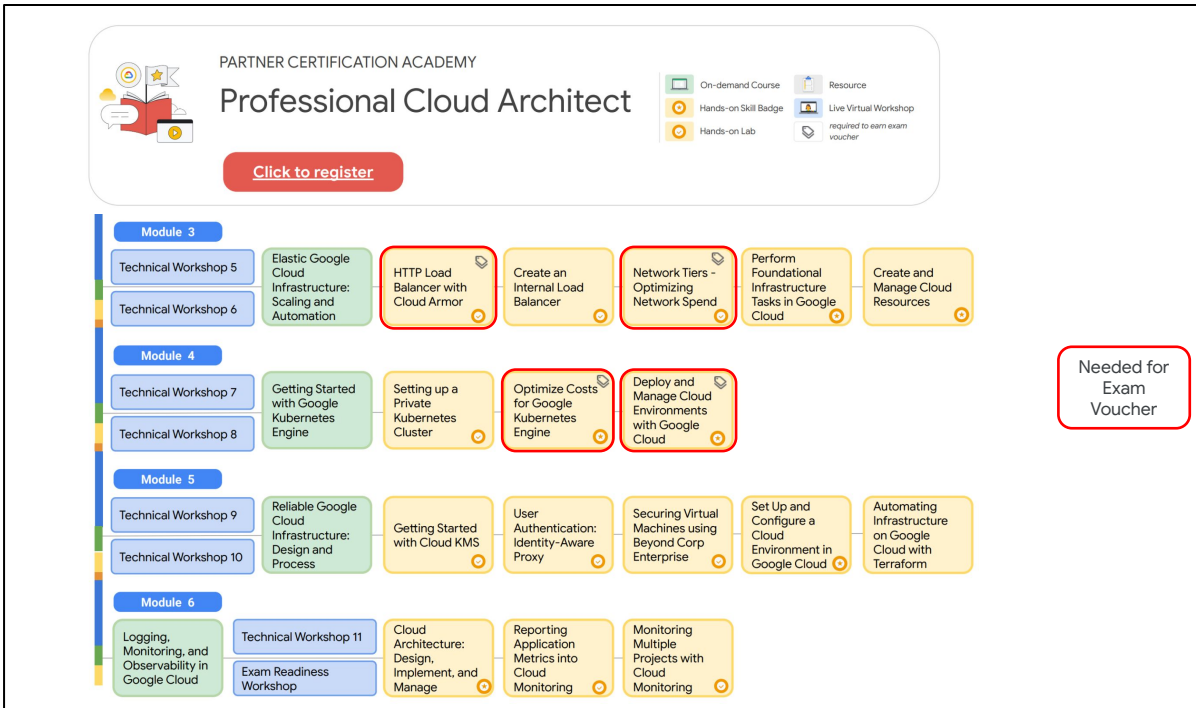
Go to: https://rsvp.withgoogle.com/events/partner-learning/google-cloud-certifications





Google Cloud


PARTNER CERTIFICATION ACADEMY
Professional Cloud Architect
Click to register
On-demand Course
Hands-on Skill Badge
Hands-on Lab
Resource
Live Virtual Workshop
required to earn exam voucher
Pre-Work
Professional Cloud Architect
Certification Learning Path: Professional Cloud Architect
Module 1
A Tour of Google Cloud Hands-on Labs
Preparing for your Professional Cloud Architect Journey (pre-work)
Technical Workshop 1
Technical Workshop 2
Google Cloud Platform Fundamentals - Core Infrastructure
Essential Cloud Infrastructure: Foundation
Google Cloud security foundations guide (white paper)
Module 2
Technical Workshop 3
Technical Workshop 4
Essential Cloud Infrastructure: Core Services
Service Accounts and Roles: Fundamentals
Needed for Exam Voucher
Google Cloud

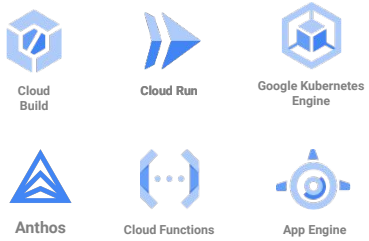

PARTNER CERTIFICATION ACADEMY
Professional Cloud Architect
On-demand Course
Hands-on Skill Badge
Hands-on Lab
Resource
Live Virtual Workshop
required to earn exam voucher
Click to register
Module 3
Technical Workshop 5
Technical Workshop 6
Elastic Google Cloud Infrastructure: Scaling and Automation
HTTP Load Balancer with Cloud Armor
Create an Internal Load Balancer
Network Tiers - Optimizing Network Spend
Perform Foundational Infrastructure Tasks in Google Cloud
Create and Manage Cloud Resources
Module 4
Technical Workshop 7
Technical Workshop 8
Getting Started with Google Kubernetes Engine
Setting up a Private Kubernetes Cluster
Optimize Costs for Google Kubernetes Engine
Deploy and Manage Cloud Environments with Google Cloud
Module 5
Technical Workshop 9
Technical Workshop 10
Reliable Google Cloud Infrastructure: Design and Process
Getting Started with Cloud KMS
User Authentication: Identity-Aware Proxy
Securing Virtual Machines using Beyond Corp Enterprise
Set Up and Configure a Cloud Environment in Google Cloud
Automating Infrastructure on Google Cloud with Terraform
Module 6
Logging, Monitoring, and Observability in Google Cloud
Technical Workshop 11
Exam Readiness Workshop
Cloud Architecture: Design, Implement, and Manage
Reporting Application Metrics into Cloud Monitoring
Monitoring Multiple Projects with Cloud Monitoring
Needed for Exam Voucher

Key concepts in the on-demand content

Google Cloud

Topics in this module

- Containers
- Cloud Build
- Cloud Run
- Google Kubernetes Engine
- Anthos
- Cloud Functions
- App Engine

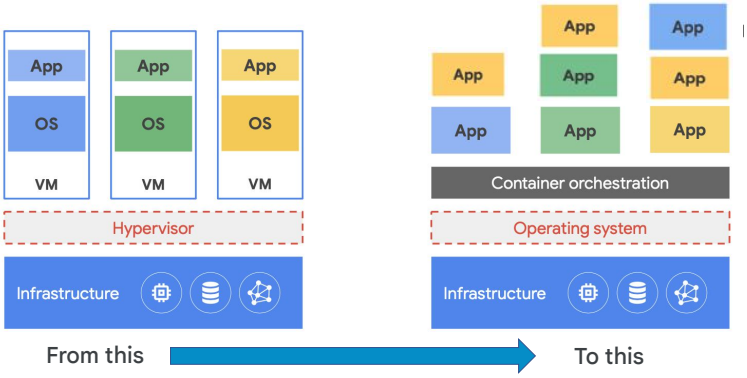


Google Cloud

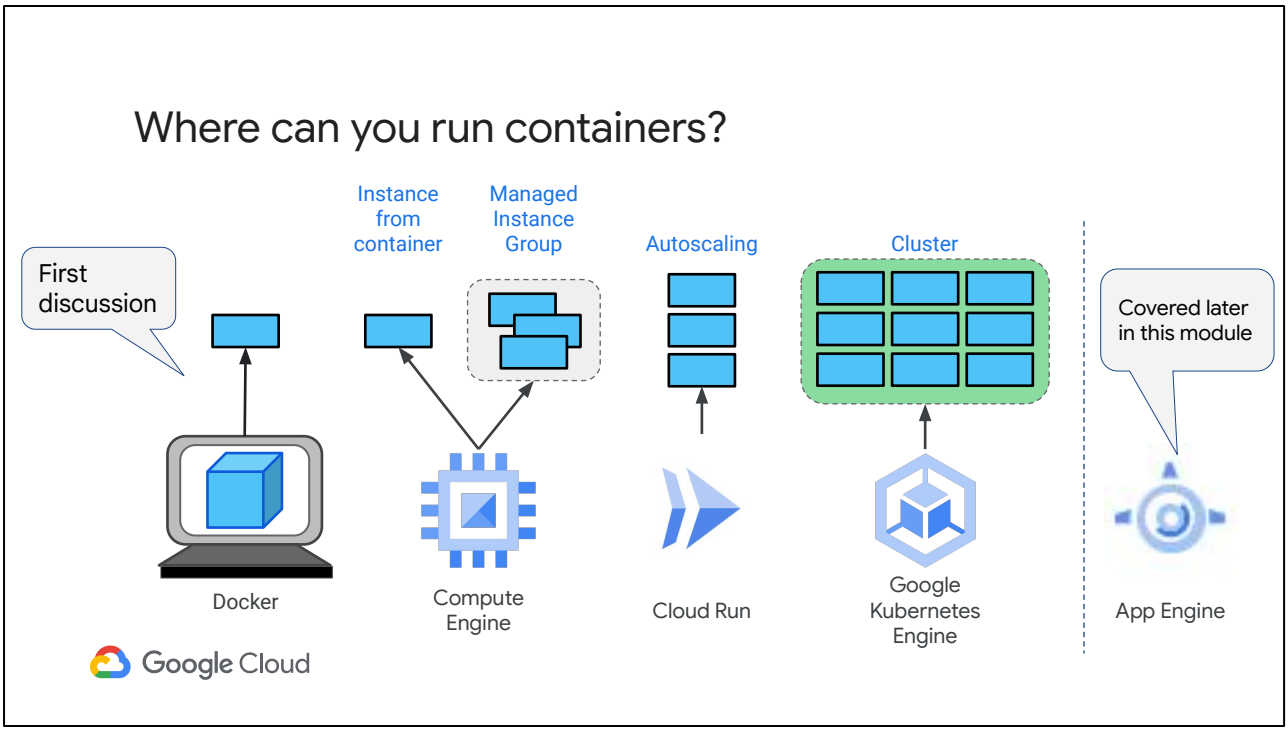
Containers are an approach to app isolation

Containers typically deployed on VMs

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



Google Cloud



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.

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

Google Cloud

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?

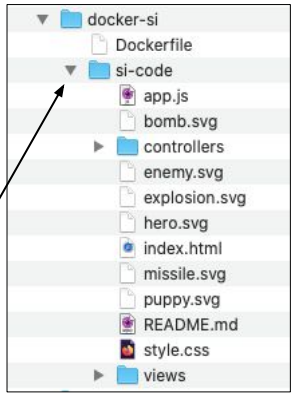
Create a `Dockerfile` which specifies such things as:

- 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
```



Google Cloud

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
```

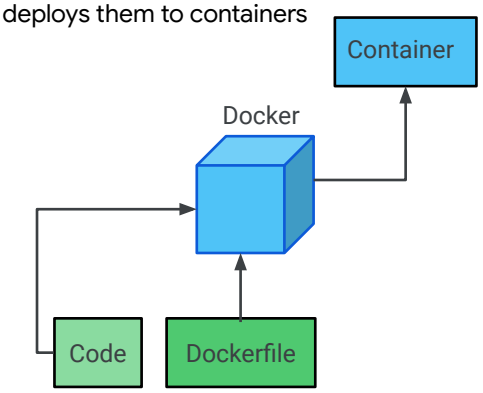
Location in the Artifact Registry

Manually pushing the image

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

Managing deployments

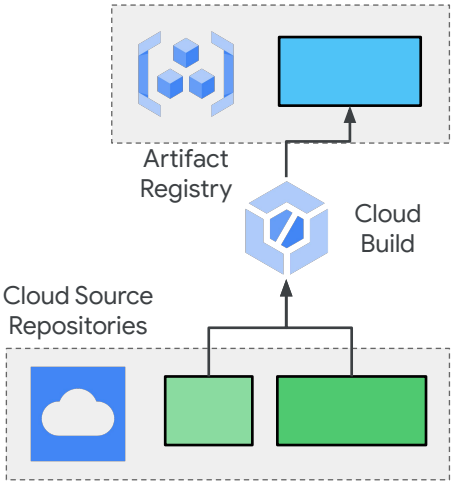
Docker packages apps into images & deploys them to containers



When local testing is complete, can manually push to Artifact Registry:

Google Cloud

Enterprise and continuous deployment



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

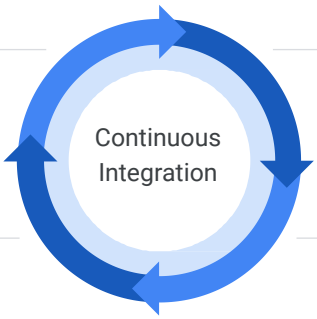
If you had a hundred developers sharing source files, you would need a system for managing them, for tracking them, versioning them, and enforcing a check-in, review, and approval process. They typically push source code to a shared code repository, such as the Google Cloud Source Repositories, as shown 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.

Google provides the components required for a continuous integration pipeline

Cloud Source Repositories
Developers push to a central repository when they want a build to occur.

Artifact/Container Registry
Store your Docker images or deployment packages in a central location for deployment.



Cloud Build
Build system executes the steps required to make a deployment package or Docker image.

Build triggers
Watches for changes in the Git repo and starts the build.

Google Cloud

Cloud Build Serverless CI/CD platform

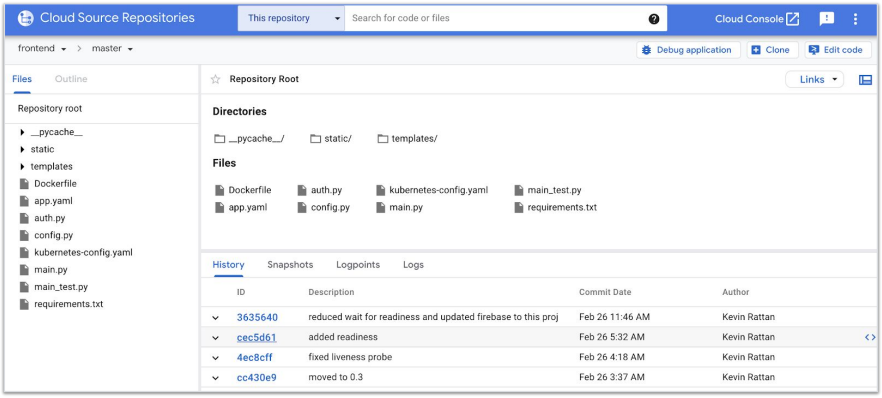
Cloud Build documentation
https://cloud.google.com/build/docs

Cloud Build - Create a build configuration file (lots of good sections to review - look at the different topics on the left)
https://cloud.google.com/build/docs/configuring-builds/create-basic-configuration

Cloud Build Youtube video
https://www.youtube.com/watch?v=2TZXSnCTd7E&list=PLTWE_Imu2InBzuPmOcgAYP7U80a87cpJd

Cloud Source Repositories: managed Git repositories

Control access to your repos using IAM within your Google Cloud projects.



Google Cloud

https://cloud.google.com/source-repositories

Cloud Source Repositories
https://cloud.google.com/source-repositories

You can use Cloud IAM to add team members to your project and to grant them permissions to create, view, and update repositories.

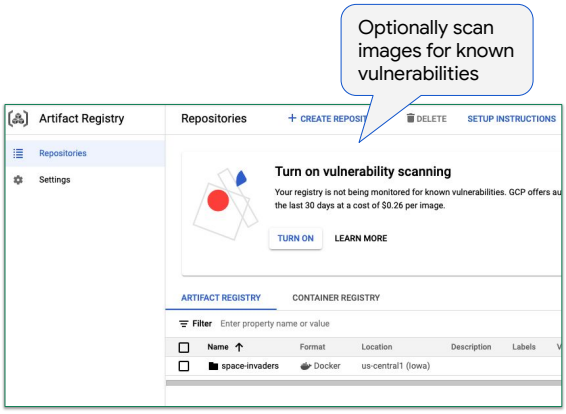
Repositories can be configured to publish messages to a specified Pub/Sub topic. Messages can be published when a user creates or deletes a repository or pushes a commit.

Some other features of Cloud Source Repositories include the ability to debug in production using Cloud Debugger, audit logging to provide insights into what actions were performed where and when, and direct deployment to App Engine. It is also possible to connect an existing GitHub or Bitbucket repository to Cloud Source Repositories. Connected repositories are synchronized with Cloud Source Repositories automatically.

Artifact Registry

Provides the same management features as Container Registry plus more:

- Offers
  - 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



Google Cloud

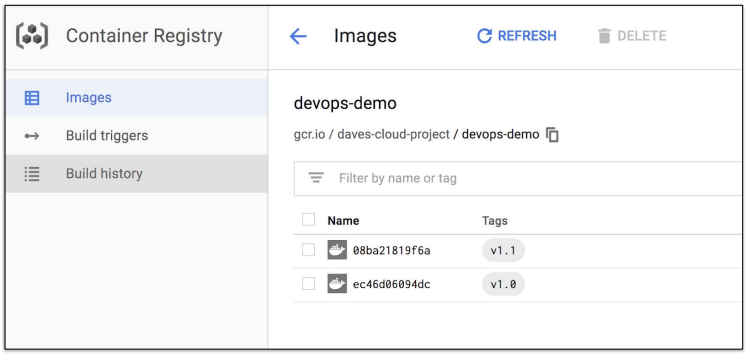
https://cloud.google.com/artifact-registry

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

Google Cloud Container Registry*

Docker registry provided by Google Cloud

- Private and only available to those who are given access via IAM



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

Google Cloud

*Artifact Registry is the recommended service for managing container images. Container Registry is still supported but will only receive critical security fixes

Container Registry
https://cloud.google.com/container-registry

Cloud Build

- Build software quickly across all programming languages, including Java, Go, Node.js, and more
- Create secure, production-ready container images from source code without a Dockerfile
- Deploy across multiple environments such as VMs, Cloud Run, Cloud Functions, App Engine and Kubernetes
- Fully managed service to build, test, or deploy source code located in Cloud Storage, Cloud Source Repositories, GitHub, or Bitbucket
    - Builds can be automated triggered when code changes are made

Google Cloud

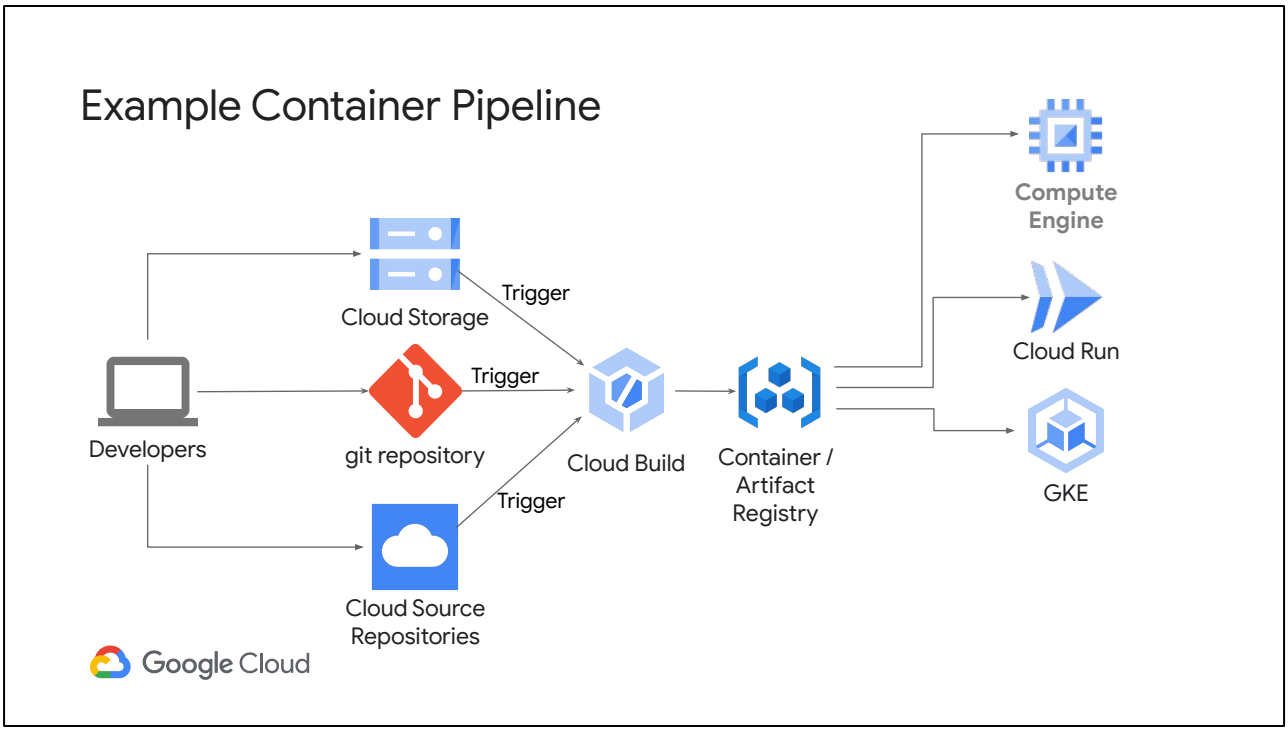
Cloud Build
https://cloud.google.com/cloud-build

Documentation
https://cloud.google.com/build/docs

Cloud Build is a service that executes your builds on Google Cloud Platform infrastructure. Cloud Build can import source code from Cloud Storage, Cloud Source Repositories, GitHub, or Bitbucket, execute a build to your specifications, and produce artifacts such as Docker containers or Java archives.

Cloud Build executes your build as a series of *build steps*, where each build step is run in a Docker container. A build step can do anything that can be done from a container irrespective of the environment. To perform your tasks, you can either use the supported build steps provided by Cloud Build or write your own build steps.

- Supported Build Steps → https://cloud.google.com/build/docs/deploying-builds/deploy-cloud-run
- Write your own build steps → https://cloud.google.com/build/docs/configuring-builds/use-community-and-custom-builders

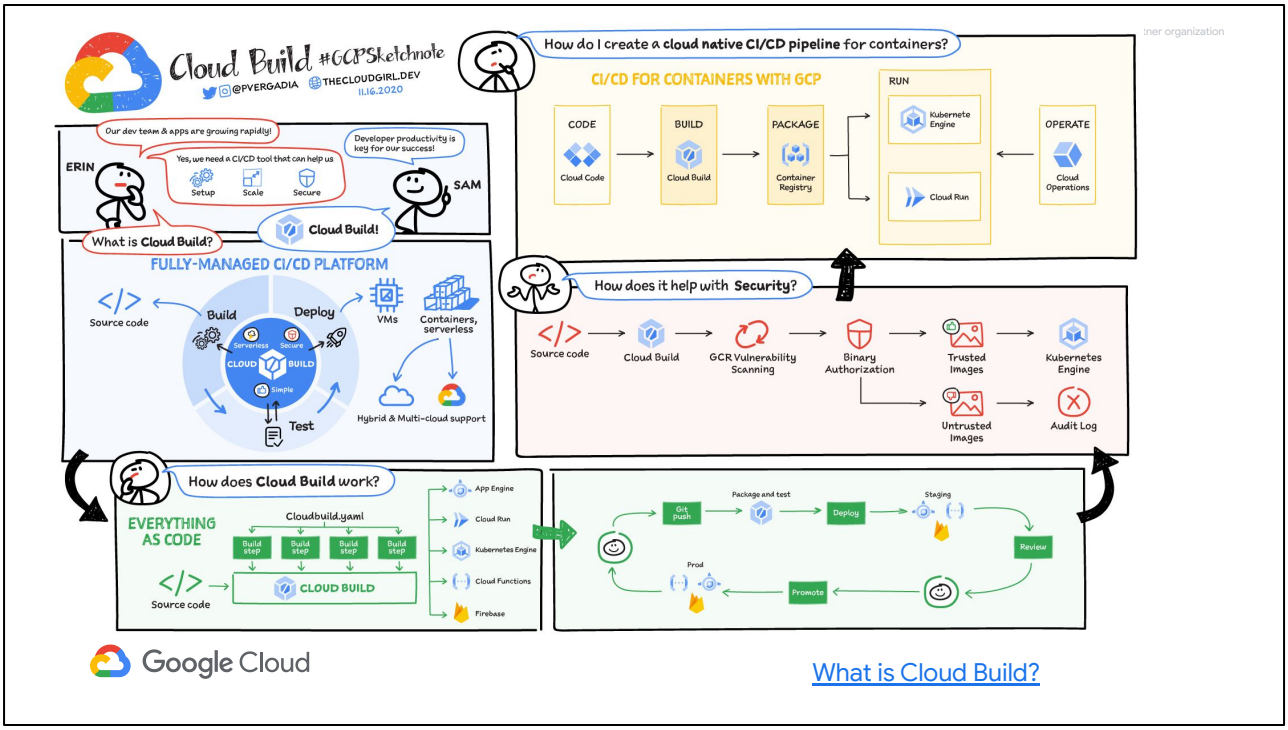


Continuous integration (CI)
https://cloud.google.com/solutions/continuous-integration

Building an automated serverless deployment pipeline with Cloud Build
https://cloud.google.com/blog/topics/developers-practitioners/building-automated-serverless-deployment-pipeline-cloud-build

Cloud Build can retrieve the source code for your builds from a variety of storage locations: Cloud Source Repositories, Cloud Storage, or git-compatible repositories like GitHub and Bitbucket.

To generate a build with Cloud Build, you define a series of steps. For example, you can configure build steps to fetch dependencies, compile source code, run integration tests, or use tools such as Docker, Gradle, and Maven. Each build step in Cloud Build runs in a Docker container.



What is Cloud Build?
https://www.youtube.com/watch?v=Bvo6jzC3J_A

Cloud Build - Deploy to Cloud Run example

```
steps:
# Build the docker image and tag it
- name: 'gcr.io/cloud-builders/docker'
  args: ['build', '-t',
'us-central1-docker.pkg.dev/spaceinvaders-images/space-invaders-v1/si-image:latest', '.']
# Push the container image to Artifact Registry
- name: 'gcr.io/cloud-builders/docker'
  args: ['push',
'us-central1-docker.pkg.dev/spaceinvaders-images/space-invaders-v1/si-image:latest']
# Deploy container image to Cloud Run
- name: 'gcr.io/google.com/cloudsdktool/cloud-sdk'
  entrypoint: gcloud
  args: ['run', 'deploy', 'space-invaders-from-cloud-build', '--image',
'us-central1-docker.pkg.dev/spaceinvaders-images/space-invaders-v1/si-image:latest',
'--region', 'us-central1']
# Name of the image in the Artifact Registry
images:
- us-central1-docker.pkg.dev/spaceinvaders-images/space-invaders-v1/si-image:latest
```

Cloud Build has multiple "builders": docker, go, kubectl, maven (mvn), Node.js (npm), cloud-sdk, git, and more*

Can trigger Cloud Build automatically via a push to a repository or run it manually `gcloud builds submit...`

Can also store artifacts in Cloud Storage

Using the cloud-sdk builder

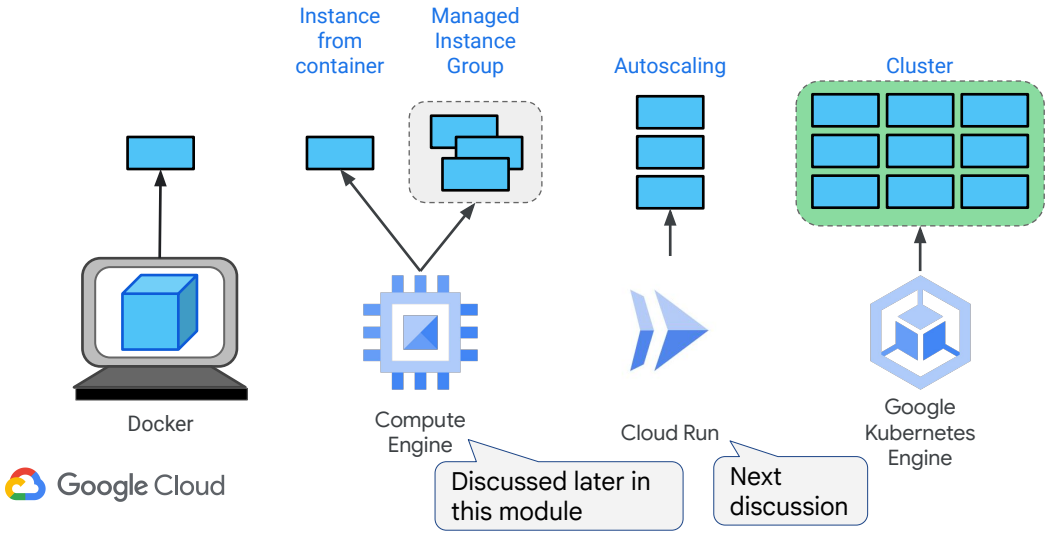
Google Cloud

*Complete list of Google Cloud Build images: https://cloud.google.com/build/docs/cloud-builders

Cloud Build - Create a build configuration file
https://cloud.google.com/build/docs/configuring-builds/create-basic-configuration

There are many good sections to review in this site - look at the different topics on the left.

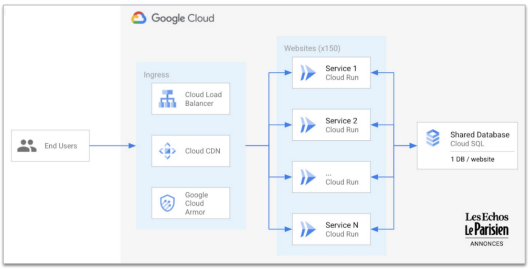
Where can you run containers?



Google Cloud

Cloud Run provides Containers-as-a-Service

- Fully managed serverless platform that runs individual containers.
- Based on the open-source knative project (https://knative.dev/)
- Two deployment methods
  - Fully managed
    - Autoscaling, connectivity managed by Google
  - Cloud Run for Anthos
    - Knative running on a GKE cluster
      - Allows you to:
        - Access VPC network
        - Tune size of compute engine instance, use GPUs, etc.
        - Run service in all GKE regions



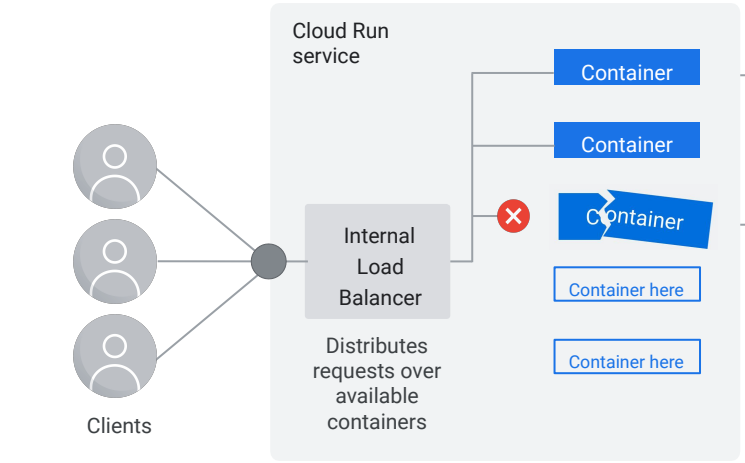
Scaling quickly to new markets with Cloud Run—a web modernization story

Google Cloud

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

Knative
https://knative.dev/

All incoming requests are handled with automatic scaling

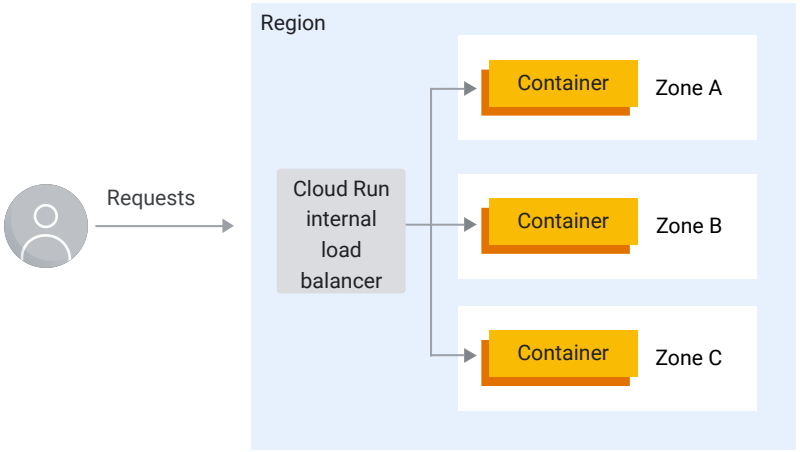


Auto scaling: Add containers to handle requests and remove them if they are unused

Auto healing: Monitor and replace unhealthy containers

Google Cloud

Cloud Run balances containers across zones in a region



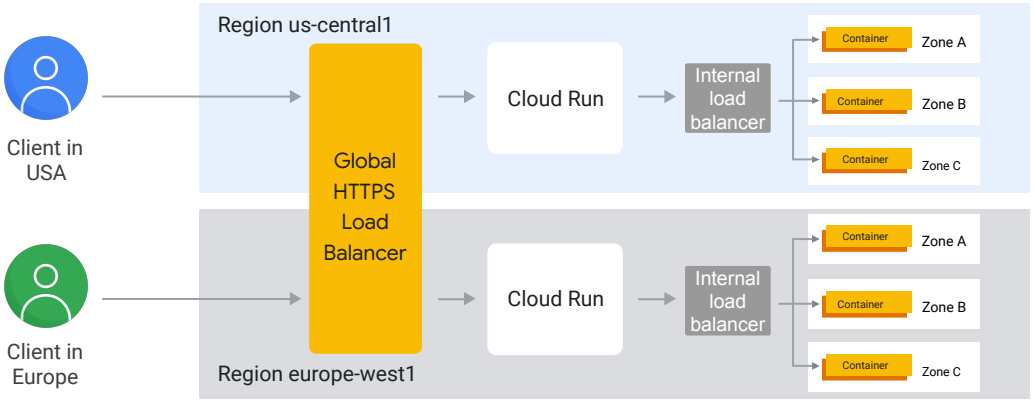
A region is a data center. For example: Council Bluffs (Iowa, North America)

Every region has three or more zones.

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

Google Cloud

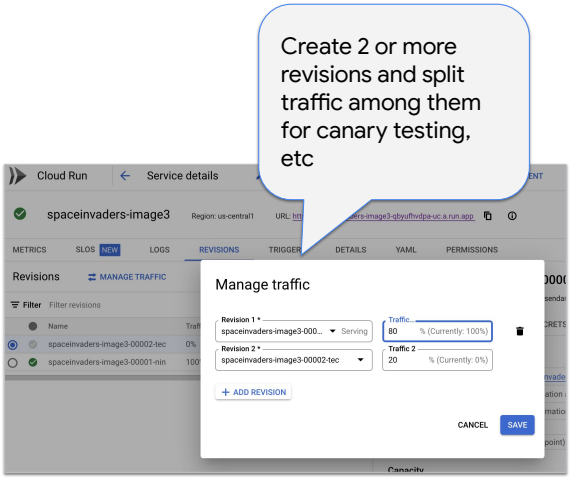
Global load balancer delivers lowest end-user latency



Google Cloud

Cloud Run traffic splitting

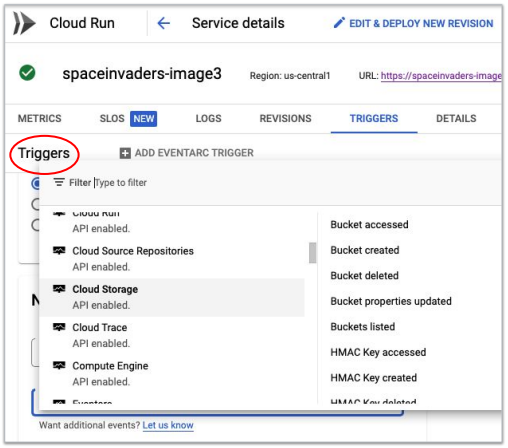
- Split traffic across two or more revision
  - Great for canary deployments
  - Gradual rollouts for revisions
- Easily roll back to a previous revision



Google Cloud

Invoking Cloud Run

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

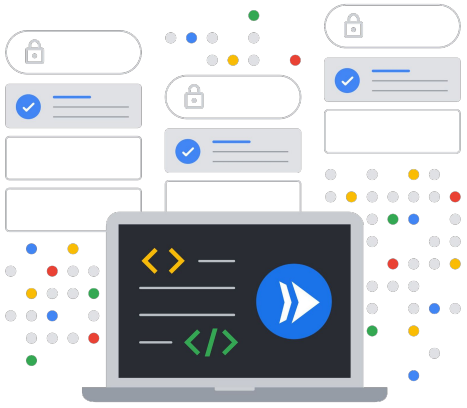


Google Cloud

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

Cloud Run use cases

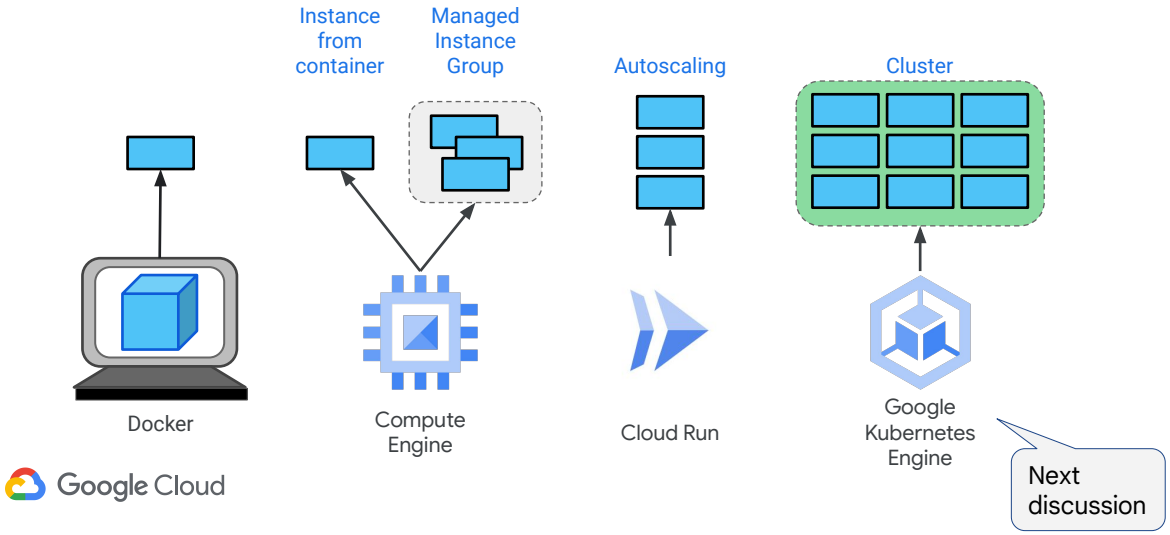
- Web applications, public APIs or websites
- Microservices-based applications that communicate using direct or asynchronous messages
- Event processing
- Scheduled tasks shorter than 60 minutes



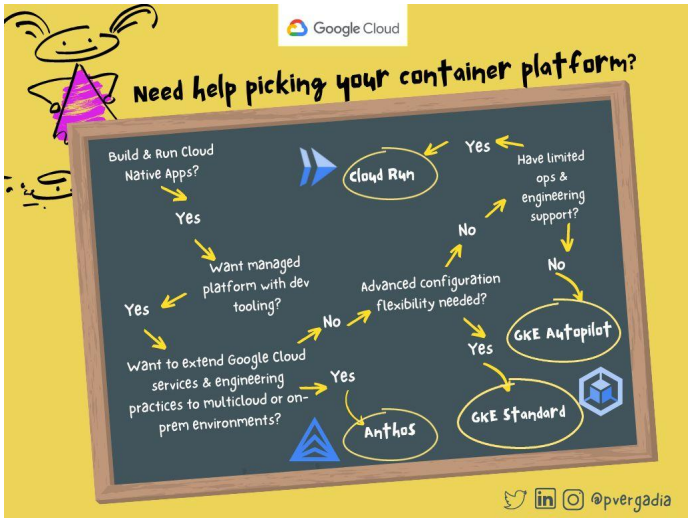
Other example use cases found toward the bottom of this page: https://cloud.google.com/run/

Google Cloud

Where can you run containers?



Google Cloud

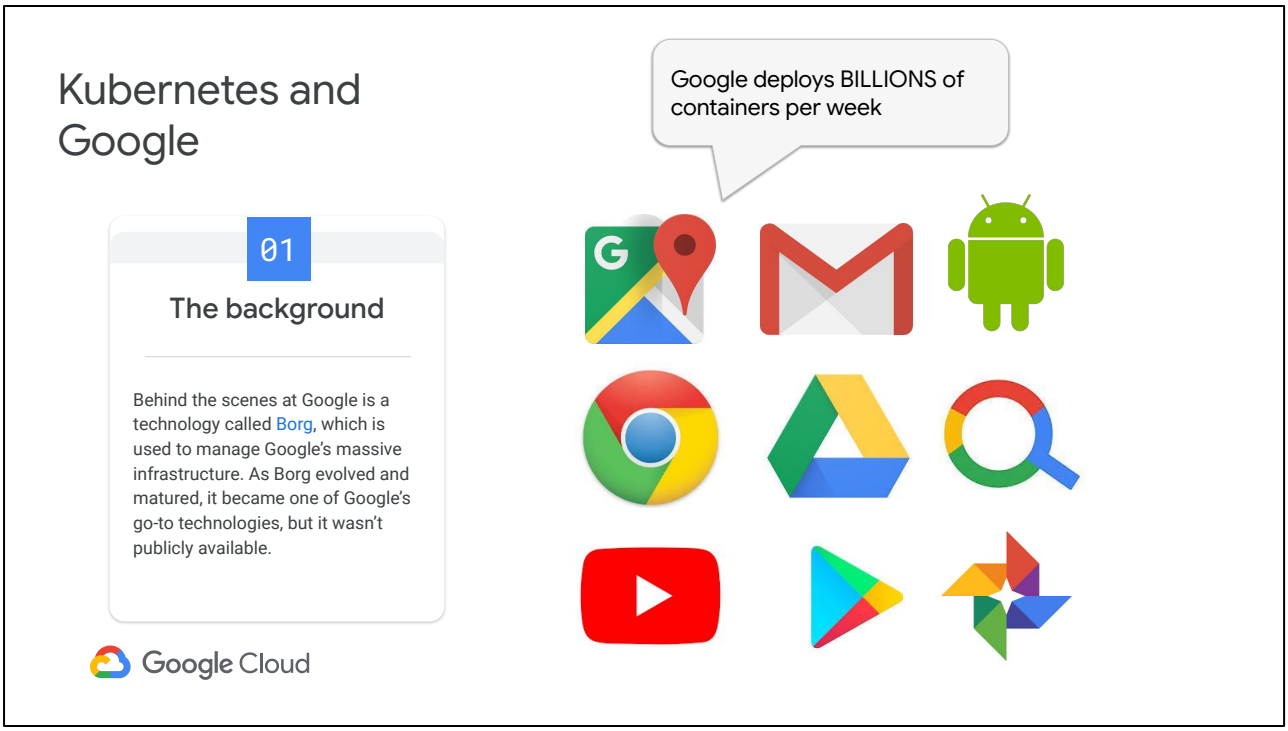


Posted on Twitter, May 12, 2021
https://mobile.twitter.com/pvergadia/status/1392555417553747968/photo/1

Google Cloud

Priyanka Vergadia
Developer Advocate at Google

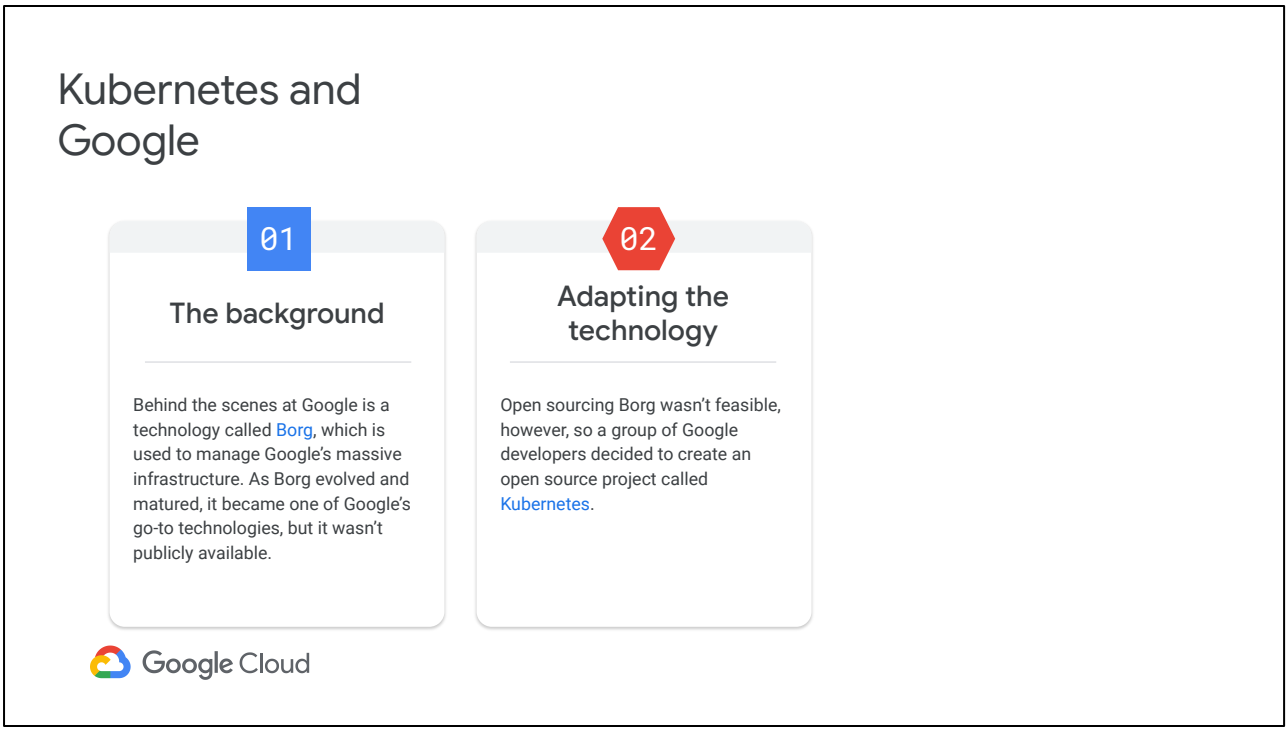
Posted on Twitter, May 12, 2021
https://mobile.twitter.com/pvergadia/status/1392555417553747968/photo/1



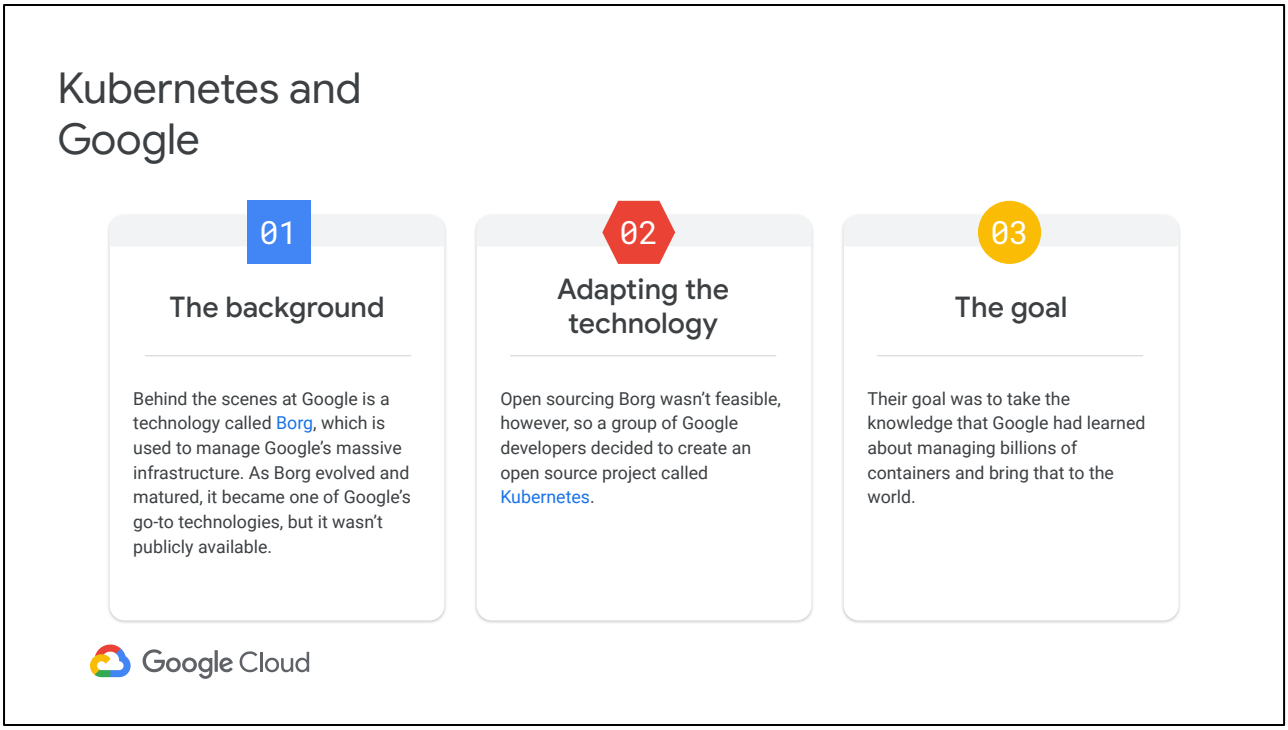
Google is the pioneer when it comes to containerized services. From Gmail to YouTube to Search, everything at Google runs in containers. Containerization allows development teams to move fast, deploy software efficiently, and operate at an unprecedented scale.

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



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.

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:

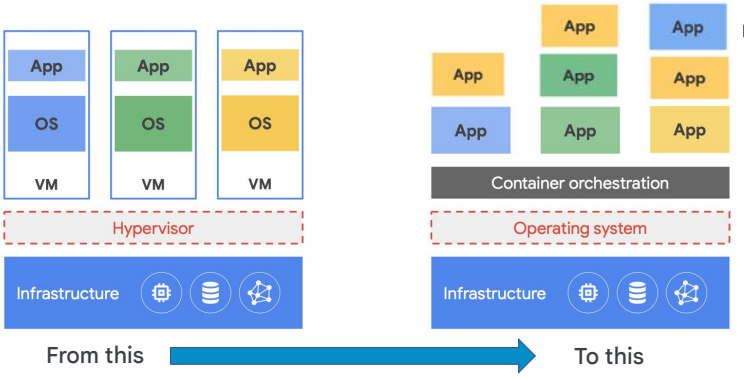
- Support on Google Cloud using Google Kubernetes Engine or GKE
- Support on Microsoft Azure using Azure Container service or AKS
- Support on AWS using Elastic Kubernetes Service or EKS
- Support on OpenStack
- Mesos and Pivotal Cloud Foundry also supports Kubernetes

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



Google Cloud

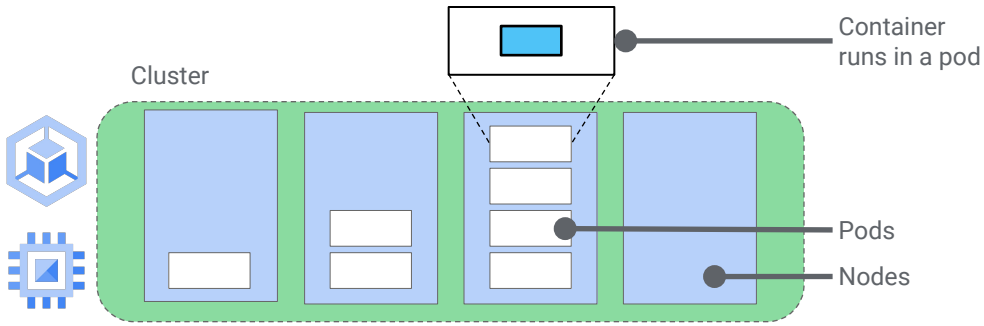
The word Kubernetes comes from the Greek word for helmsman or pilot

- 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 Cloud

Kubernetes cluster has nodes, pods, and containers



Google Cloud

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

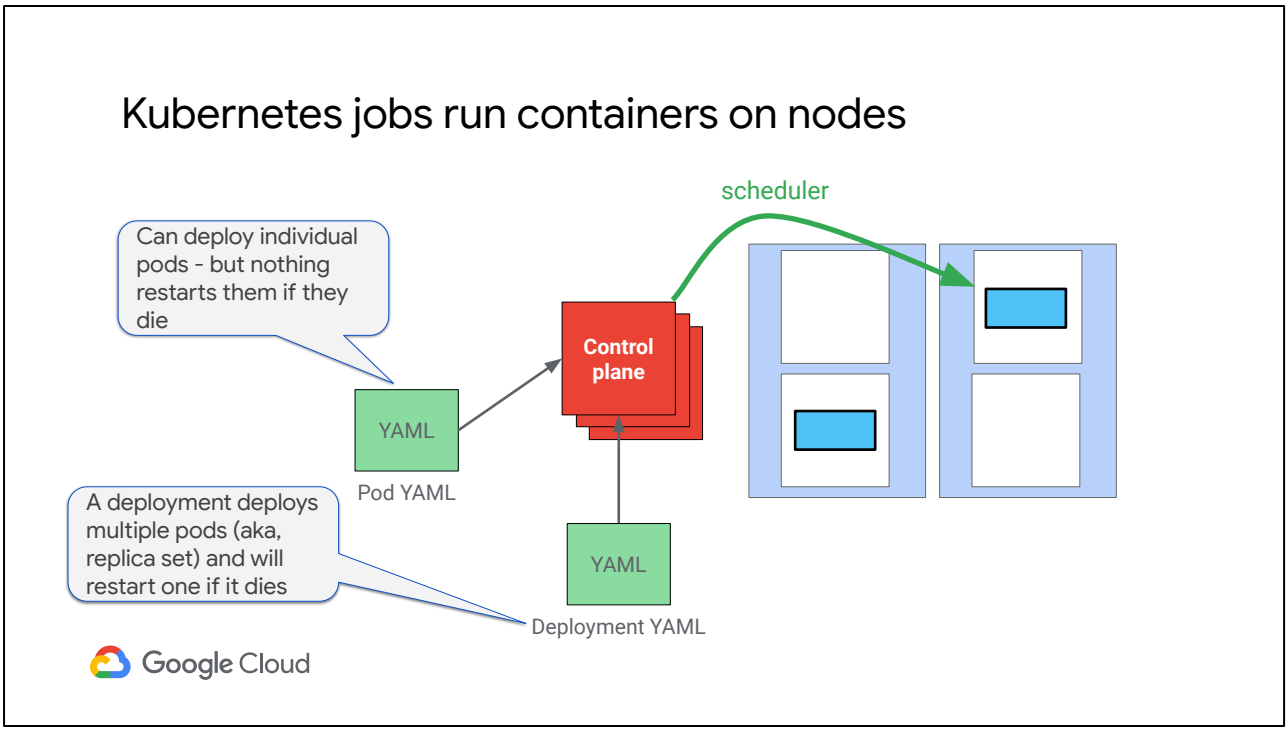
A Kubernetes cluster is composed of nodes, which are a unit of hardware resources. Nodes in GKE are implemented as VMs in Compute Engine. Each node has pods. Pods are resource management units. A pod is how Kubernetes controls and manages resources needed by applications and how it executes code. Pods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

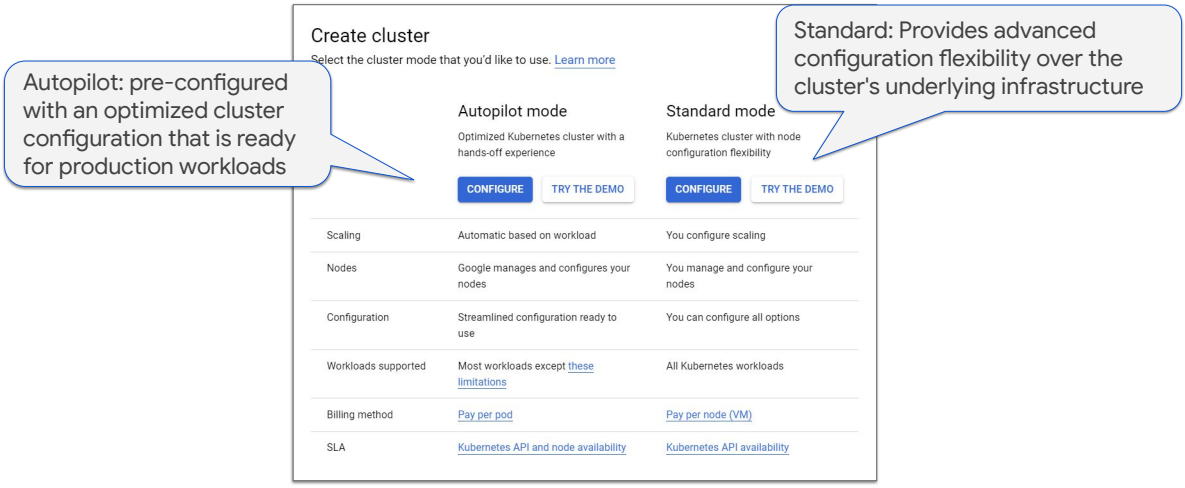
This design enables advanced controls and operations that gives systems built on Kubernetes unique qualities.



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.

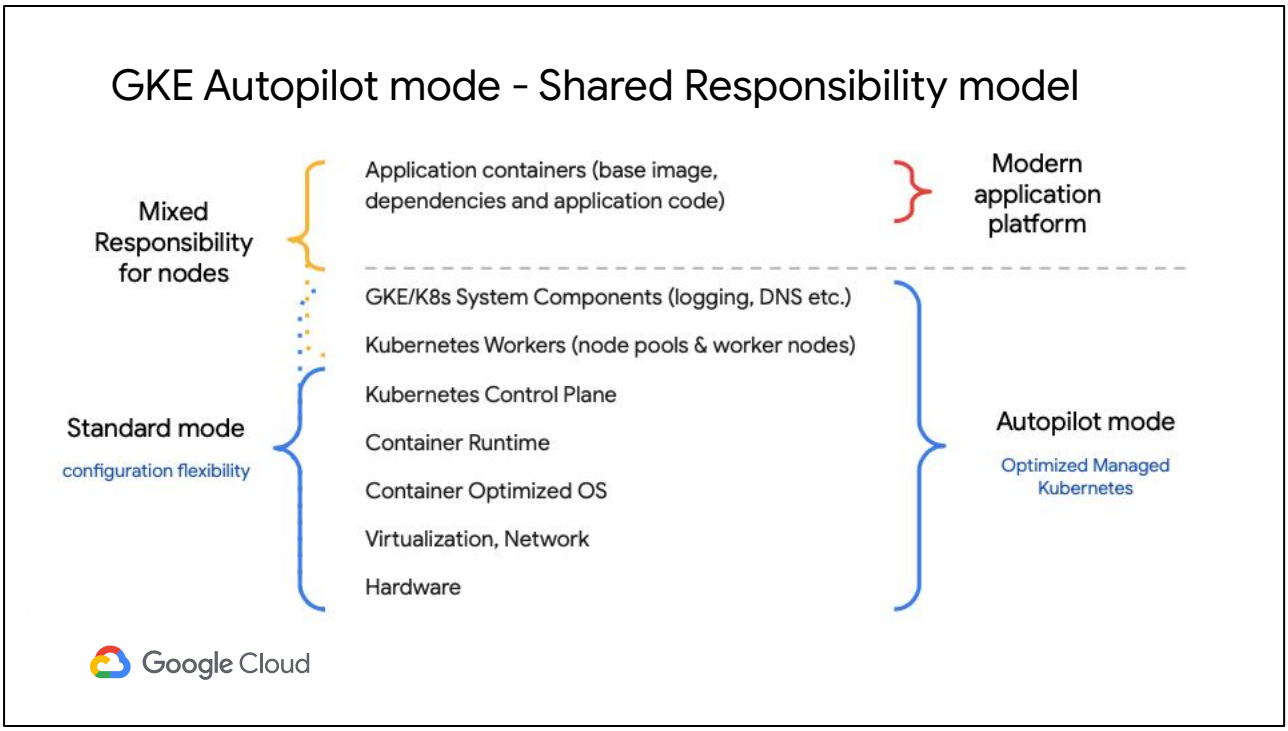
For example a pod YAML file provides the information to start up and run a pod on a node. If for some reason a pod stops running or a node is lost, the pod will not automatically be replaced. The Deployment YAML tells Kubernetes how many pods you want running. So the Kubernetes deployment is what keeps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

Creating Kubernetes Engine Clusters



Google Cloud

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



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

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

Google Cloud

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

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

kubectl Cheat Sheet
https://kubernetes.io/docs/reference/kubectl/cheatsheet/

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 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

Show details of a deployment

```
kubectl describe deployments devops-deployment
```

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
```

Google Cloud

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

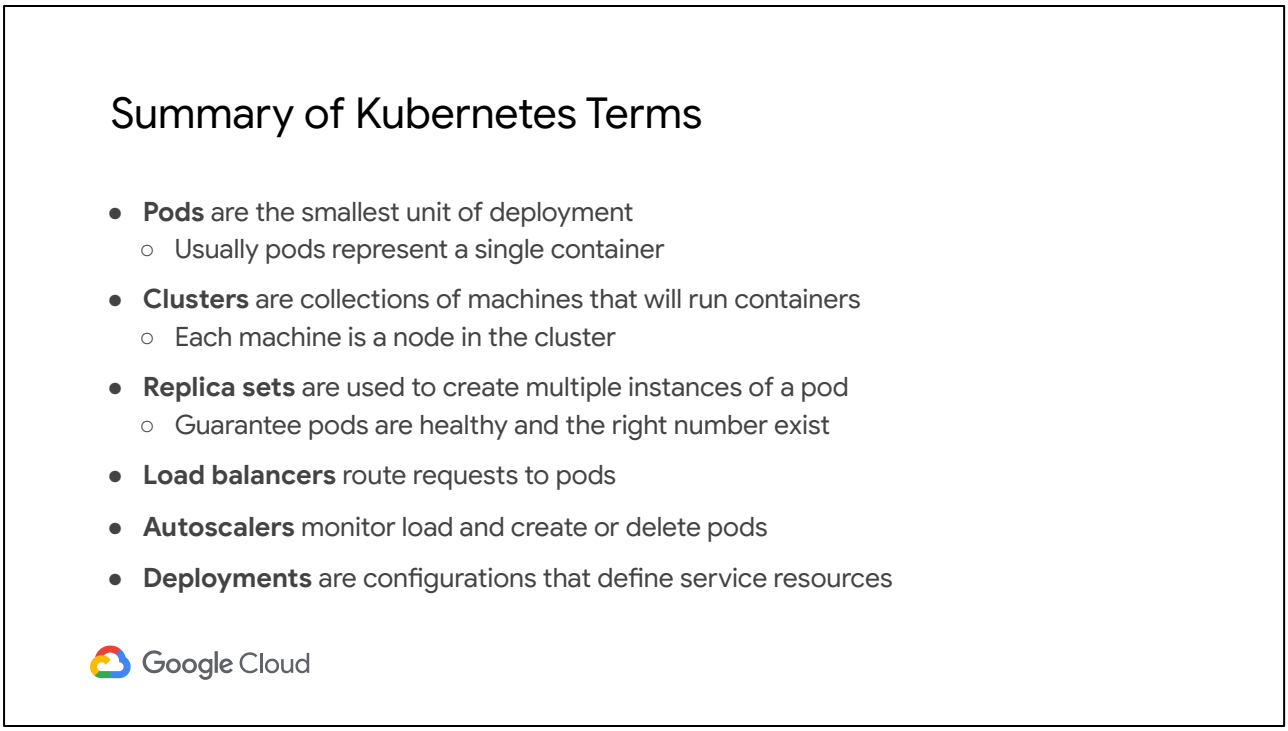
```
kubectl delete services [name here]
```

Google Cloud

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.



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.

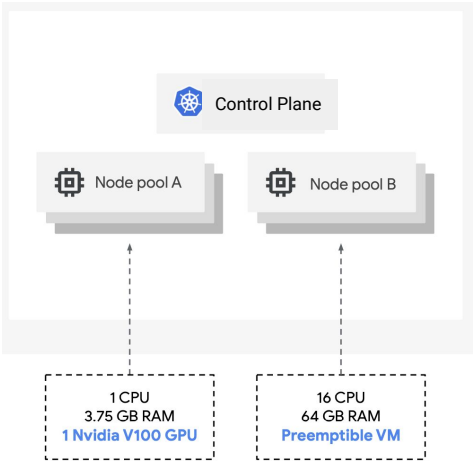
Kubernetes can leverage cloud load balancers to distribute traffic to multiple pods. The load balancer is run as a network service.

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

Deployments are configurations that define service resources.

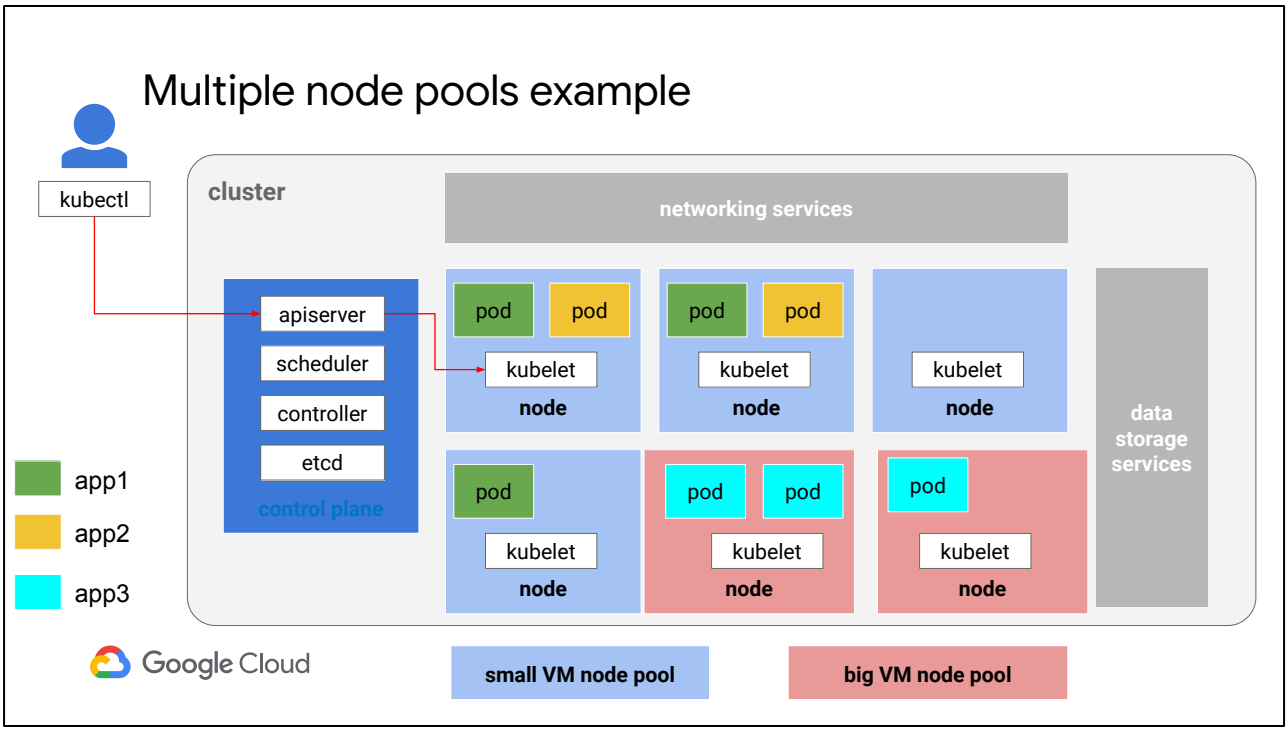
Node pools are cluster subsets with identical machine configurations

- 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



Google Cloud

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



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.

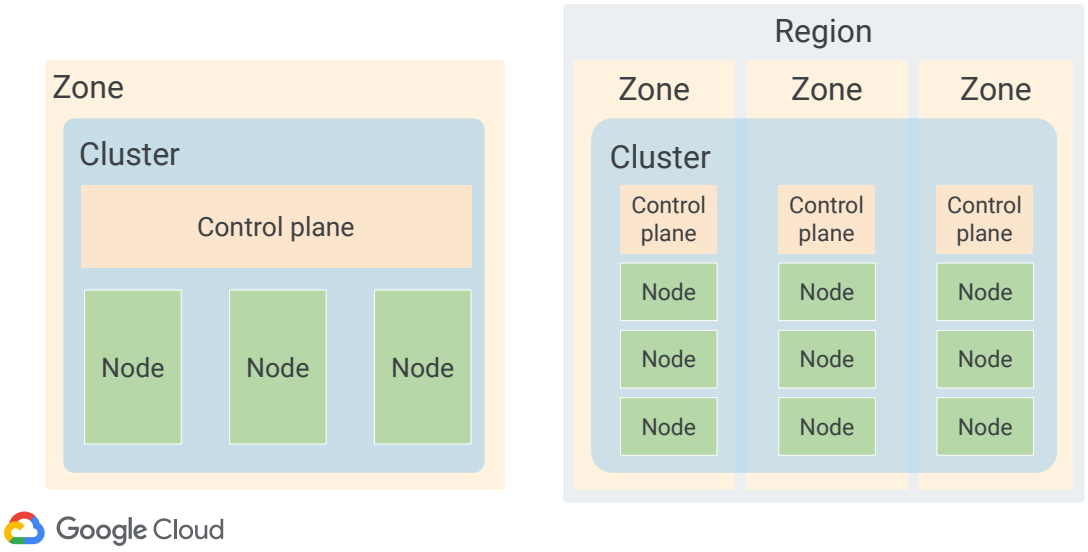
Cluster administrators configure the cluster by sending requests to **apiservers** on the Control Plane using a command-line tool called **kubectl**. Kubectl can be installed and run anywhere.

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).

Zonal vs regional clusters



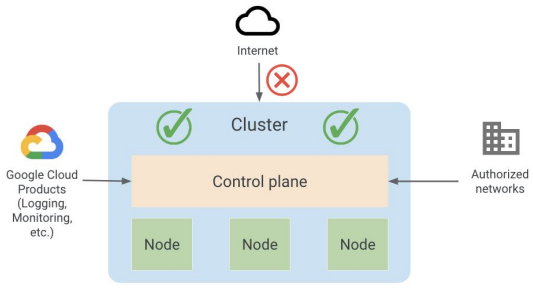
By default, a cluster launches in a single Google Cloud compute zone with three identical nodes, all in one node pool. The number of nodes can be changed during or after the creation of the cluster. Adding more nodes and deploying multiple replicas of an application will improve an application's availability. 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



Google Cloud

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

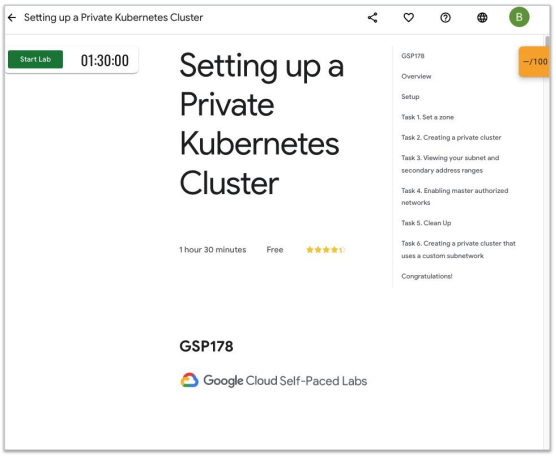
Cluster control planes can be accessed by Google Cloud products, such as Cloud Logging or Cloud Monitoring, through an internal IP address.

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.

Suggested Lab: Setting up a Private Kubernetes Cluster

Topics include

- Creating a private cluster where GKE automatically creates the subnets
- Creating a private cluster where you create the subnets
- Enabling authorized networks to access the private cluster



https://partner.cloudskillsboost.google/catalog_lab/908

Google Cloud

Setting up a Private Kubernetes Cluster
https://partner.cloudskillsboost.google/catalog_lab/908

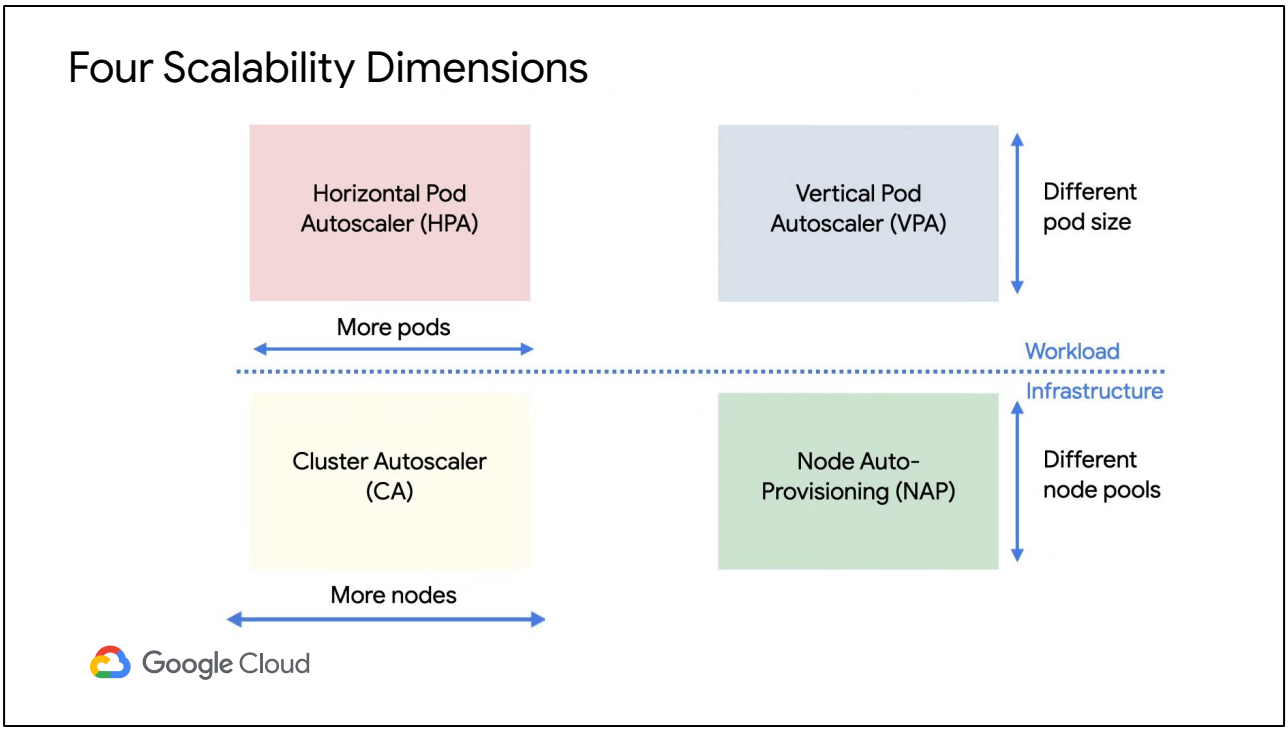
Autoscaling Strategies

Horizontal Pod Autoscaler (HPA)

Vertical Pod Autoscaler (VPA)

Cluster Autoscaler (CA)

Node Auto Provisioning (NAP)

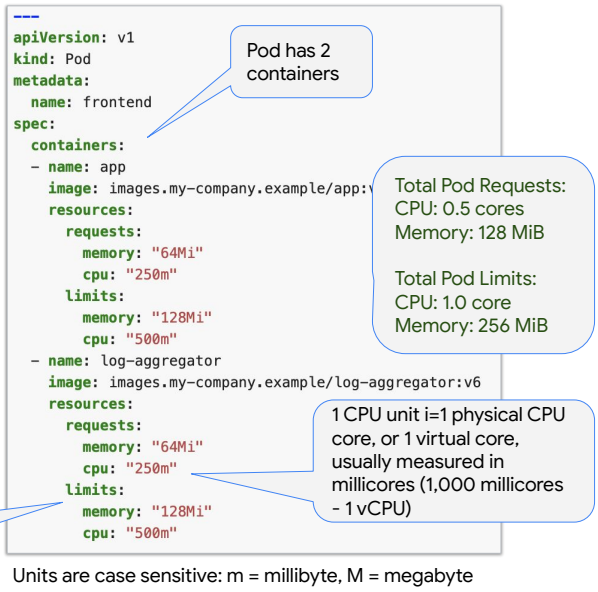


Configuring multidimensional Pod autoscaling
https://cloud.google.com/kubernetes-engine/docs/how-to/multidimensional-pod-autoscaling

Recommendations for planning, architecting, deploying, scaling, and operating large workloads on Google Kubernetes Engine (GKE) clusters
https://cloud.google.com/kubernetes-engine/docs/best-practices/scalability

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



Google Cloud

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

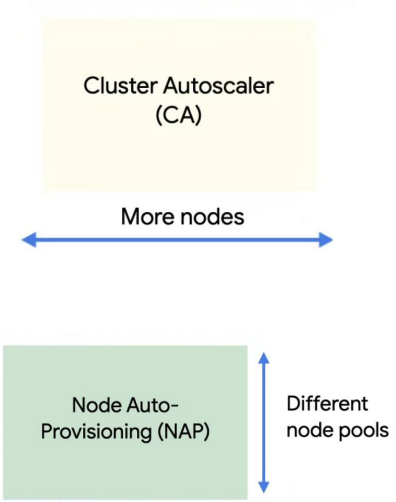
In Kubernetes, 1 CPU unit is equivalent to **1 physical CPU core**, or **1 virtual core**, usually measured in millicores (1,000 millicores - 1 vCPU)

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



Google Cloud

Autoscaling nodes

- **Cluster autoscaler** - 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

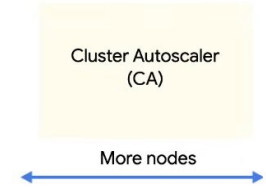
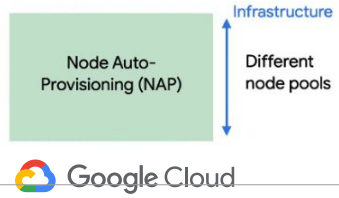
Google Cloud

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

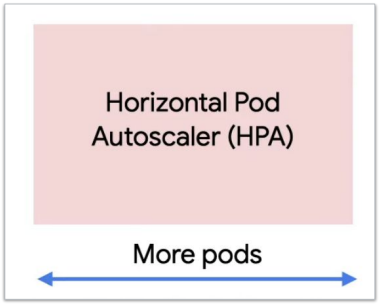
Enabling node auto-provisioning:
https://cloud.google.com/kubernetes-engine/docs/how-to/node-auto-provisioning#enable

Summary: Cluster Autoscaling vs Node Auto-provisioning

	Automatically resizes node pools based on the workload demands High demand: adds nodes to the node pool. 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 • Under provisioned: Extra overhead of adding additional nodes when needed (+ node cost) • Over provisioned: Paying too much per node
 Google Cloud	Google optimizes the compute resources in each node pool to match workload requirements	• Not best for sudden spikes in traffic • 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



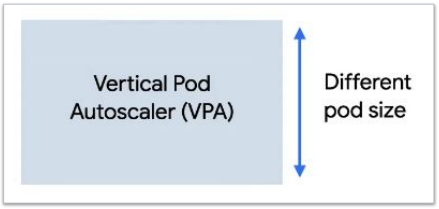
Google Cloud

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

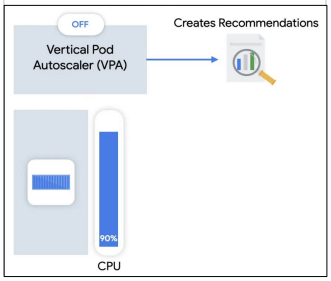


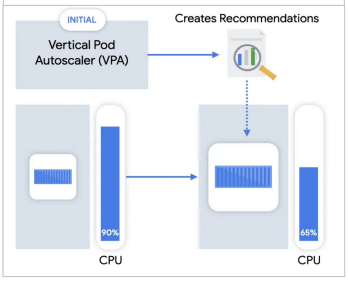
Google Cloud

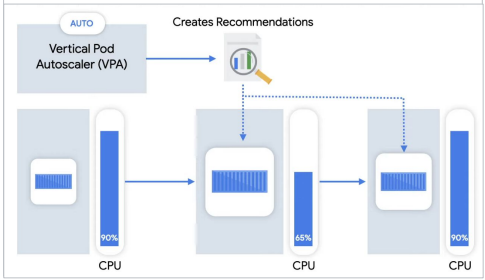
Vertical pod scaling
https://cloud.google.com/kubernetes-engine/docs/concepts/verticalpodautoscaler

Vertical Pod Autoscaler Modes

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





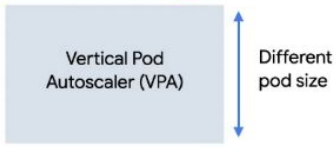
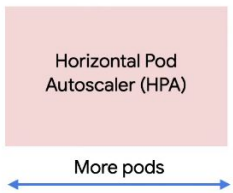


- Google recommends leaving VPA off for at least a week to accumulate recommendations
  - Then switch to Initial or Auto based on needs

Google Cloud

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 Can use once or on a ongoing basis	If you do it manually: • Over-estimating memory/cpu results in over-provisioning the # of nodes needed when pods scale up, increasing costs • 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

Kubernetes Engine StatefulSets

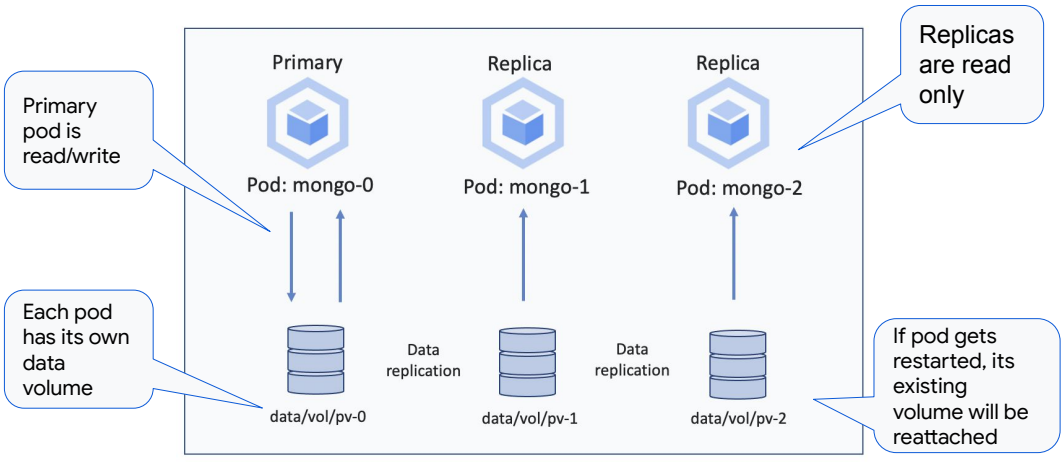
- StatefulSets represent a set of Pods with unique, persistent identities and stable hostnames that GKE maintains regardless of where they are scheduled
- Some examples of use cases include:
  - A Redis pod that needs to maintain access to the same storage volume
    - Even if it is redeployed or restarted
  - 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

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

Google Cloud

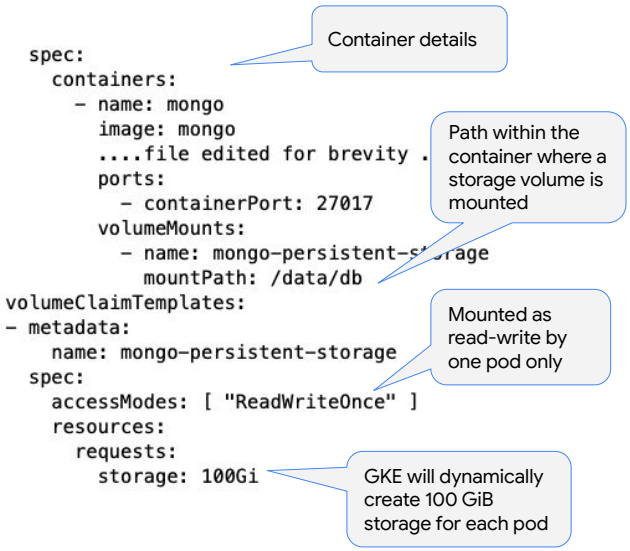
There are several MongoDB implementations in Cloud Marketplace. MongoDB automatically handles the replication.

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

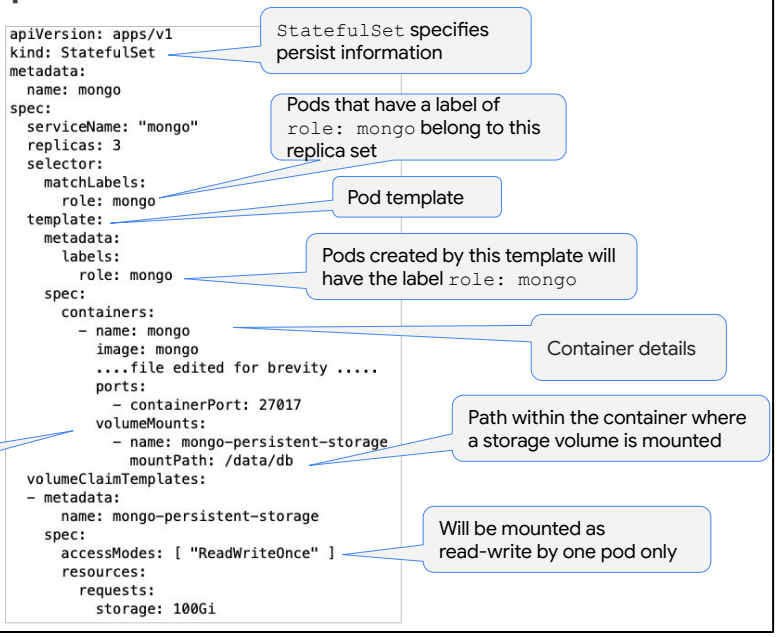


The prior page mentions volumes. This is an example of a manifest that creates a dynamic volume for a MongoDB container

Deploying a stateful application

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

```
kubectl apply -f
STATEFULSET_FILENAME
```

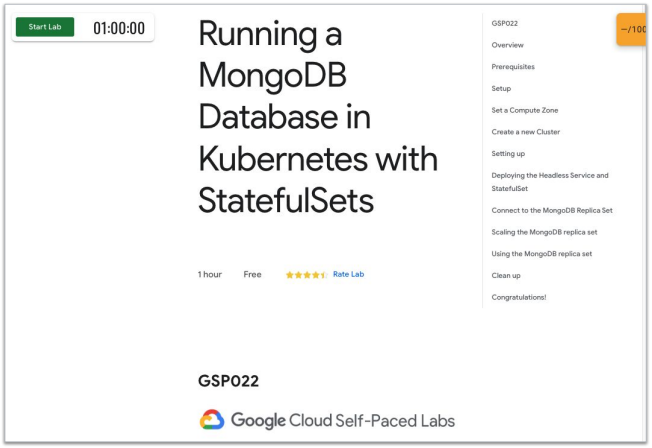


Google Cloud

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.



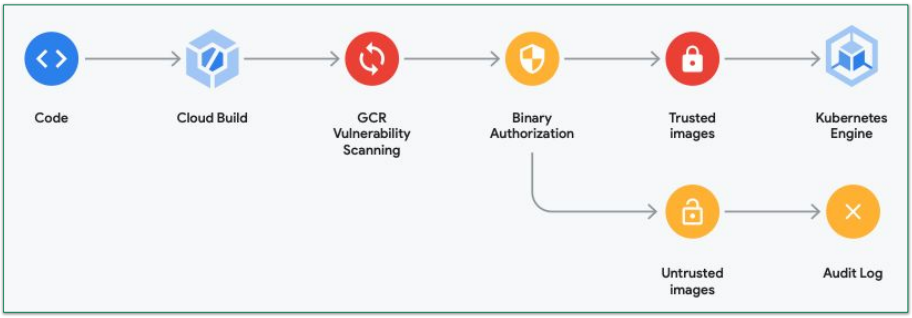
Google Cloud

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

Kubernetes Engine...Binary Authorization

Binary authorization - deploy trusted containers

- Managed service
- Adds a policy that requires signed images
- When an image is built by Cloud Build an “attestor” verifies that it was from a trusted repository (Google Source Repositories, for example).
- Container/Artifact Registry includes a vulnerability scanner that scans containers



Google Cloud

Binary authorization ensures that internal processes that safeguard the quality and integrity of your software have been successfully completed before an application is deployed to your production environment. Binary authorization is a Google Cloud service and is based on the Kritis specification: https://github.com/grafeas/kritis/blob/master/docs/binary-authorization.md

The Kritis signer listens to Pub/Sub notifications from a container registry vulnerability scanner when new image versions are uploaded and makes an attestation if the image passed the vulnerability scan. Google Cloud binary authorization service then enforces the policy requiring attestations by the Kritis signer before a container image can be deployed.

This flow prevents deployment of images with vulnerabilities below a certain threshold.

More details can be found at:

https://cloud.google.com/binary-authorization/docs/cloud-build

https://cloud.google.com/binary-authorization/docs/vulnerability-scanning

Suggested Lab (if time allows)

https://partner.cloudskillsboost.google/catalog_lab/1718

Google Cloud

https://partner.cloudskillsboost.google/catalog_lab/1718

From the exam guide case studies - EHR Healthcare

Technical requirements section

- Maintain legacy interfaces to insurance providers with connectivity to both on-premises systems and cloud providers.
  - **Cloud Interconnect and/or VPN**
- Provide a consistent way to manage customer-facing applications that are container-based.
  - **Kubernetes Engine, Anthos**
- Provide a secure and high-performance connection between on-premises systems and Google Cloud.
  - **Cloud Interconnect**
- Provide consistent logging, log retention, monitoring, and alerting capabilities.
  - **Cloud Operations**
- Maintain and manage multiple container-based environments.
  - **Anthos**

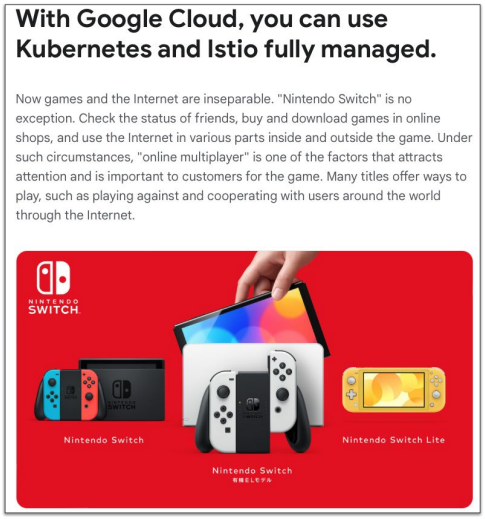
Note: The items in **green** are services that could possibly fit the requirements. This does NOT MEAN that these are correct and should be used as your choices during the exam. This serves as a lead-in to the Anthos discussion coming up next.

Google customer case study - Nintendo

Nintendo supports it's multiplayer game environments using:

- **GKE**
- **Anthos Service Mesh**
- Spanner
- Cloud Load Balancing
- Cloud Storage
- BigQuery
- Cloud Operations

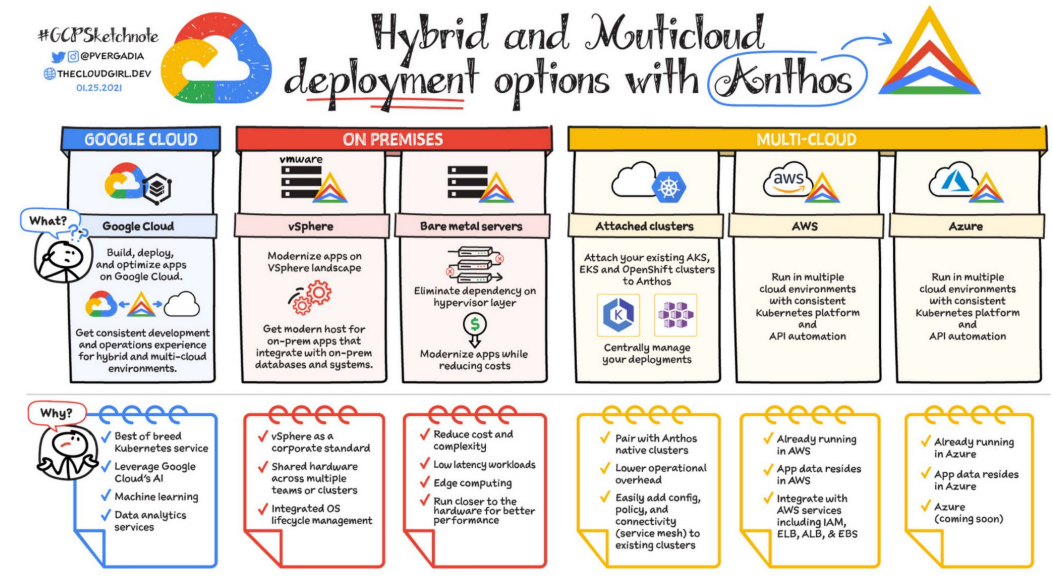
Google Cloud



Nintendo: Build a new general-purpose game server using Google Kubernetes Engine, Cloud Spanner, etc.

Note: The link is in Japanese. You will need to use the browser's translator capabilities (unless you speak Japanese).

Nintendo: Build a new general-purpose game server using Google Kubernetes Engine, Cloud Spanner, etc.
https://cloud.google.com/blog/ja/topics/customers/nintendo-new-game-servers-built-with-gke-cloud-spanner?hl=ja

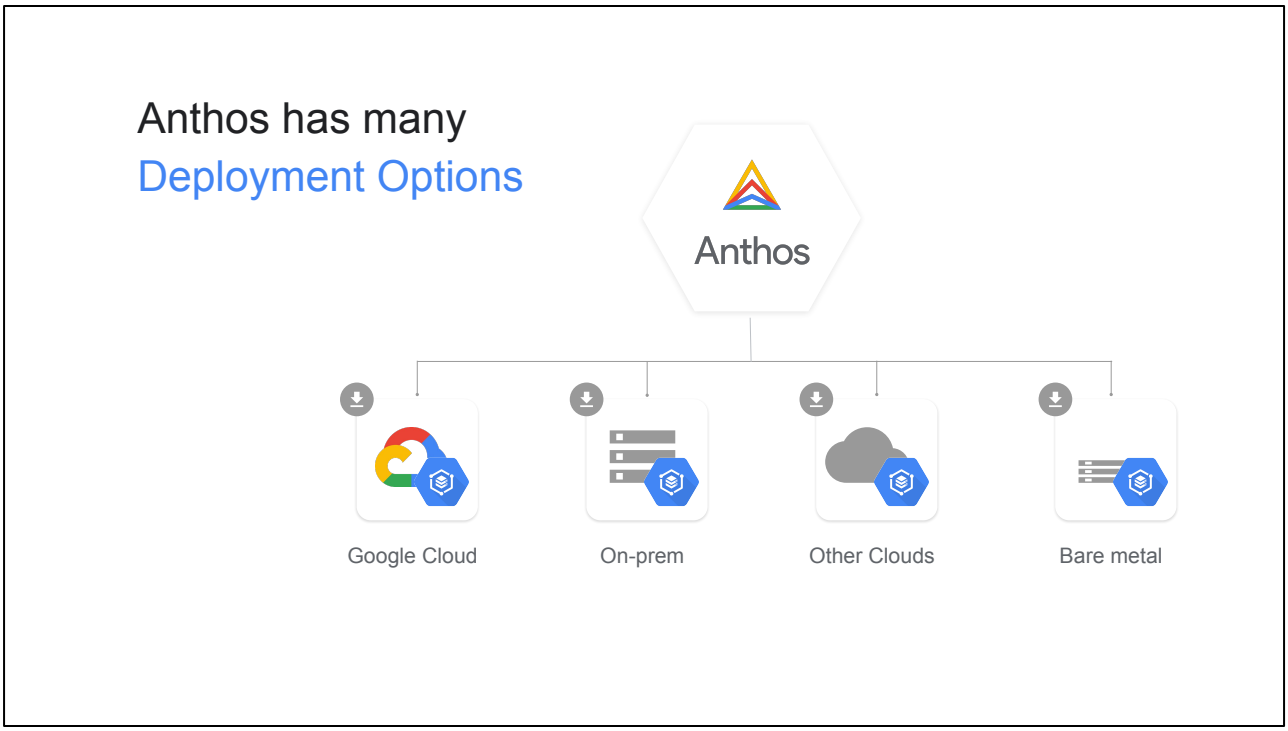


Anthos
https://cloud.google.com/blog/topics/developers-practitioners/what-are-my-hybrid-and-multicloud-deployment-options-anthos

Anthos Overview

- Anthos overview
  - Anthos overview
- Anthos blog
  - https://cloud.google.com/blog/topics/developers-practitioners/how-does-anthos-simplify-hybrid-multicloud-deployments
- Youtube
  - https://www.youtube.com/watch?v=FfJNAjoX3Uc&list=PLTWE_Imu2InBzuPmOcgAYP7U80a87cpJd

Google Cloud



Anthos clusters provide a unified way to work with Kubernetes clusters as part of Anthos, extending GKE to work in multiple environments. You have consistent, unified, and secure infrastructure, cluster, and container management, whether you're using Anthos on Google Cloud (with traditional GKE), hybrid cloud, or multiple public clouds.

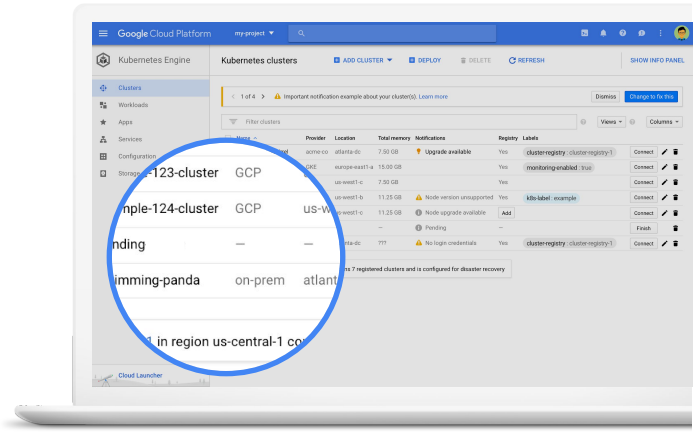
Anthos has many deployment options. There are GKE-based options including:

- On Google Cloud with GKE
- On-prem with VMware
- On-prem or at the edge with bare metal
- On AWS

You can also attach non-GKE clusters as connected clusters - these can run in any environment, and take advantage of some Anthos functionality.

Anthos Dashboard

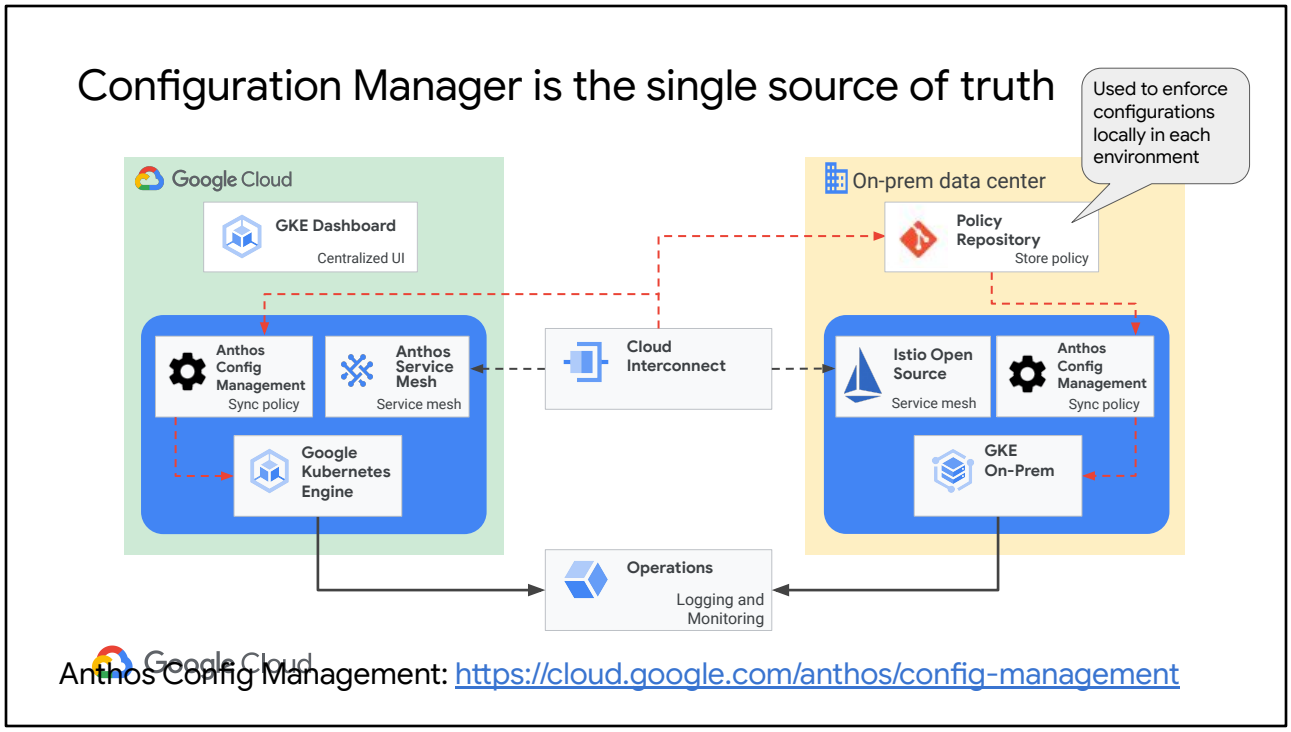
- Orchestrate and manage on-prem containers just like GKE in the cloud
- Consistent operating model with access to GCP services across hybrid environments
- Single-pane-of-glass for multiple Kubernetes clusters, no matter where



Unified UX across cloud and on-premises

The ability to take hundreds of clusters, be they on premise, across multiple clouds or at the edge of the of the network and present them on a single pane of glass.
This is achieved using GKE Connect.

- GKE connect is a Kubernetes pod deployed into its own namespace in one of the user clusters.
- A Pod is granted the credentials needed to talk back to Google.
- It creates an outbound TLS connection to the Google cloud.
- The outbound connection does not require customers to open up rules on firewall.
- The connection is launched from inside the firewall, goes outside the firewalls and establishes a HTTPs connection.
- Once the connection is established a secure tunnel of communication is set up between the on-premise environment and the cloud



Google Kubernetes Engine:

- Is a managed, production-ready environment for deploying containerized applications.
- Operates seamlessly with high availability and an SLA.
- Runs Certified Kubernetes, ensuring portability across clouds and on-premises.
- Includes auto node repair, auto upgrade, auto scaling.

Uses regional clusters for high availability with multiple control planes, node storage replication across multiple zones

Its counterpart on the on-premises side of our hybrid network is GKE On-Prem.

GKE On-Prem:

- Is a turn-key, production-grade, conformant version of Kubernetes with a best-practice configuration already pre-loaded.
- Provides an easy upgrade path to the latest Kubernetes releases that have been validated and tested by Google.
- Provides access to Container services on Google Cloud, such as Cloud Build, Container Registry, Cloud Audit Logs, and more.
- Integrates with Istio, Knative, and Cloud Marketplace solutions.
- Ensures a consistent Kubernetes version and experience across cloud and on-premises environments.

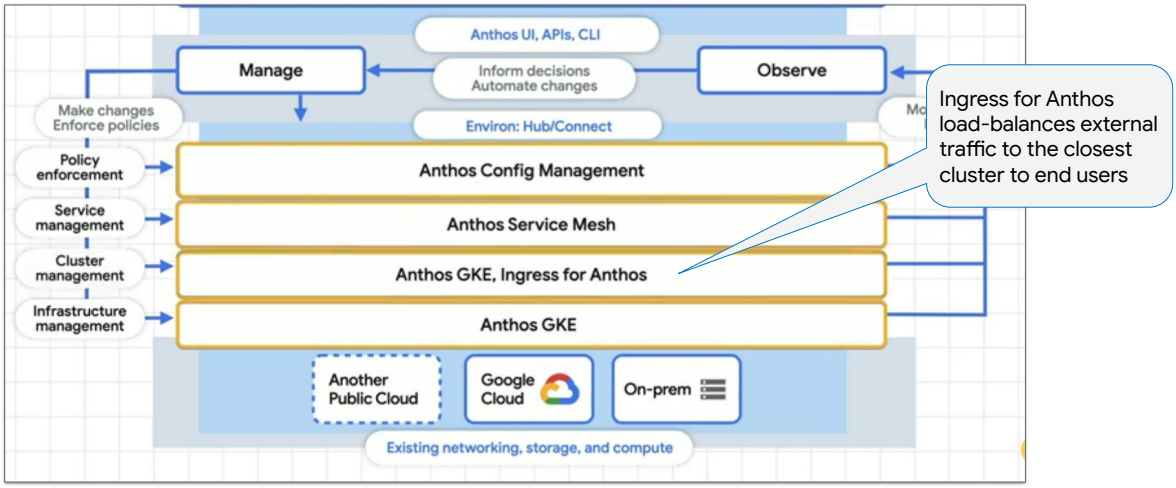
Enterprise applications may use hundreds of microservices to handle computing workloads. Keeping track of all of these services and monitoring their health can quickly become a challenge. Anthos and Istio Open Source service meshes take all of the guesswork out of managing and securing your microservices. These service mesh layers communicate across the hybrid network using Cloud Interconnect, as shown, to sync and pass their data.

Cloud Logging and Cloud Monitoring are the built-in logging and monitoring solutions for Google Cloud. Google Cloud's operations suite offers a fully managed logging, metrics collection, monitoring, dashboarding, and alerting solution that watches all sides of your hybrid or multi-cloud network. It's the ideal solution for customers wanting a single, easy to configure, powerful cloud-based observability solution that also gives you a single pane of glass dashboard to monitor all of your environments.

Lastly, Anthos Configuration Management provides a single source of truth for your clusters configuration. That source of truth is kept in the Policy Repository, which is actually a git repository (in this illustration this repository happened to be located on premises, but it can also be hosted in the cloud.) The Anthos Configuration Management agents use the Policy Repository to enforce configurations locally in each environment, managing the complexity of owning clusters across environments.

Anthos Configuration Management also provides administrators and developers the ability to deploy code changes with a single repository commit, and the option to implement configuration inheritance by using Namespaces.

Anthos Architecture



Google Cloud

Image from “What is Anthos” youtube video
https://www.youtube.com/watch?v=Qtwt7QcW4J8&t=276s

Anthos Config Management unifies access policies and provides governance across multiple clusters.

Anthos Service Mesh allows you to set and monitor SLOs on your services, secure traffic between services even across the cluster boundary.

And ingress for Anthos allows you to load-balance external traffic to the closest cluster to your end users.

This helps you to build low latency and high availability services in multiple GCP regions.

Anthos Config Management - Centralized Configuration Management

- An at-a-glance overview of the configuration state of all your clusters
- Deploy and monitor configuration changes stored in a central Git repository
  - Auditable
  - Revertable
  - Single source of truth
- Provides governance across multiple clusters

Google Cloud

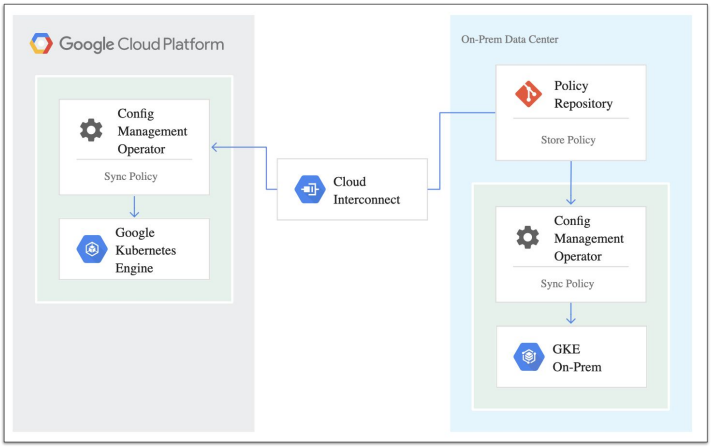


Image and more detail here:
https://cloud.google.com/anthos/docs/concepts/overview#centralized_config_management

Service Mesh Dashboard

- Provides critical, service-level metrics on three of the four golden signals of monitoring: latency, traffic, and errors.
- Define, review, and set alerts against service level objectives (SLOs)
- Secure traffic between services, even across clusters
- Provides detailed information about the endpoints for each service
    - View the traffic flowing between services, and performance for each
- A service topology graph is available that displays all mesh's services and their relationships.

Google Cloud

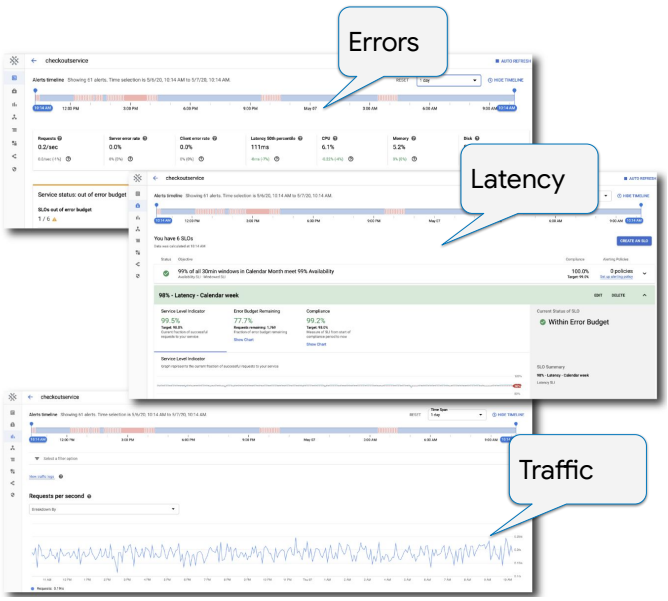
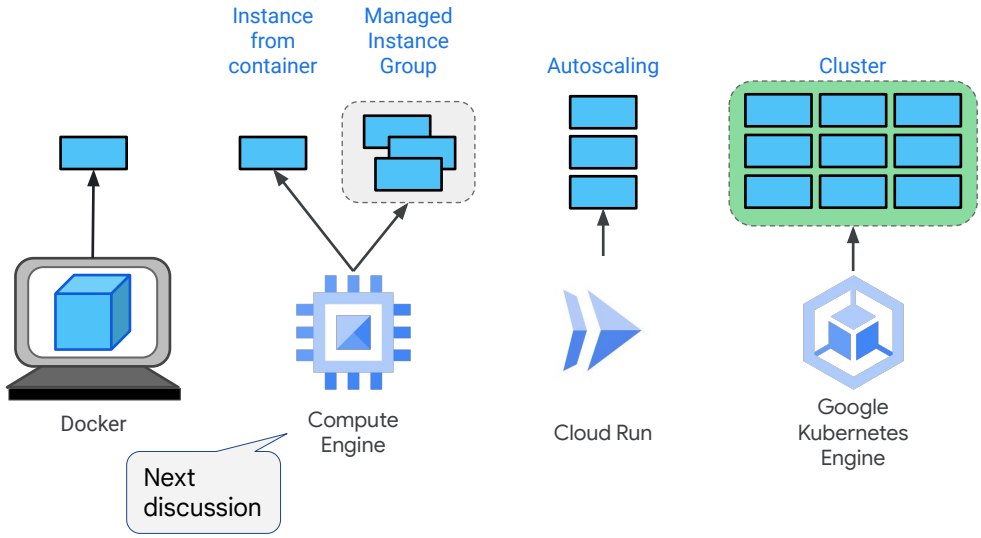


Image source:
https://cloud.google.com/anthos/docs/concepts/overview#service_mesh_dashboard

Video #6 in the Anthos 101 learning series - Service Mesh
https://www.youtube.com/watch?v=EDcy3KwV22o&t=294s

Where can you run containers?

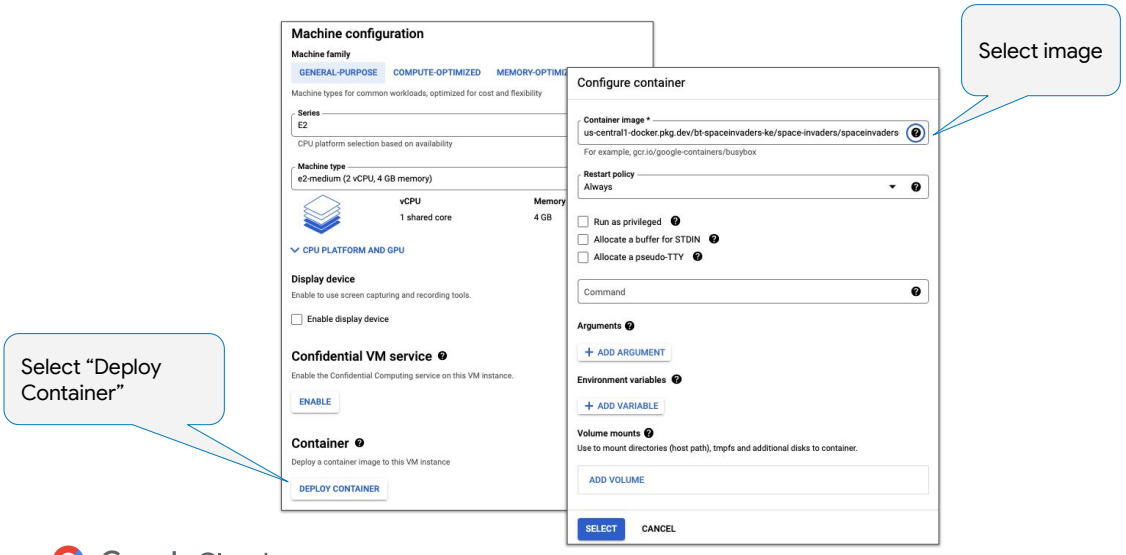


Run containers directly on Compute Engine using a Container Optimized OS

- Simplifies app deployment for your VM infrastructure
  - Use familiar tools such as the Google Cloud CLI or the Compute Engine API to manage VMs
  - Create scalable services using managed instance groups (MIGs) running containers
    - Provides autoscaling, autohealing, rolling updates, multi-zone deployments, and backends for load balancing
- Example use case
  - Lift and shift from on-premise VMs running a containerized application
- Limitations
  - Each VM instance is limited to one container
  - Consider Google Kubernetes Engine if need to deploy multiple containers per VM instance

Deploying containers on VMs and MIGs
https://cloud.google.com/compute/docs/containers/deploying-containers

Compute Engine - Deploying a container



Google Cloud

Containers on Compute Engine

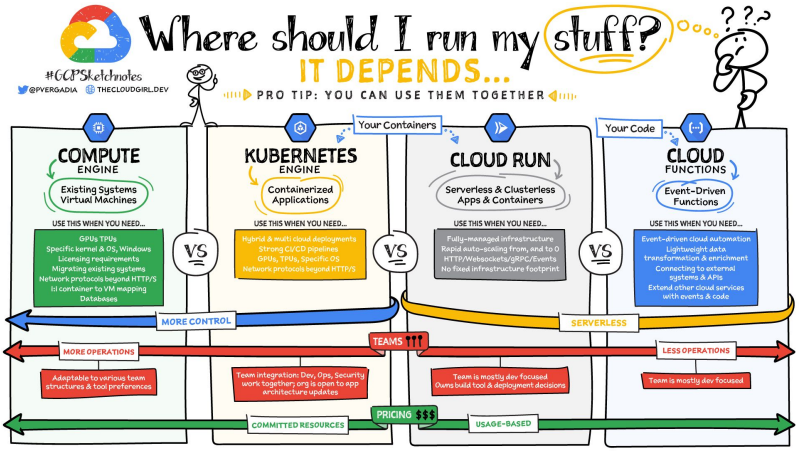
Containers on Compute Engine
https://cloud.google.com/compute/docs/containers

Limitation: only 1 container per VM:
https://cloud.google.com/compute/docs/containers/deploying-containers#limitations

To run more than one container per VM, use Google Kubernetes Engine

Compute Options

Have discussed all in this list except for Cloud Functions



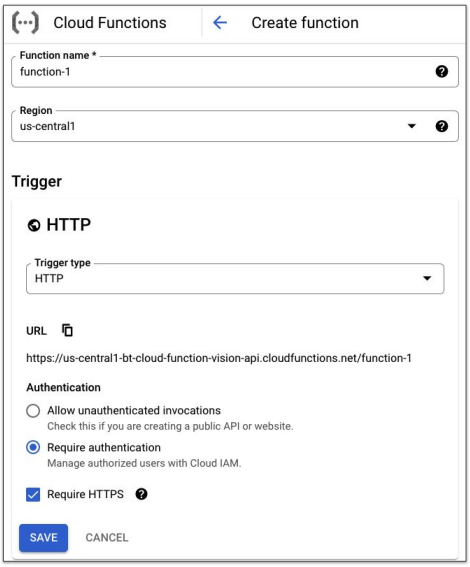
[Where should I run my stuff? Choosing a Google Cloud compute option]

Google Cloud

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

Google Cloud

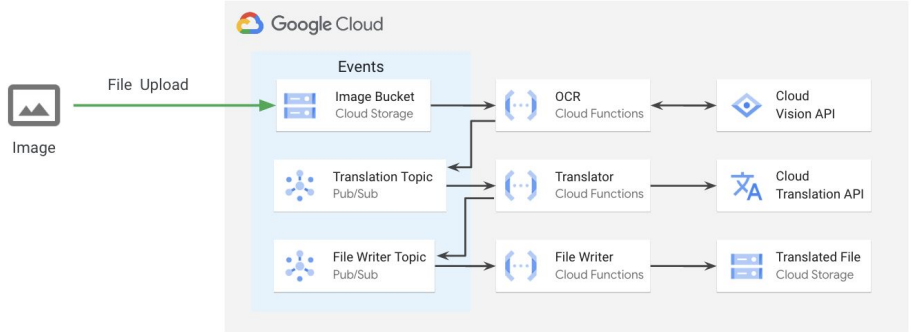


Cloud Functions
https://cloud.google.com/functions

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

Useful when need event-driven, highly scalable microservices

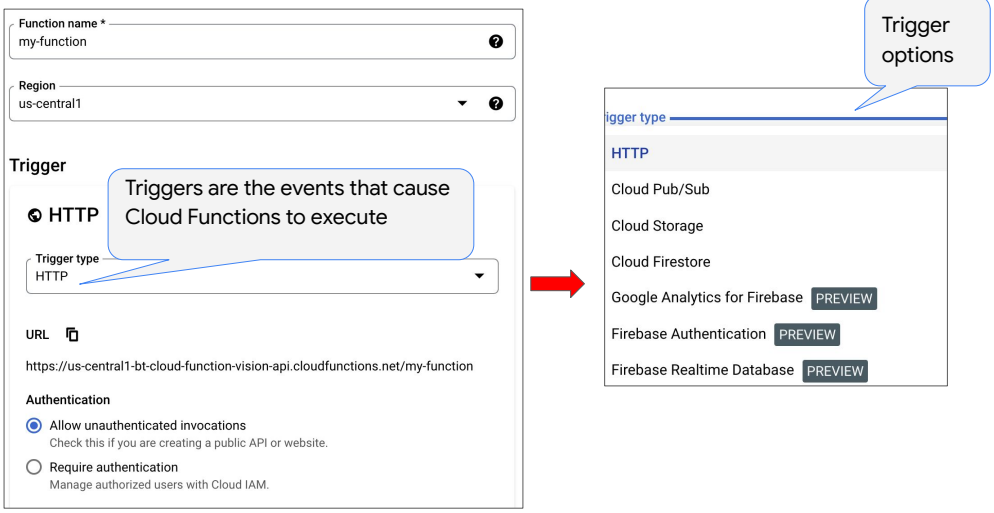
- Can be triggered by changes in a storage bucket, Pub/Sub messages, web requests and other types of events
- Completely managed, scalable, and inexpensive



Google Cloud

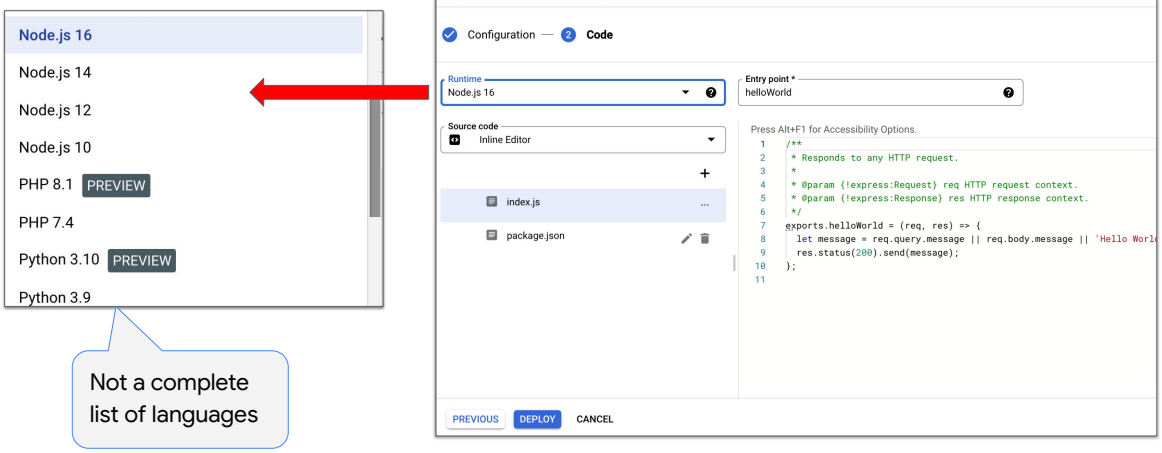
Tutorial: https://cloud.google.com/functions/docs/tutorials/ocr

Creating Cloud Functions



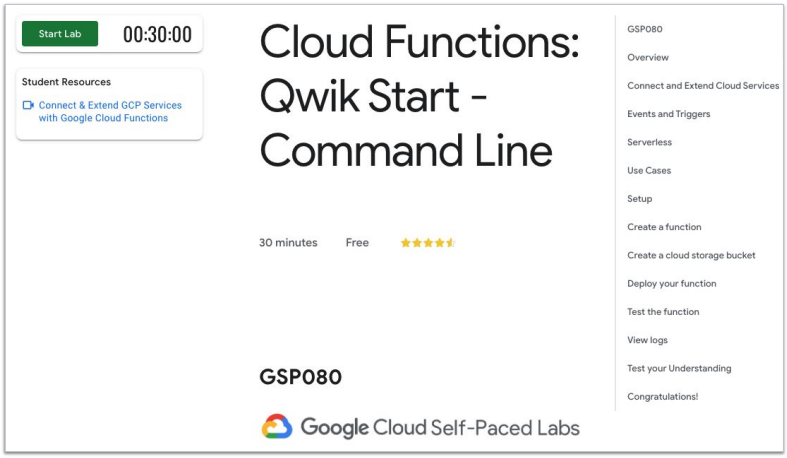
Google Cloud

Creating Cloud Functions



Google Cloud

Suggested Lab (if time allows)



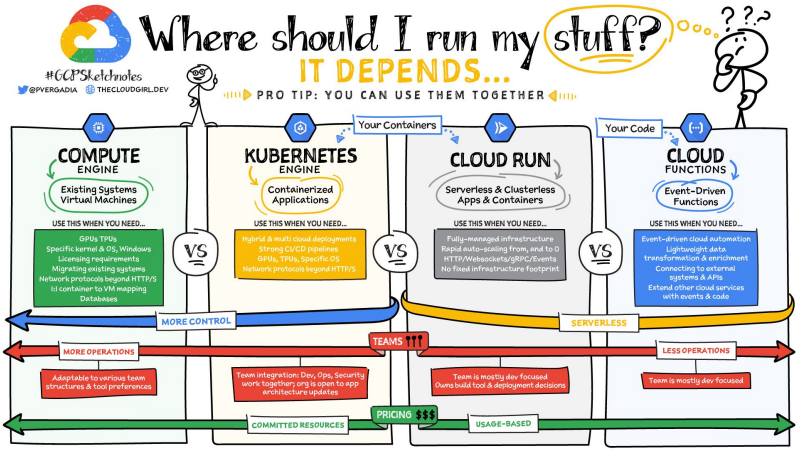
https://partner.cloudskillsboost.google/catalog_lab/924

Google Cloud

https://partner.cloudskillsboost.google/catalog_lab/924

Compute Options

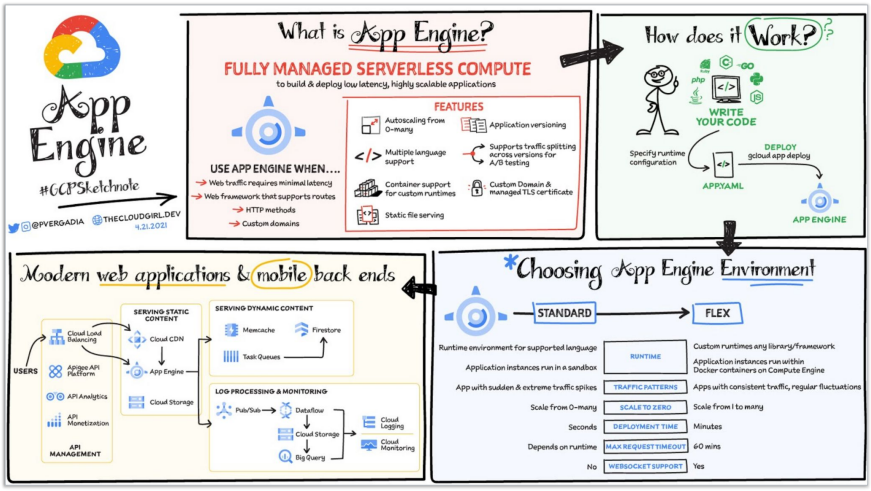
What happened to App Engine?



Google Cloud

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

What is App Engine?



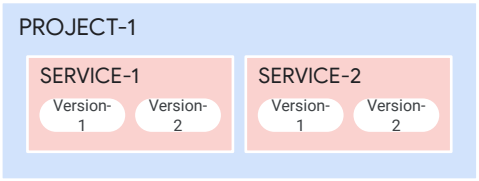
[The ultimate App Engine cheat sheet]

Google Cloud

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

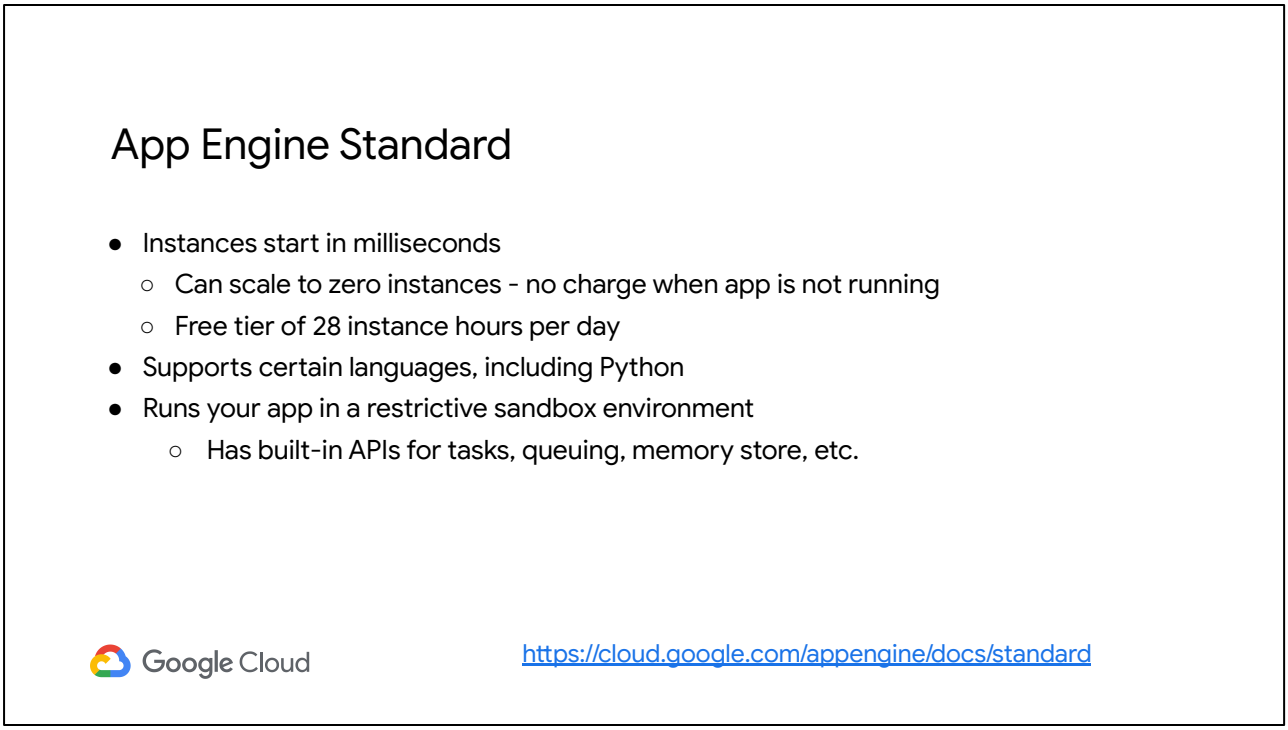
- Developers focus on code, Google manages infrastructure
- Supports Node.js, Java, Ruby, C#, Go, Python, or PHP
- Autoscales automatically depending on load
- 2 Types
  - App Engine Standard
    - Limited language support
    - Free tier
  - App Engine Flexible
    - Google Cloud builds a Docker container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



Google Cloud

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

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
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

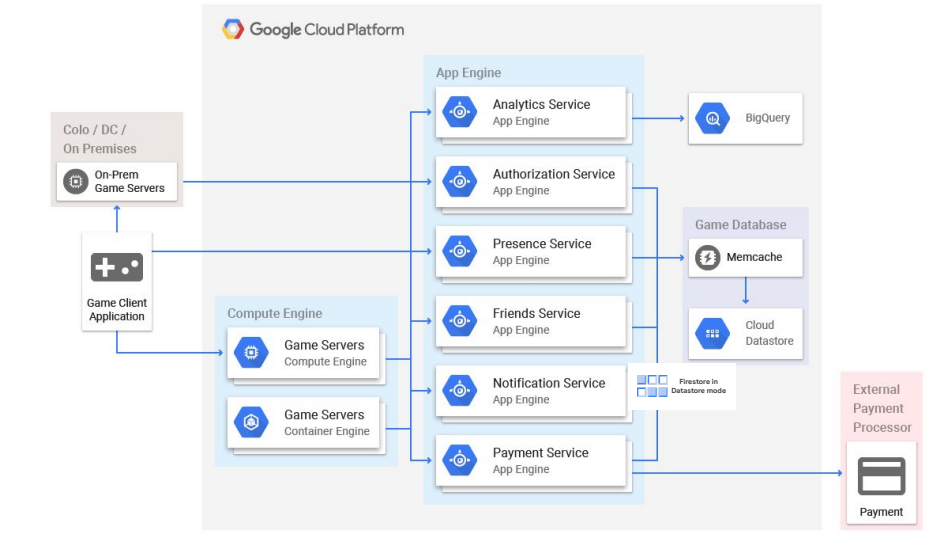
*Check the documentation for a complete list of supported programming languages

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



Google Cloud

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.

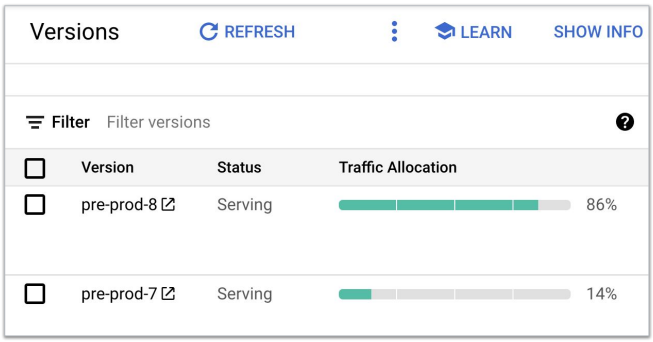
Game Database: Cloud Firestore in Datastore mode along with a dedicated Memcache can provide a fast, reliable, scalable App Engine native NoSQL database. This database pattern has been proven to scale seamless for games that started out serving thousands and ended up serving millions, such as Pokemon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

Choosing an App Engine environment

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

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

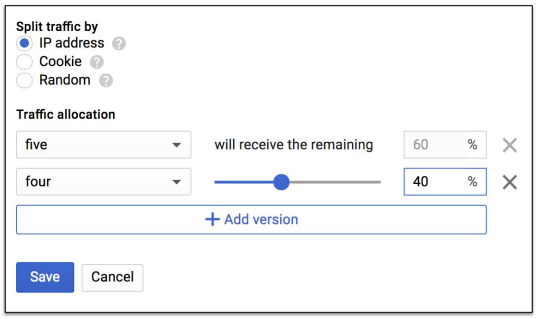


Google Cloud

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.

A/B Testing with App Engine

- A/B testing allows multiple versions of a program to run at the same time
  - Test the new version with a portion of the users
  - Compare usage and errors of the two versions
- Allows developers to get feedback and testing from real users prior to fully committing to a new version
- In App Engine, use traffic splitting for A/B testing



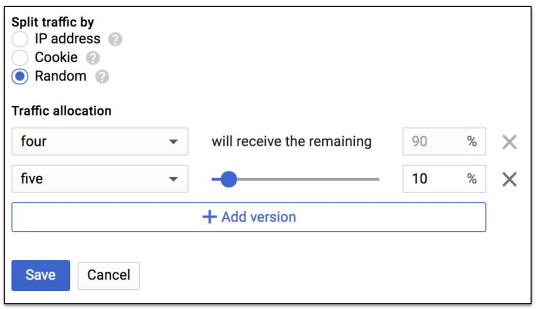
Google Cloud

A/B testing is used when you want to compare multiple versions of an app to see which is better. You can, for example, give two versions 50% of the traffic. Split by IP address, that way once a user starts getting one version they get that version for every subsequent request. You can then define some objective measure to determine which version is performing better.

You can also do A/B testing and split the traffic with a cookie. The cookie would determine which version the user gets. You might define a set of beta testers who will get the latest version. When they log in, you assign them the cookie.

Canary Deployments with App Engine

- A new version of a service is put into production alongside old versions
  - A small subset of select traffic is routed to the canary release
- Canary releases help developers know how a new version will perform in production
- Easy to rollback if they fail their testing
- Automated in App Engine

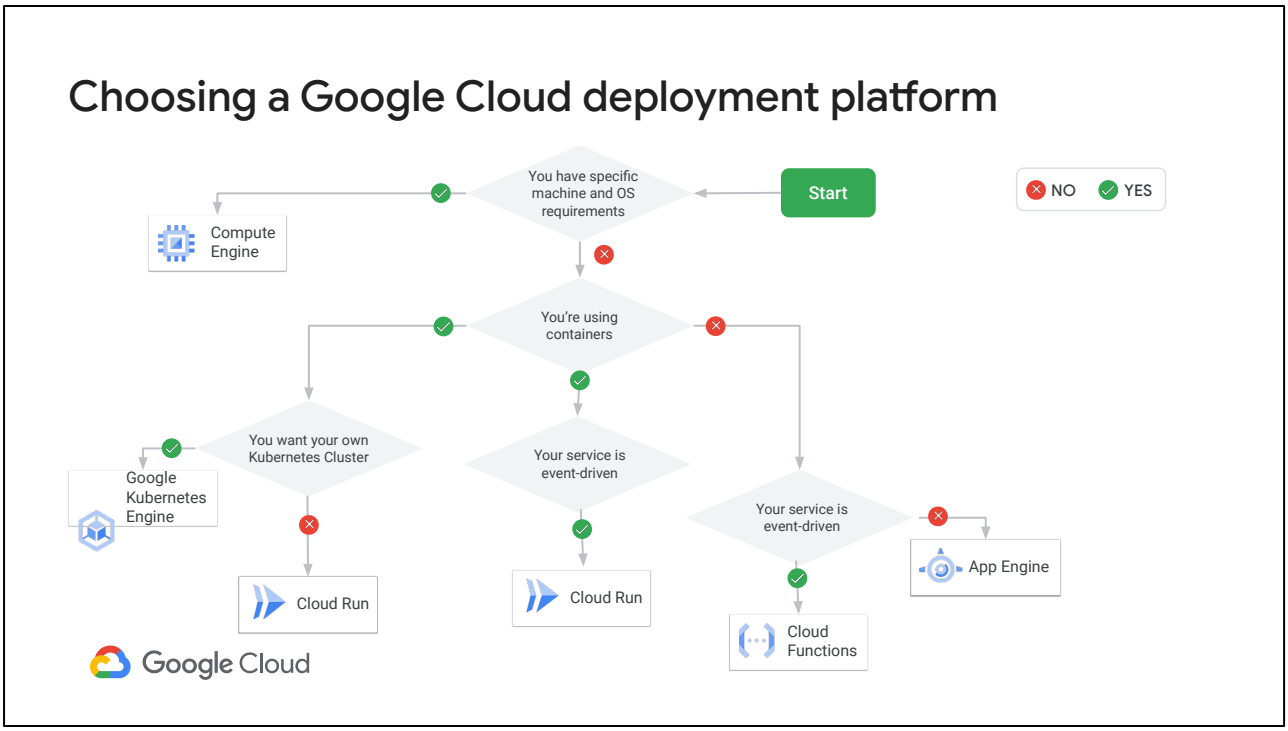


Google Cloud

A canary release is used when you create a new version but you want to only give it a small percentage of requests to test it for bugs before migrating 100% of requests to it.

For example, give a new version 10% ot the requests and split the traffic randomly. You can then monitor for errors. If there are no errors, you can gradually increase the percentage. Eventually, you can give the new version 100% of the requests.

If it turns out your new version has flaws, you can easily migrate 100% of the traffic back to the older version.



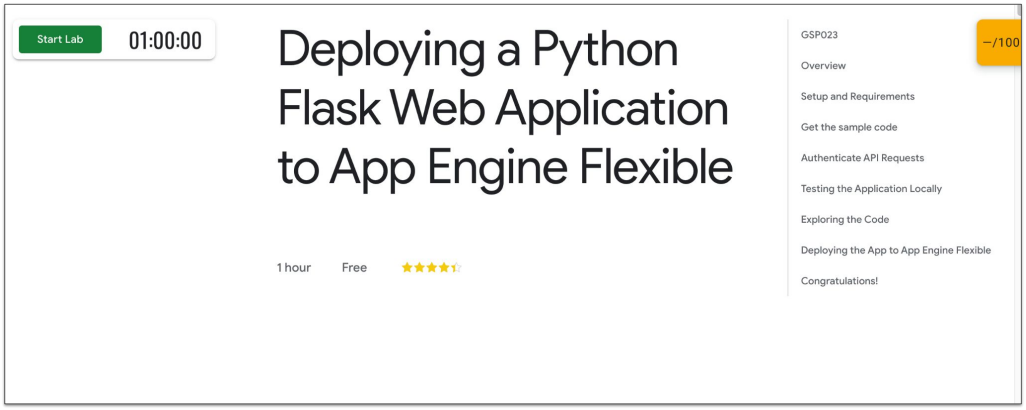
Let me give you a high-level overview of how you could decide on the most suitable platform for your application.

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.

Suggested lab (if time allows)



https://partner.cloudskillsboost.google/catalog_lab/1161

Google Cloud


Google Cloud