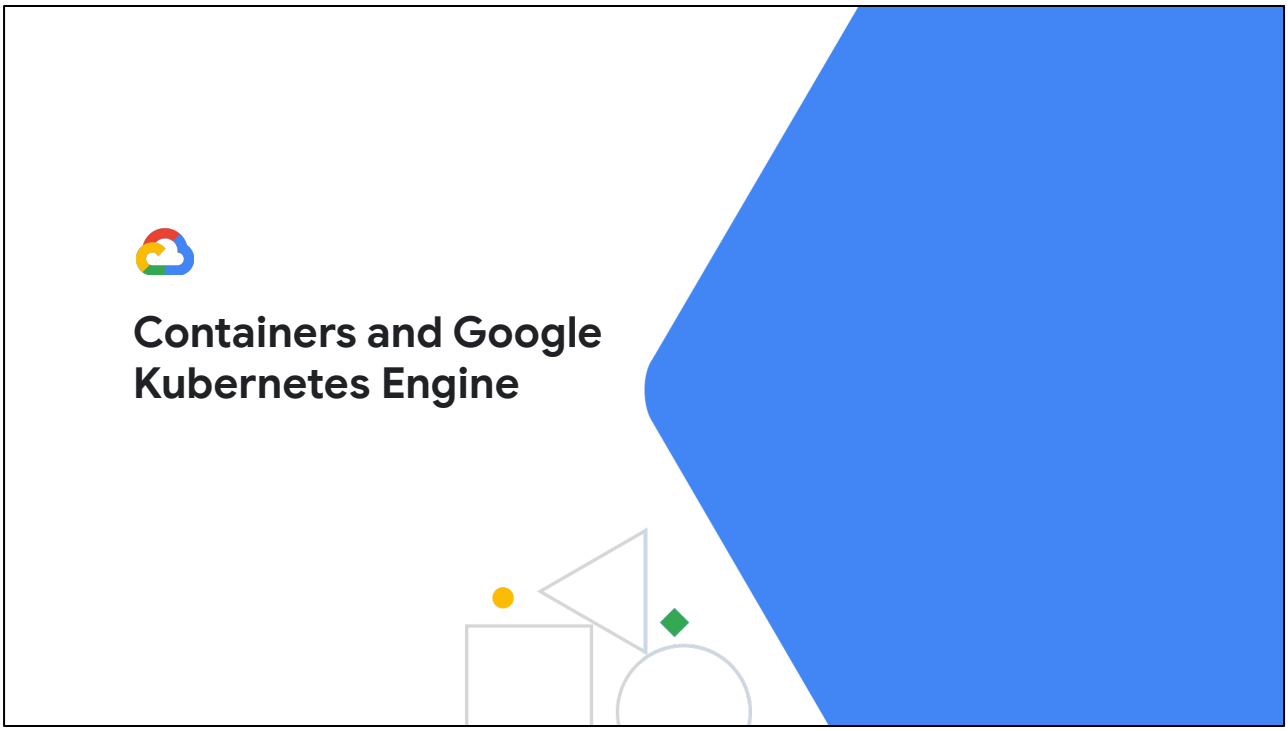
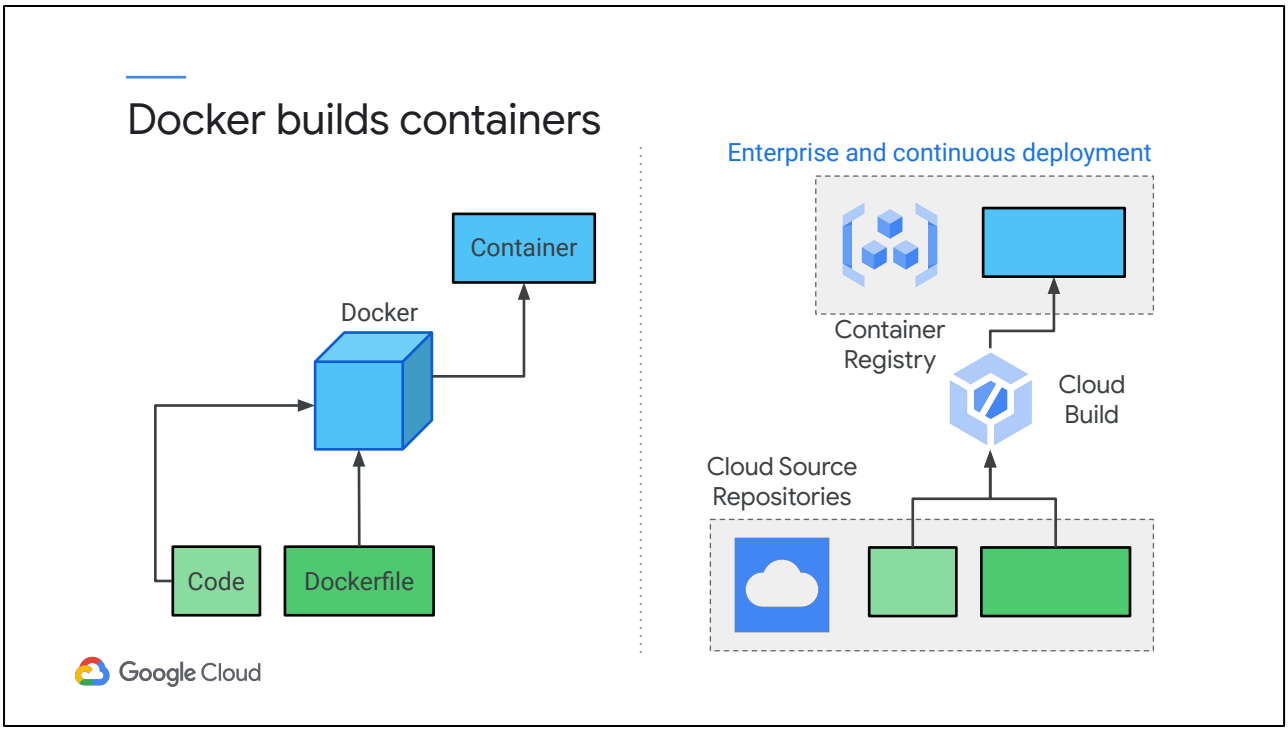


Between 2017 and 2018 the number of organizations using containers for software development and to deploy their services doubled. And the trend shows no signs of slowing. For this reason, container knowledge and skill with Kubernetes is increasingly important for the job of a Cloud Architect. And, of course, if you need more of these skills for the job, you will also need them to prepare for the exam.



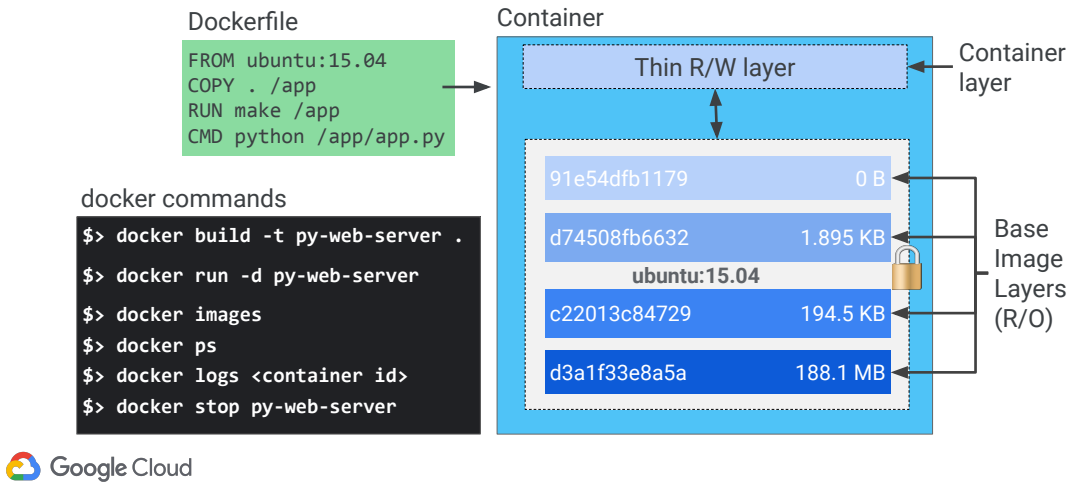
Docker is software that builds containers. You supply application code, and instructions, called a Dockerfile, and Docker follows the instructions and assembles the code and dependencies into the container. Containers can be "run", much as an application can run. However, it is a self-contained environment that can run on many platforms.

Google Cloud offers a service called Cloud Build which functions similarly to Docker. It accepts code and configuration and builds containers. 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 it is designed to scale to handle many application developers working on and continuously updating a live global service.

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. Cloud Source Repositories is a cloud-based solution.

If you were deploying hundreds of containers you would not being keeping it to yourself. One of the reasons to use containers is to share them with others. So you need a way to manage and share them. And this is the purpose of Container Registry. Container Registry has various integrations with Continuous Integration / Continuous Deployment services.

What is really in a container...



A docker container is an image built in layers. Each layer is created by an instruction in the dockerfile.

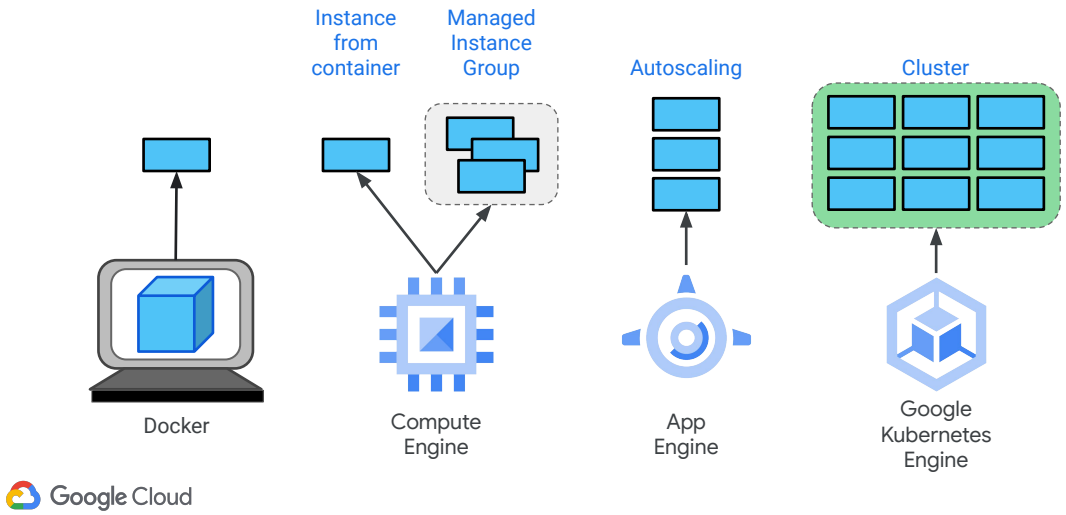
All the layers except for the top one are locked. The thin read/write layer at the top is where you can make changes to a running container. For example, if you needed to change a file, those changes would be written here.

The layered design inside of a container isolates functions. This is what makes the container stable and portable.

Here are a few of the common docker commands.

- The docker build command creates the container image.
- The docker run command runs the container.
- There are other docker commands that can help you list images, check the status of a running container, work with logs, or stop a running container.

Where can you run containers?



You can run a container in Docker itself, as you saw with the "docker run" command.

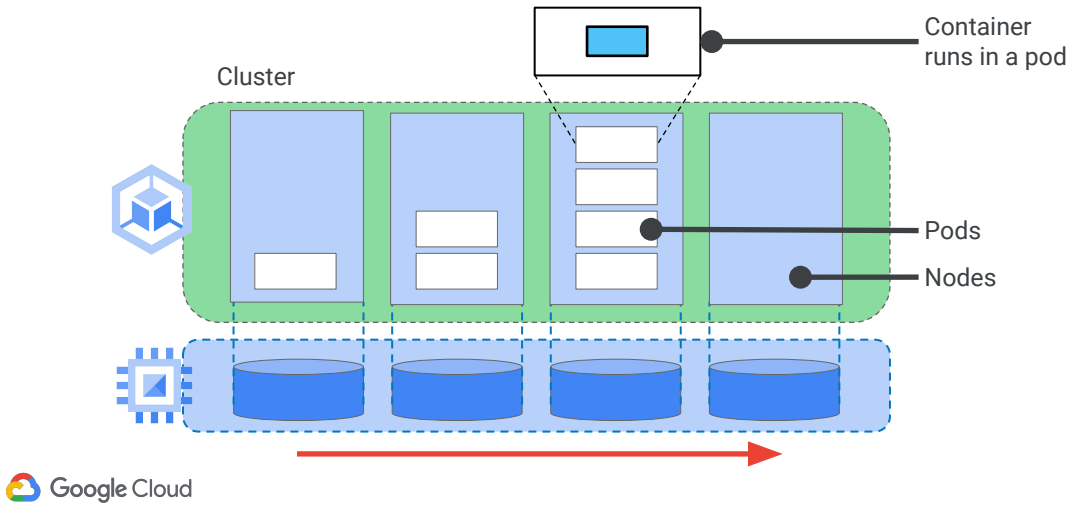
You can also run containers using Compute Engine. Compute Engine gives you the alternative to start up a virtual machine from a container rather than from an OS Image boot disk. You also have this option when creating an instance template, which means you can create Managed Instance Groups from containers.

App Engine supports containers as custom runtimes. The main difference between the App Engine standard environment and the App Engine flexible environment is that the flexible environment hosts applications in Docker containers. It creates Docker containers and persists them in Container Registry.

A Container Orchestrator is a full service for managing, running, and monitoring containers. Both App Engine flexible environment and Google Kubernetes Engine are Container Orchestrators.

Kubernetes is open standard software, so you can run a Kubernetes cluster in your data center. Google Kubernetes Engine provides Kubernetes as a managed service.

Kubernetes cluster has nodes, pods, and containers



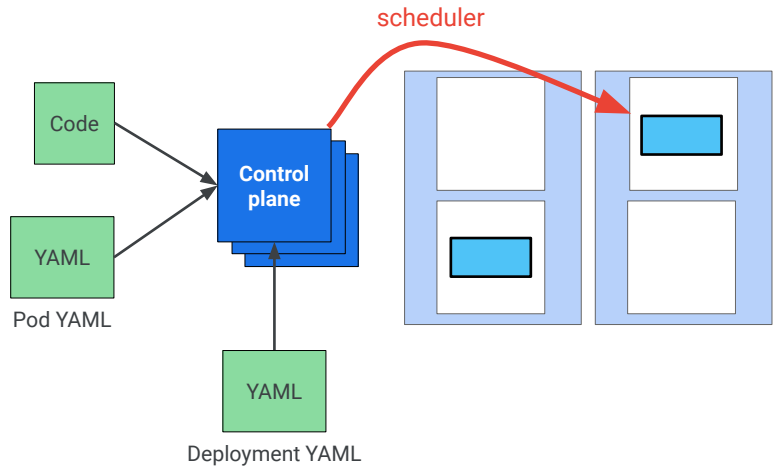
A Kubernetes cluster is composed of nodes, which are a unit of hardware resources. Nodes in GKE are implemented as VMs in Compute Engine. Each node has pods. Pods are resource management units. A pod is how Kubernetes controls and manages resources needed by applications and how it executes code. Pods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

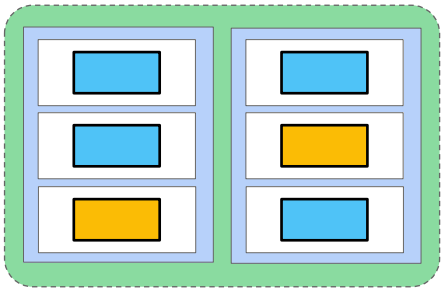


Google Cloud

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.

Advanced operations: A/B testing, rolling updates



Google Cloud

One of the advanced things that Kubernetes deployments allow you to do is roll out software to some pods and not others. So you can actually keep version A in production on most of the pods and try out version B with a sample group in other pods. This is called A/B testing and it is great because you can test the new software in the real production environment without risking the integrity of the entire service.

Another thing you can do with deployments is a rolling update. Basically, you load up the new software in a replacement pod, switch the load to the new pod, and turn down the old one. This allows you to perform a controlled and gradual roll-out of the new software across the service. If something goes wrong, you can detect the problem and roll back to the previous software.

Really, if you are going to run an enterprise production service you will need these kinds of operations. And that is one major reason to adopt Kubernetes.

There are a number of subjects that were not covered in this brief overview. For example, how containers running in the same pod can share resources, how containers running in different pods can communicate, and how networking is handled between a node's IP and the applications. These subjects and more are covered in the course "Getting Started with Google Kubernetes Engine" or you can find more information in the online documentation.