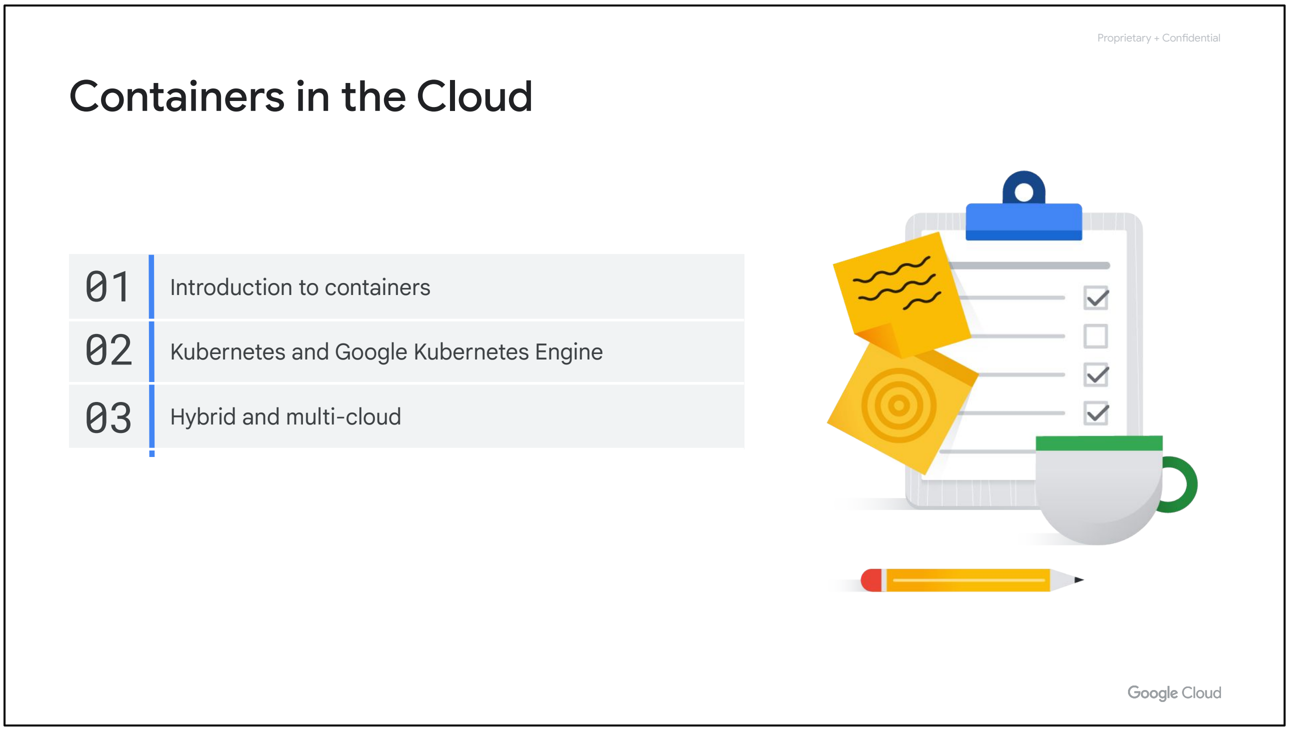
Proprietary + Confidential

05

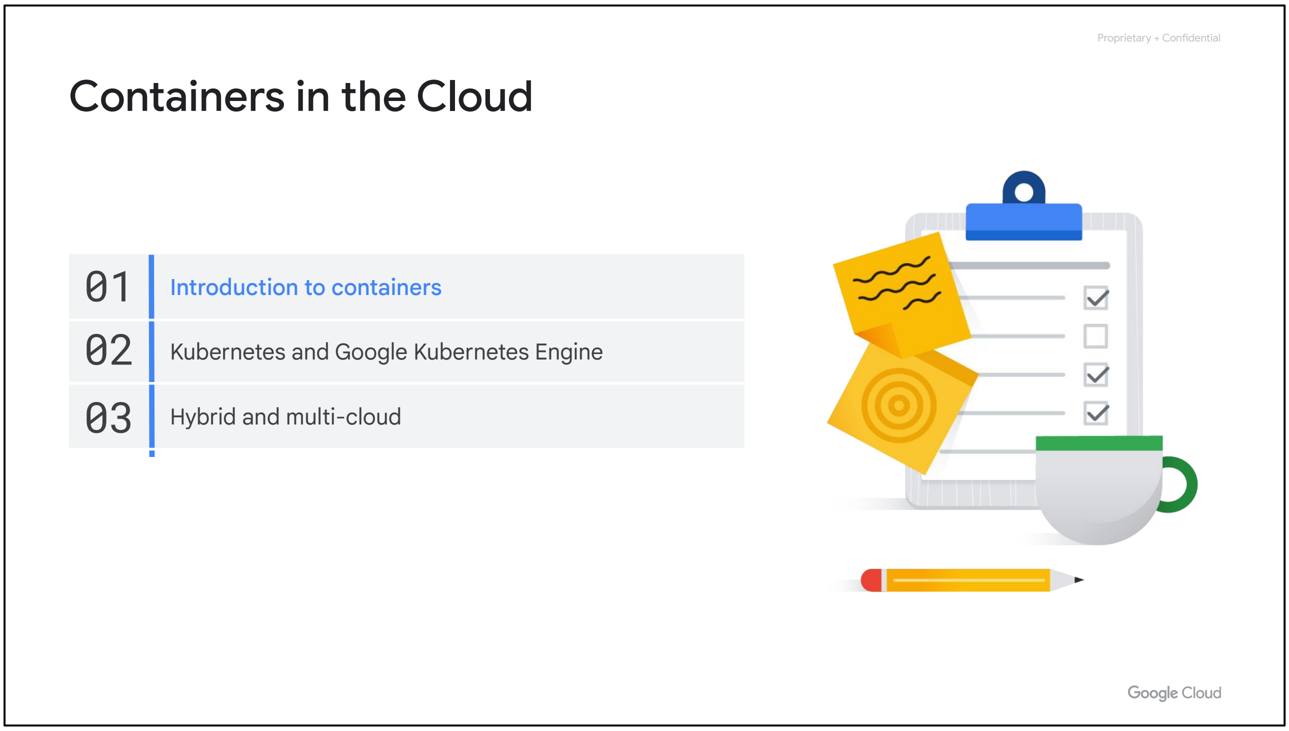
Containers in the Cloud

Google Cloud



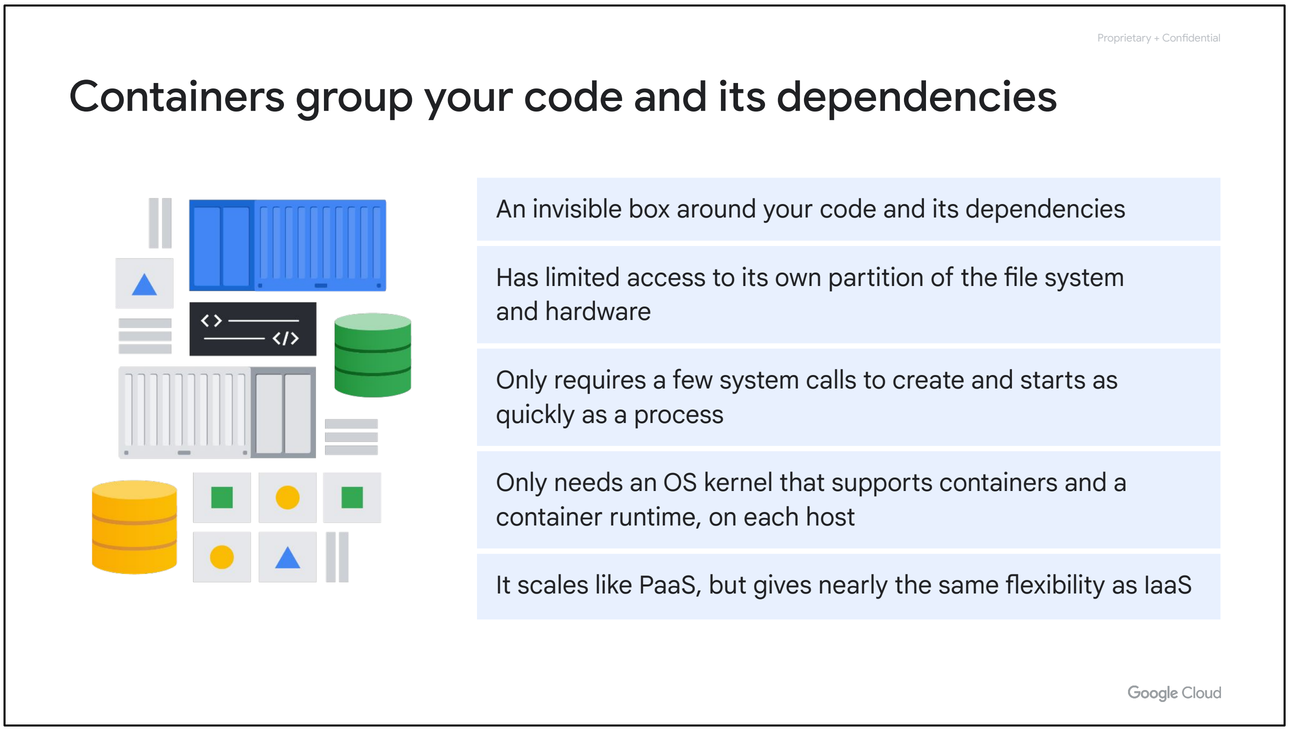
Containers in the Cloud

Now we’ll explore containers and help you understand how they are used.

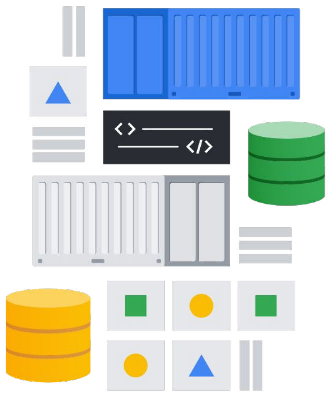


Containers in the Cloud

This will include learning about Kubernetes and Google Kubernetes Engine. We'll also introduce Anthos, Google Cloud’s platform for hybrid and multi-cloud computing.



Containers group your code and its dependencies



An invisible box around your code and its dependencies

Has limited access to its own partition of the file system and hardware

Only requires a few system calls to create and starts as quickly as a process

Only needs an OS kernel that supports containers and a container runtime, on each host

It scales like PaaS, but gives nearly the same flexibility as IaaS

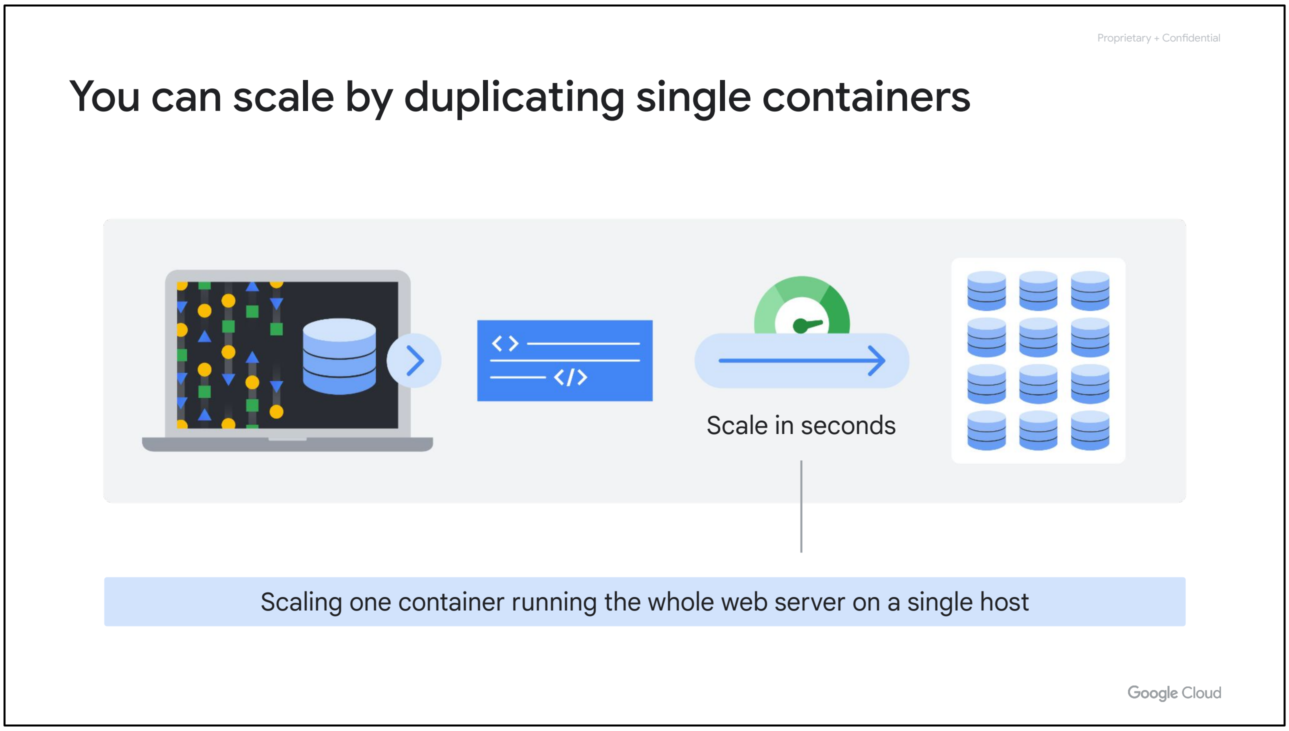
Google Cloud

What are containers?

A container is an invisible box around your code and its dependencies with limited access to its own partition of the file system and hardware. It only requires a few system calls to create and it starts as quickly as a process. All that's needed on each host is an OS kernel that supports containers and a container runtime.

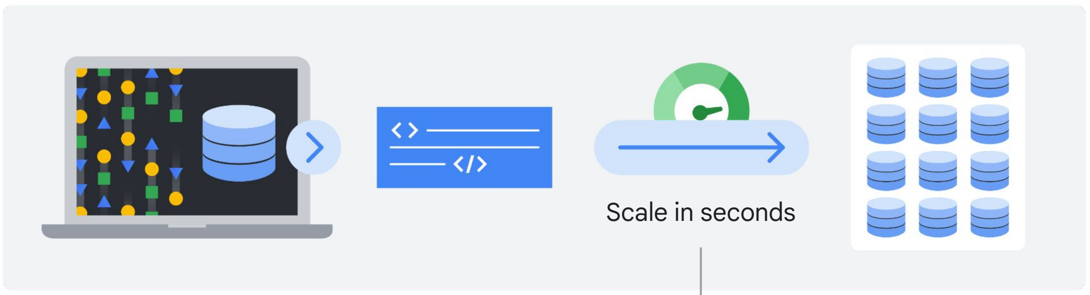
In essence, the OS and dependencies are being virtualized. A container gives you the best of two worlds - it scales like Platform as a Service (PaaS) but gives you nearly the same flexibility as Infrastructure as a Service (IaaS.) This makes your code ultra portable, and the OS and hardware can be treated as a black box. So you can go from development, to staging, to production, or from your laptop to the cloud, without changing or rebuilding anything.

In short, containers are portable, *loosely coupled* boxes of application code and dependencies that allow you to "code once, and run anywhere."



Proprietary + Confidential

You can scale by duplicating single containers

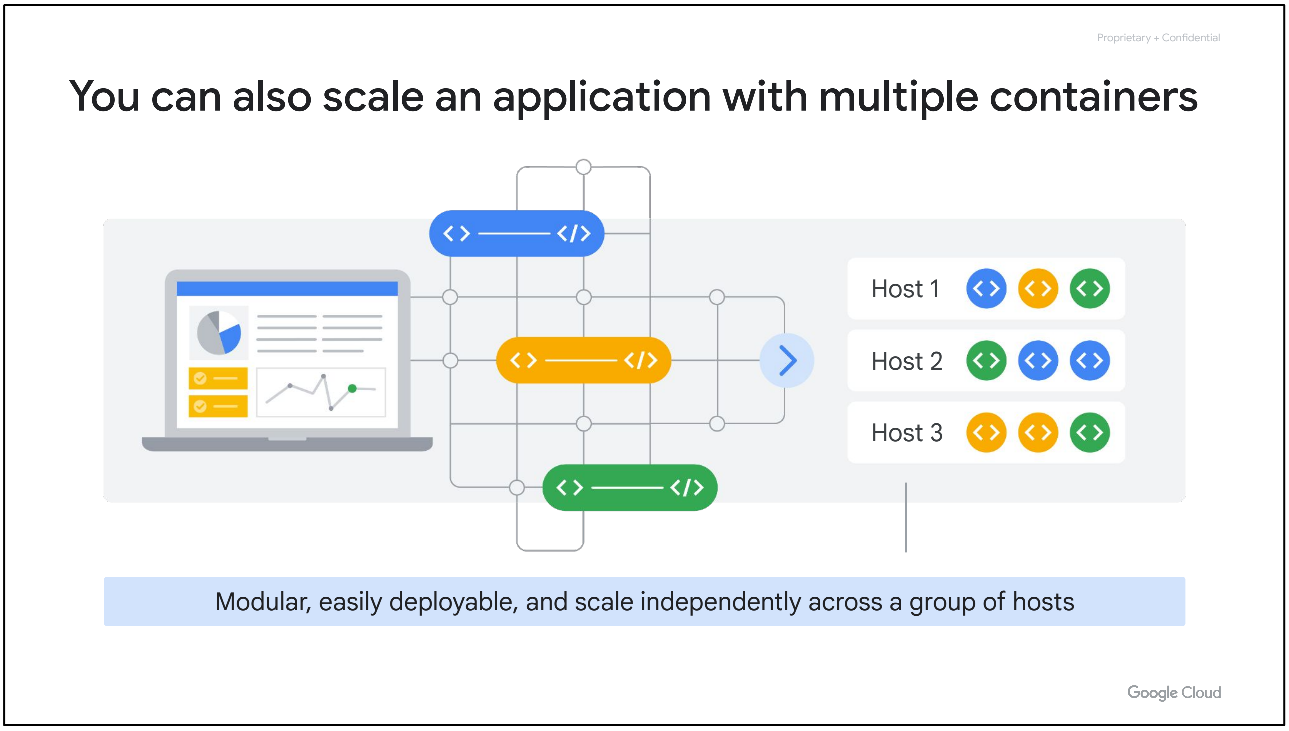


Scaling one container running the whole web server on a single host

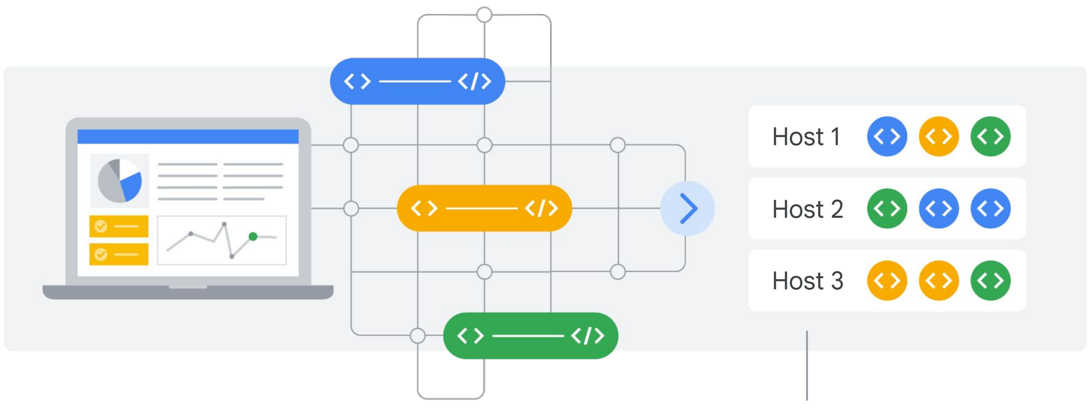
Google Cloud

As an example, let's say you want to launch and then scale a web server. With a container, you can do this in seconds and deploy dozens or hundreds of them, depending on the size or your workload, on a single host. This is because containers can be easily "scaled" to meet demand.

This is just a simple example of scaling one container which is running your whole application on a single host.



You can also scale an application with multiple containers

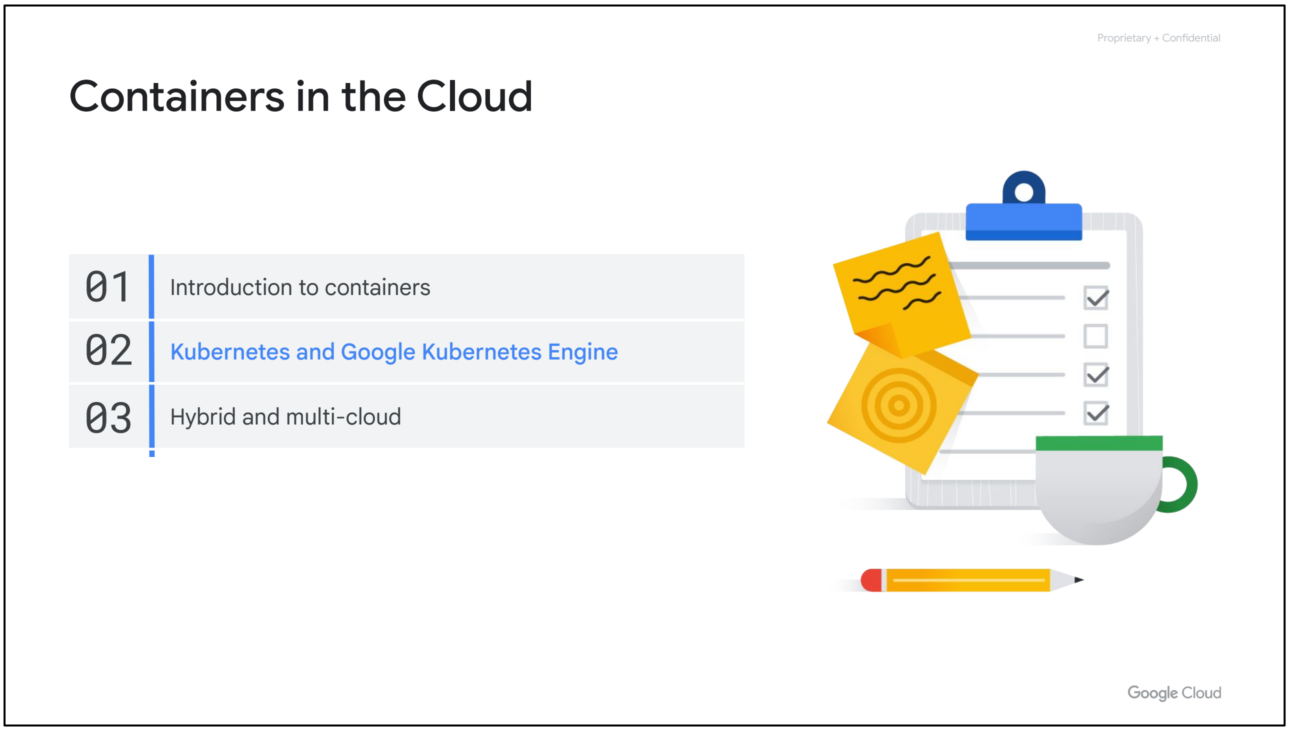


Modular, easily deployable, and scale independently across a group of hosts

However, in practice you'll probably want to build your applications using lots of containers, each performing their own function, as is done when using a microservices architecture.

If you build applications this way and connect them with network connections, you can make them modular, easily deployable, and able to be scaled independently across a group of hosts.

The hosts can scale up and down and start and stop containers as demand for your application changes, or as hosts fail.



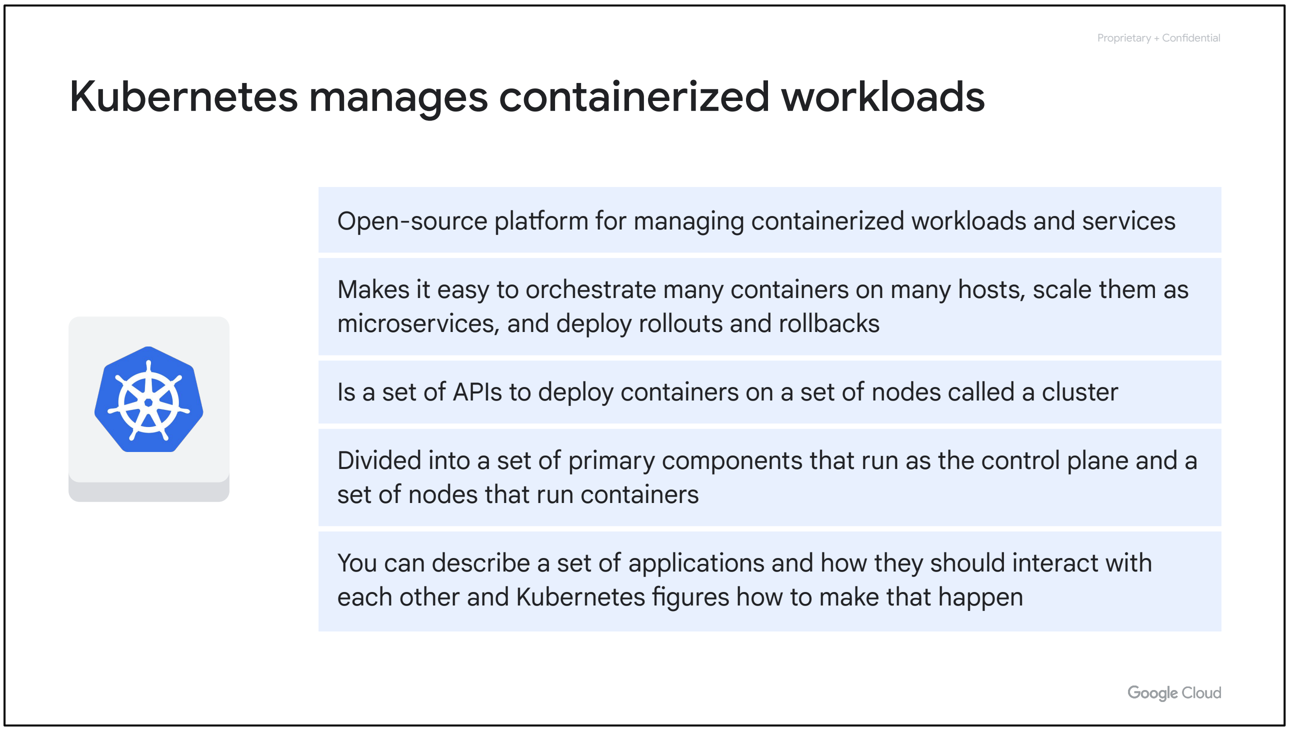
Proprietary + Confidential

Containers in the Cloud

01	Introduction to containers
02	Kubernetes and Google Kubernetes Engine
03	Hybrid and multi-cloud

Google Cloud

A product that helps manage and scale containerized applications is **Kubernetes**. So to save time and effort when scaling applications and workloads, Kubernetes can be bootstrapped using **Google Kubernetes Engine** or GKE.



Kubernetes manages containerized workloads

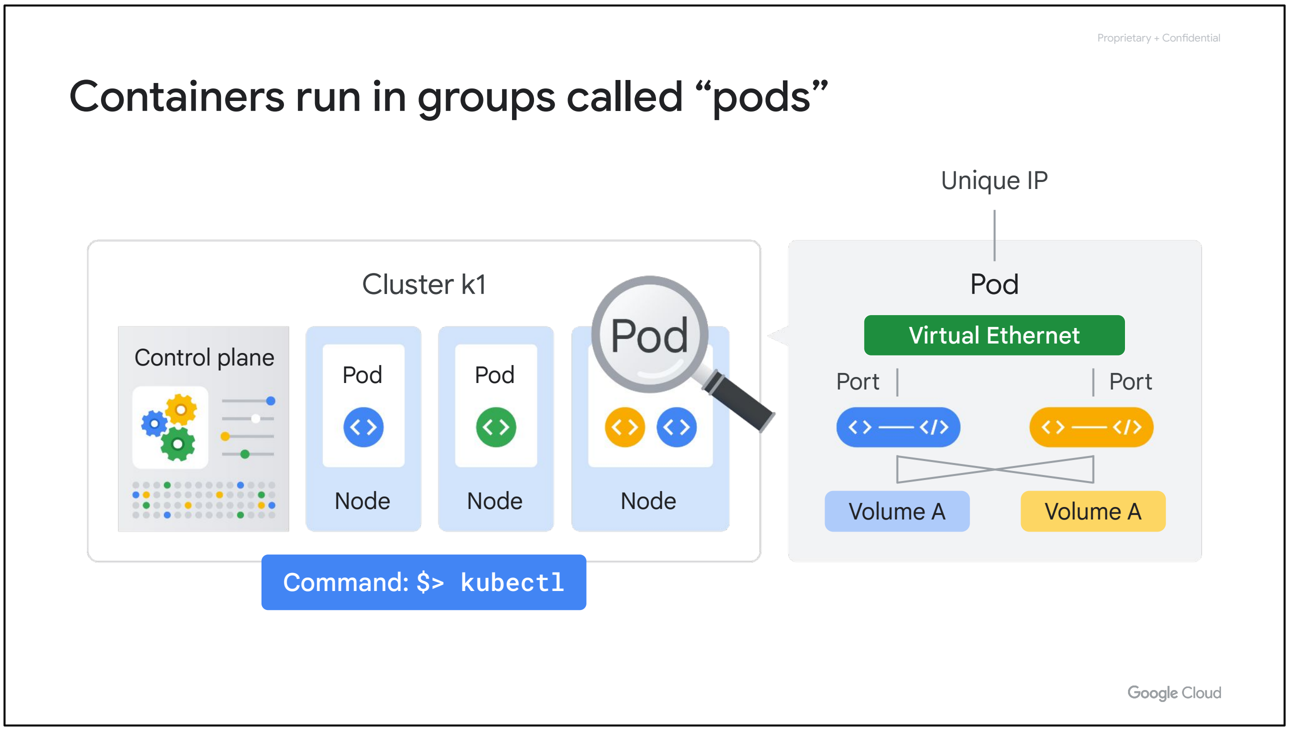
- Open-source platform for managing containerized workloads and services
- Makes it easy to orchestrate many containers on many hosts, scale them as microservices, and deploy rollouts and rollbacks
- Is a set of APIs to deploy containers on a set of nodes called a cluster
- Divided into a set of primary components that run as the control plane and a set of nodes that run containers
- You can describe a set of applications and how they should interact with each other and Kubernetes figures how to make that happen

So, what is Kubernetes? Kubernetes is an open-source platform for managing containerized workloads and services. It makes it easy to orchestrate many containers on many hosts, scale them as microservices, and easily deploy rollouts and rollbacks.

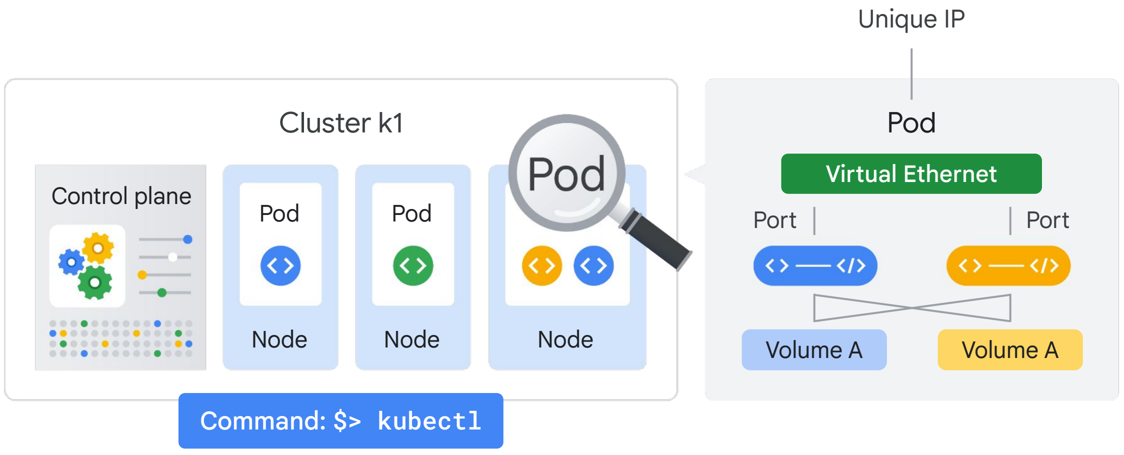
At the highest level, Kubernetes is a set of APIs that you can use to deploy containers on a set of nodes called a cluster.

The system is divided into a set of primary components that run as the control plane and a set of nodes that run containers. In Kubernetes, a node represents a computing instance, like a machine. Note that this is different to a node on Google Cloud which is a virtual machine running in Compute Engine.

You can describe a set of applications and how they should interact with each other, and Kubernetes determines how to make that happen.



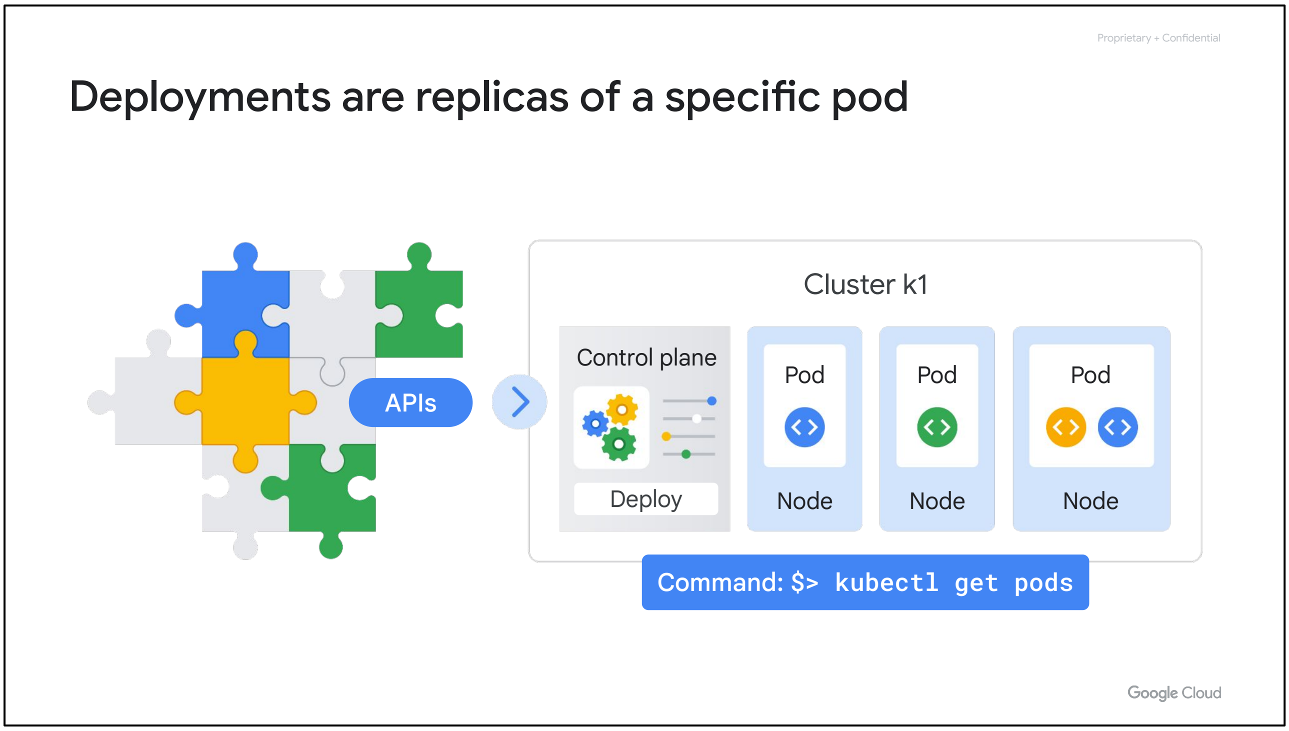
Containers run in groups called “pods”



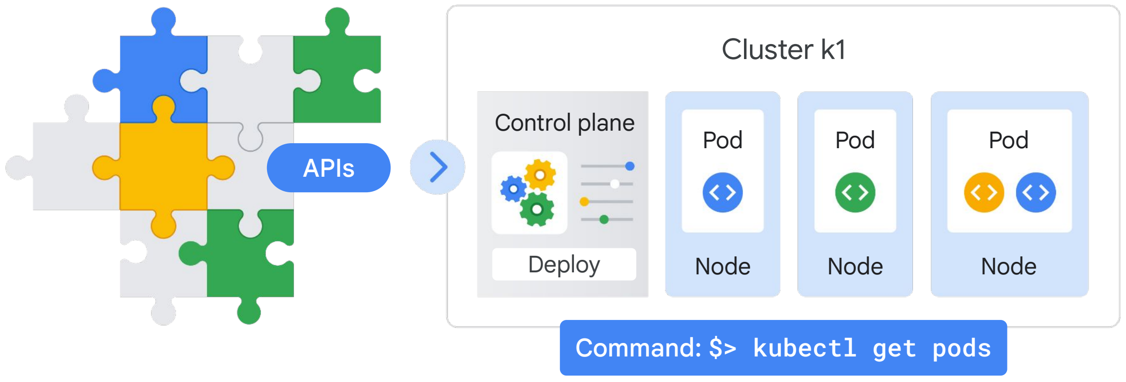
Deploying containers on nodes by using a wrapper around one or more containers is what defines a Pod. A Pod is the smallest unit in Kubernetes that you create or deploy. It represents a running process on your cluster as either a component of your application or an entire app.

Generally, you only have one container per pod, but if you have multiple containers with a hard dependency, you can package them into a single pod and share networking and storage resources between them. The Pod provides a unique network IP and set of ports for your containers and configurable options that govern how your containers should run.

One way to run a container in a Pod in Kubernetes is to use the `kubectl` run command, which starts a Deployment with a container running inside a Pod.



Deployments are replicas of a specific pod

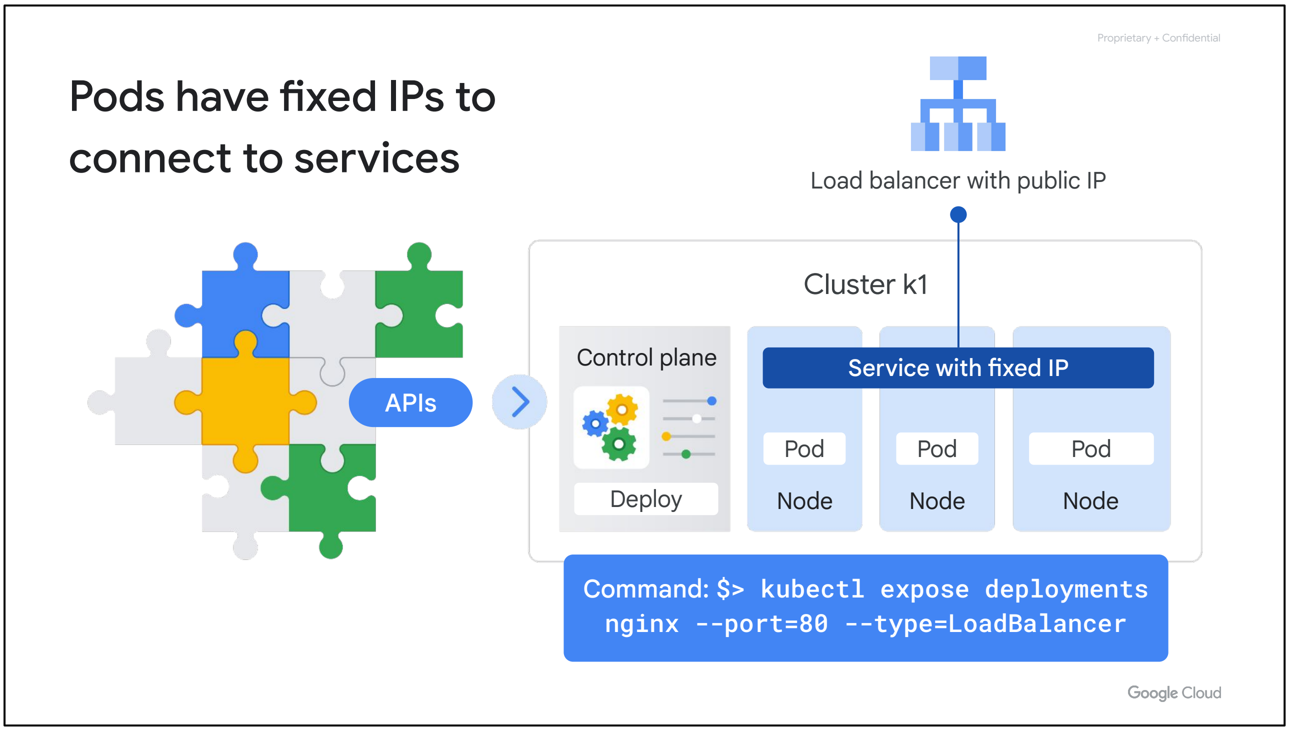


Command: $> `kubectl get pods`

A Deployment represents a group of replicas of the same Pod and keeps your Pods running even when the nodes they run on fail. A Deployment could represent a component of an application or even an entire app.

To see a list of the running Pods in your project, run the command:

```
$ kubectl get pods
```

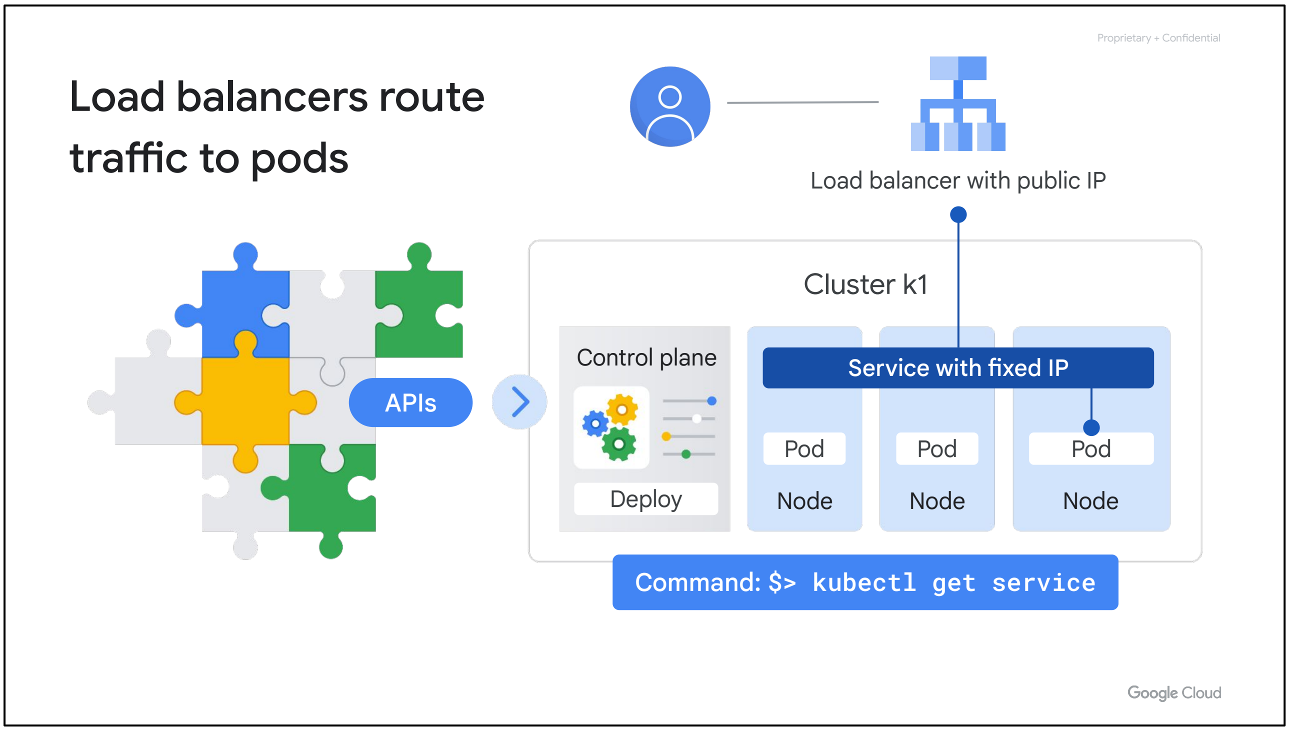


Pods have fixed IPs to connect to services

Command: $> `kubectl expose deployments nginx --port=80 --type=LoadBalancer`

Kubernetes creates a Service with a fixed IP address for your Pods, and a controller says "I need to attach an external load balancer with a public IP address to that Service so others outside the cluster can access it".

In GKE, the load balancer is created as a network load balancer.



Load balancers route traffic to pods

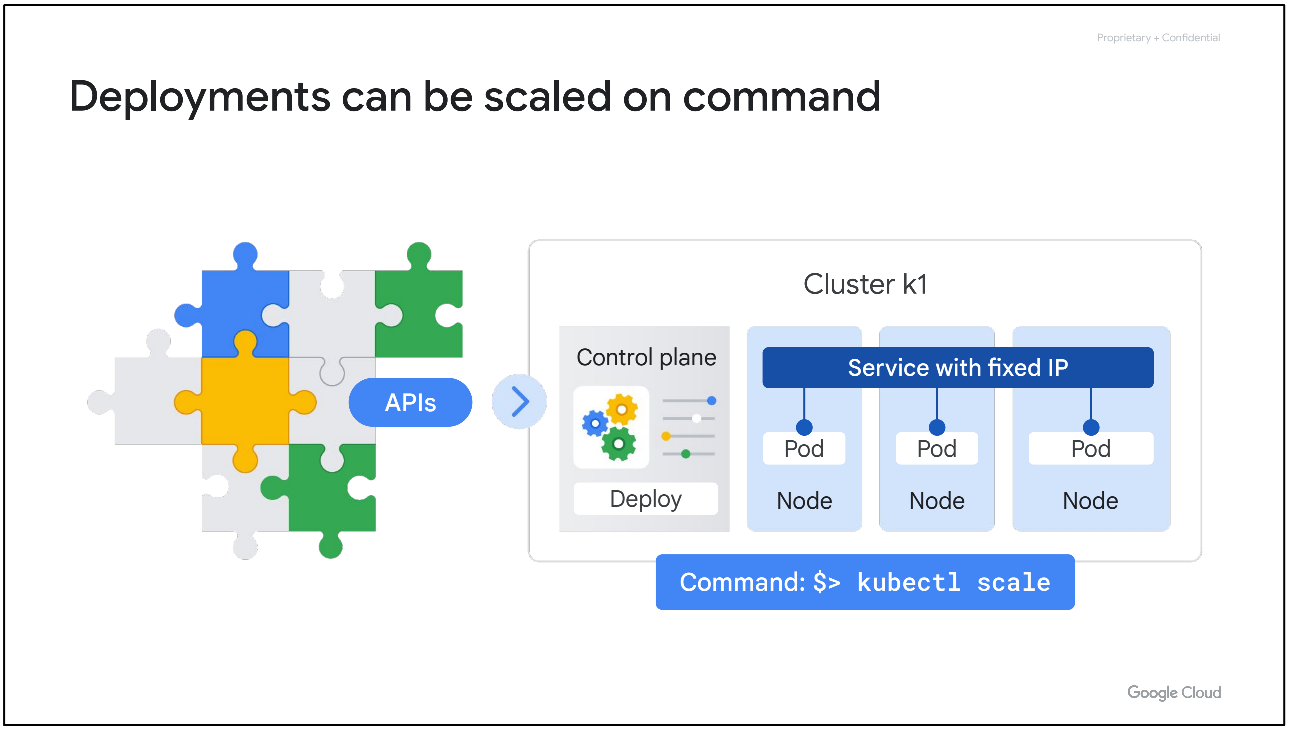
Command: `$> kubectl get service`

Any client that reaches that IP address will be routed to a Pod behind the Service. A Service is an abstraction which defines a logical set of Pods and a policy by which to access them.

As Deployments create and destroy Pods, Pods will be assigned their own IP addresses, but those addresses don't remain stable over time.

A Service group is a set of Pods and provides a stable endpoint (or fixed IP address) for them.

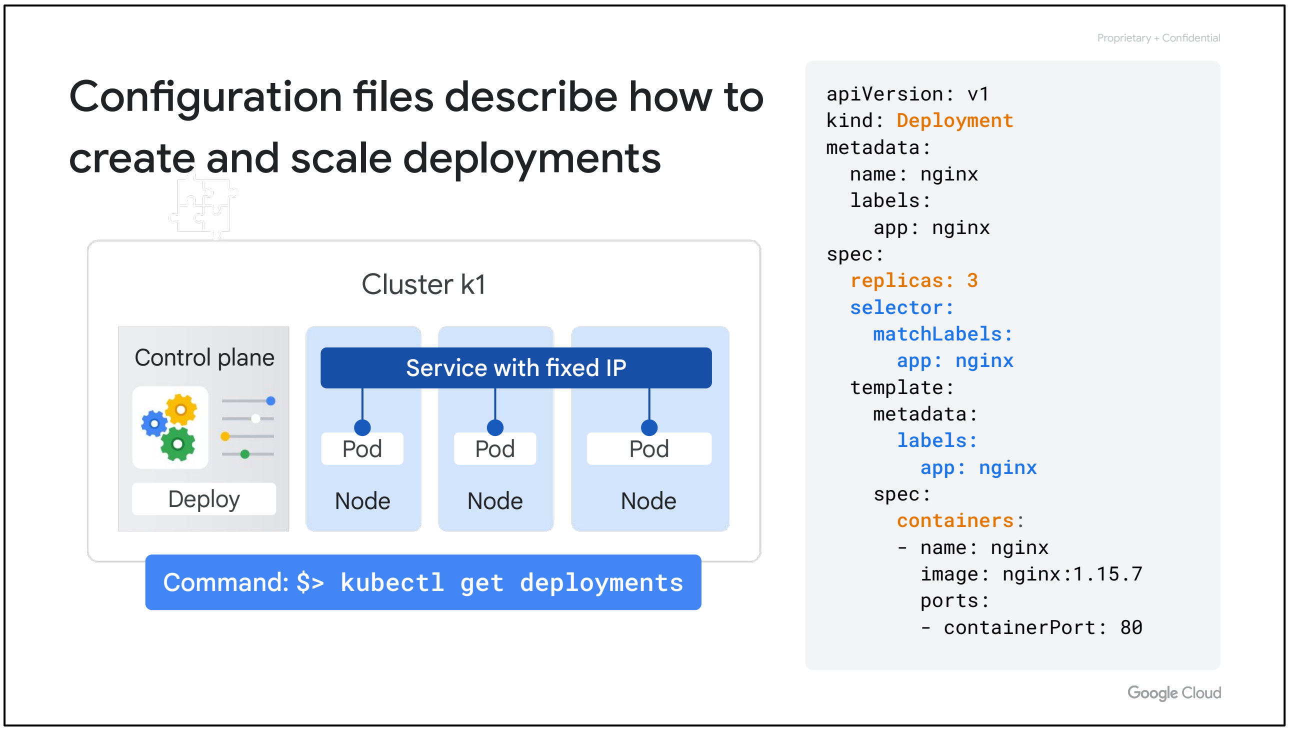
For example, if you create two sets of Pods called `frontend` and `backend` and put them behind their own Services, the `backend` Pods might change, but `frontend` Pods are not aware of this. They simply refer to the backend Service.


Proprietary + Confidential
Deployments can be scaled on command
APIs
Cluster k1
Control plane
Deploy
Service with fixed IP
Pod
Pod
Pod
Node
Node
Node
Command: $> kubectl scale
Google Cloud

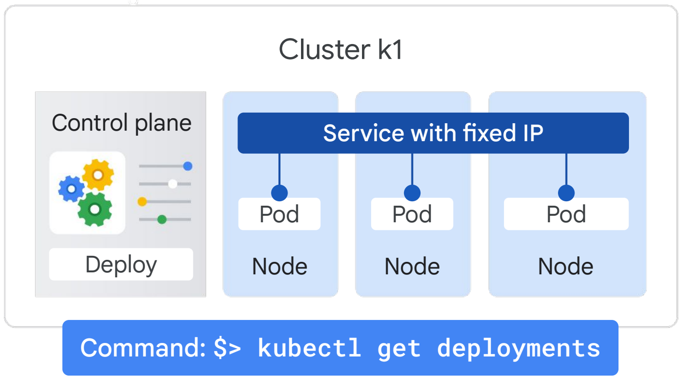
To scale a Deployment, run the `kubectl scale` command.

In this example, three Pods are created in your Deployment, and they're placed behind the Service and share one fixed IP address.

You could also use autoscaling with other kinds of parameters; for example, you can specify that the number of pods should increase when CPU utilization reaches a certain limit.



Configuration files describe how to create and scale deployments



```
apiVersion: v1
kind: Deployment
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.15.7
        ports:
        - containerPort: 80
```

Google Cloud

So far, we've seen how to run imperative commands like expose and scale. This works well to learn and test Kubernetes step-by-step.

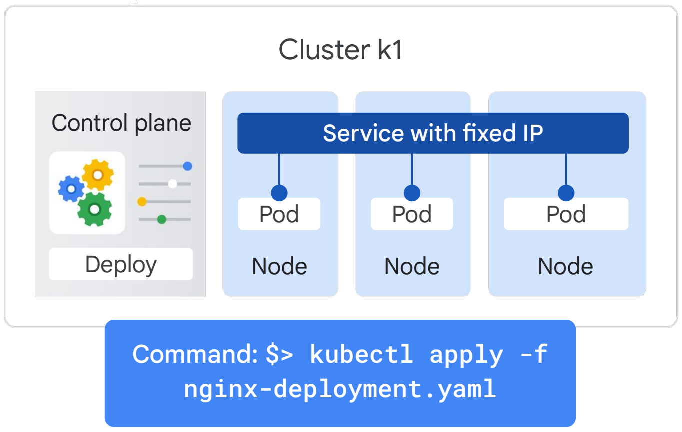
But the real strength of Kubernetes comes when you work in a declarative way.

Instead of issuing commands, you provide a configuration file that tells Kubernetes what you want your desired state to look like, and Kubernetes determines how to do it.

You accomplish this by using a Deployment config file.

To get this file, you can run a `kubectl get deployments` command, and you'll get a Deployment configuration file that looks like this.

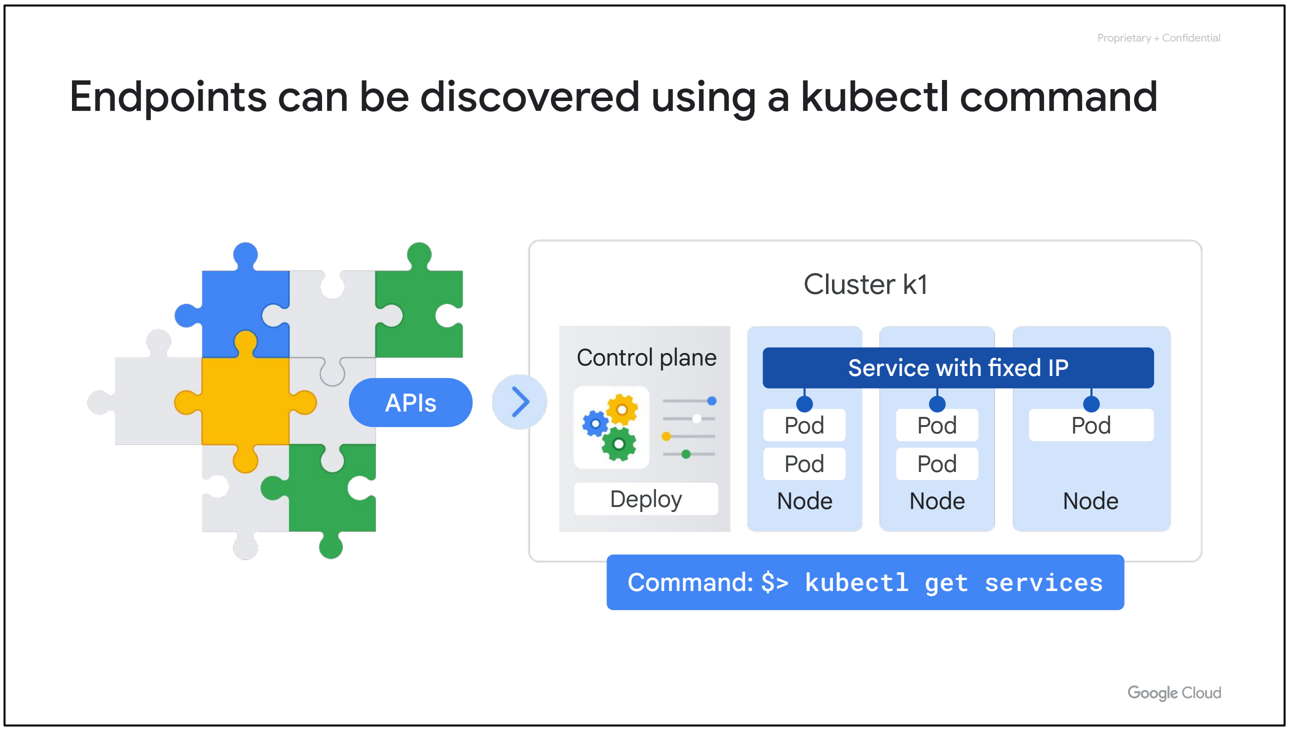
Deployments can be modified by changing configuration files



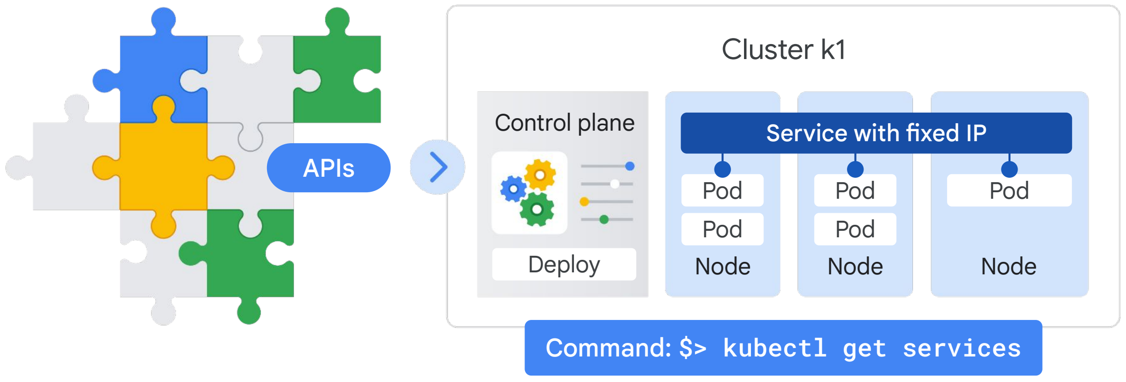
Command: `$> kubectl apply -f nginx-deployment.yaml`

```
apiVersion: v1
kind: Deployment
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  replicas: 5
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.10.0
        ports:
        - containerPort: 80
```

You can check your Deployment to make sure the proper number of replicas is running by using either `kubectl get deployments` or `kubectl describe deployments`. To run five replicas instead of three, all you do is update the Deployment config file and run the `kubectl apply` command to use the updated config file.

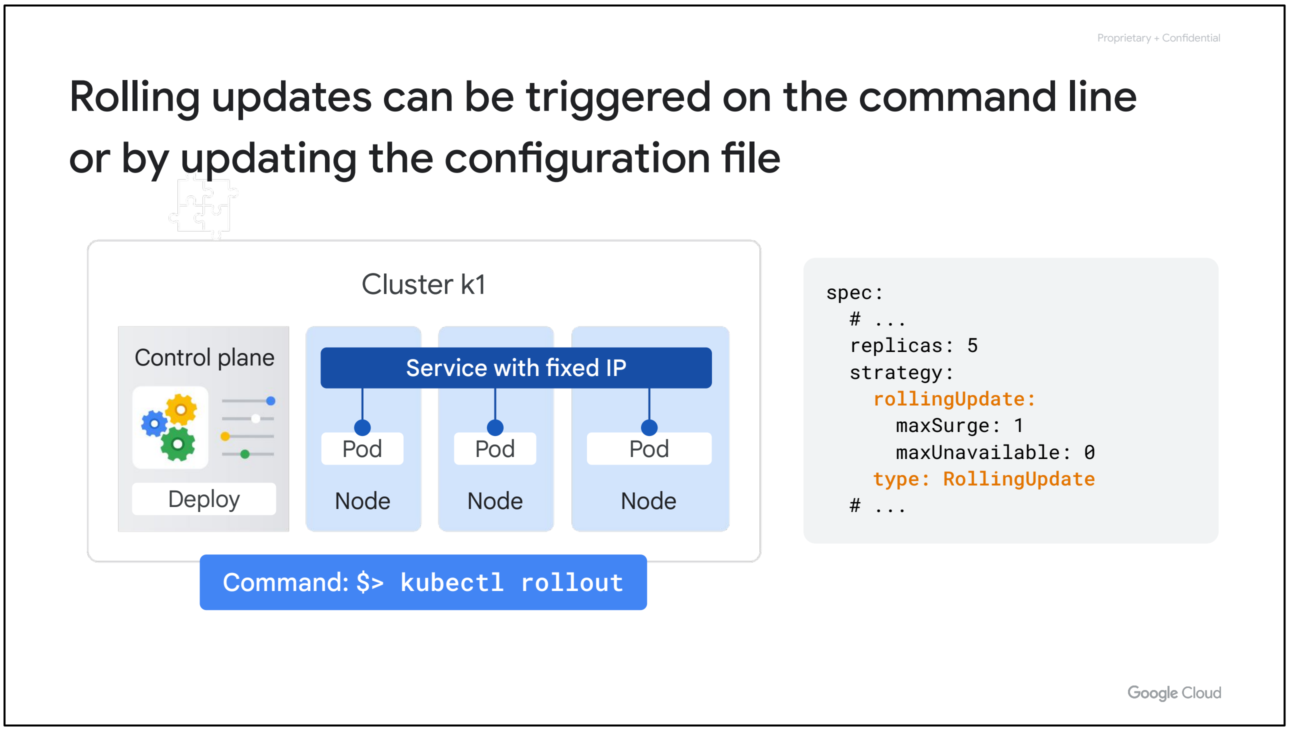


Endpoints can be discovered using a kubectl command

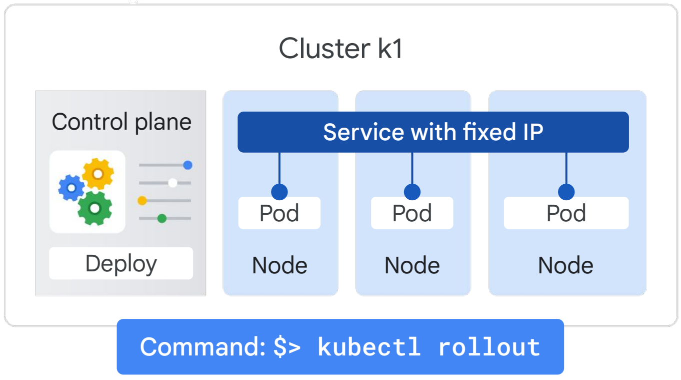


Command: `$> kubectl get services`

You can still reach your endpoint as before by using `kubectl get services` to get the external IP of the Service and reach the public IP address from a client.



Rolling updates can be triggered on the command line or by updating the configuration file



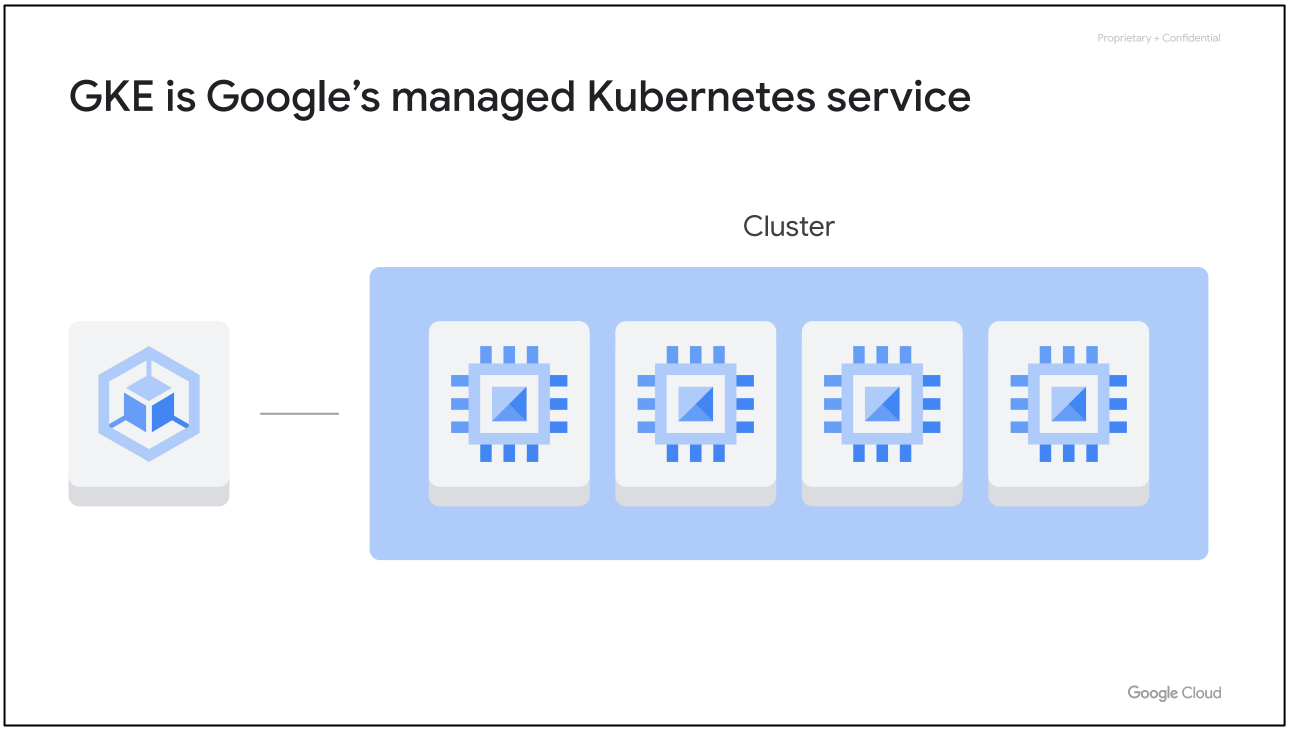
```
spec:
  # ...
  replicas: 5
  strategy:
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 0
    type: RollingUpdate
  # ...
```

The last question is, what happens when you want to update a new version of your app?

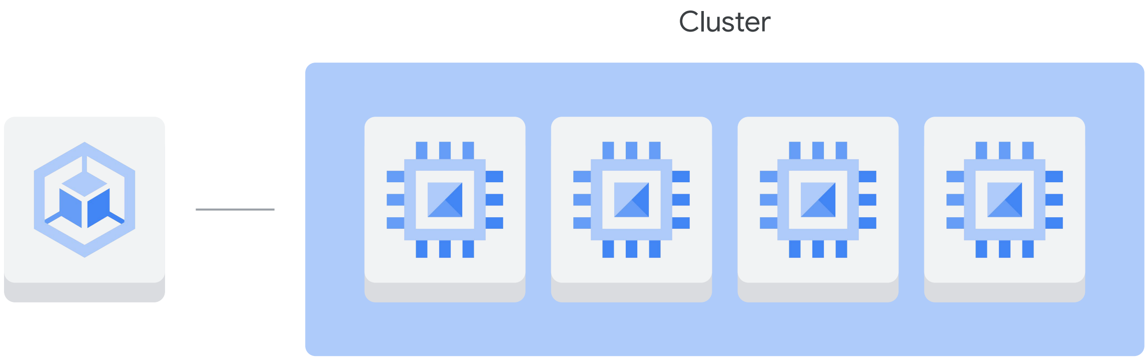
Well, you want to update your container to get new code in front of users, but rolling out all those changes at one time would be risky.

So in this case, you would use `kubectl rollout` or change your deployment configuration file and then apply the change using `kubectl apply`.

New Pods will then be created according to your new update strategy. Here is an example configuration that will create new version Pods individually and wait for a new Pod to be available before destroying one of the old Pods.

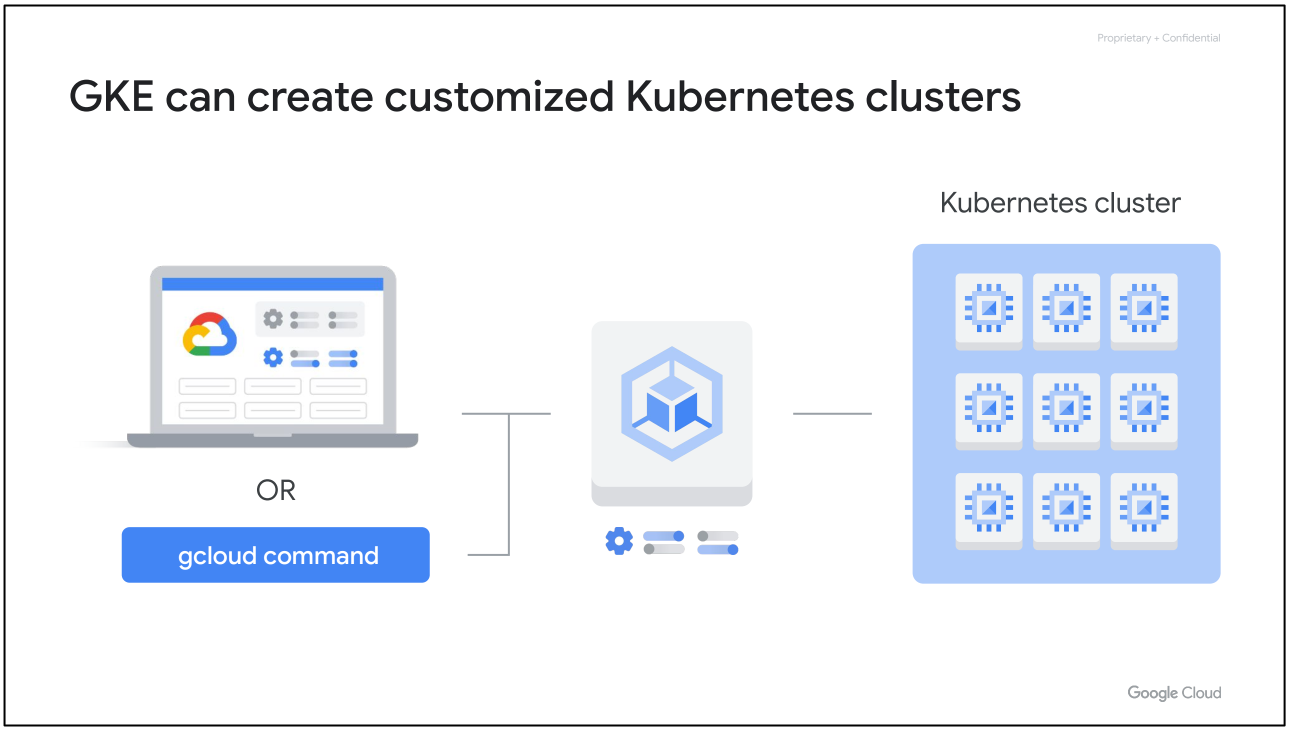


GKE is Google's managed Kubernetes service



So now that we have a basic understanding of containers and Kubernetes, let's talk about Google Kubernetes Engine, or GKE.

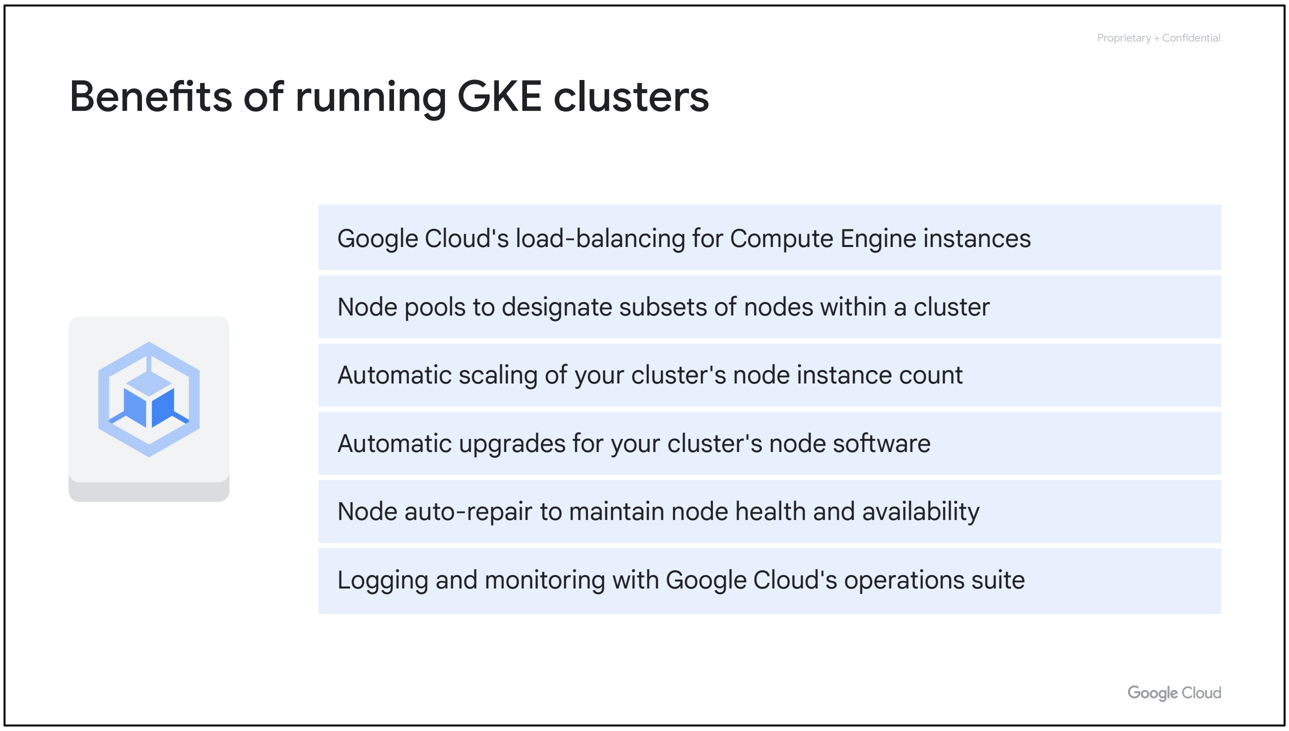
GKE is a Google-hosted managed Kubernetes service in the cloud. The GKE environment consists of multiple machines, specifically Compute Engine instances, grouped together to form a cluster.



GKE can create customized Kubernetes clusters

You can create a Kubernetes cluster with Google Kubernetes Engine by using the Google Cloud console or the gcloud command that's provided by the Cloud software development kit. GKE clusters can be customized, and they support different machine types, number of nodes, and network settings.

Kubernetes provides the mechanisms through which you interact with your cluster. Kubernetes commands and resources are used to deploy and manage applications, perform administration tasks, set policies, and monitor the health of deployed workloads.


Proprietary + Confidential

Benefits of running GKE clusters

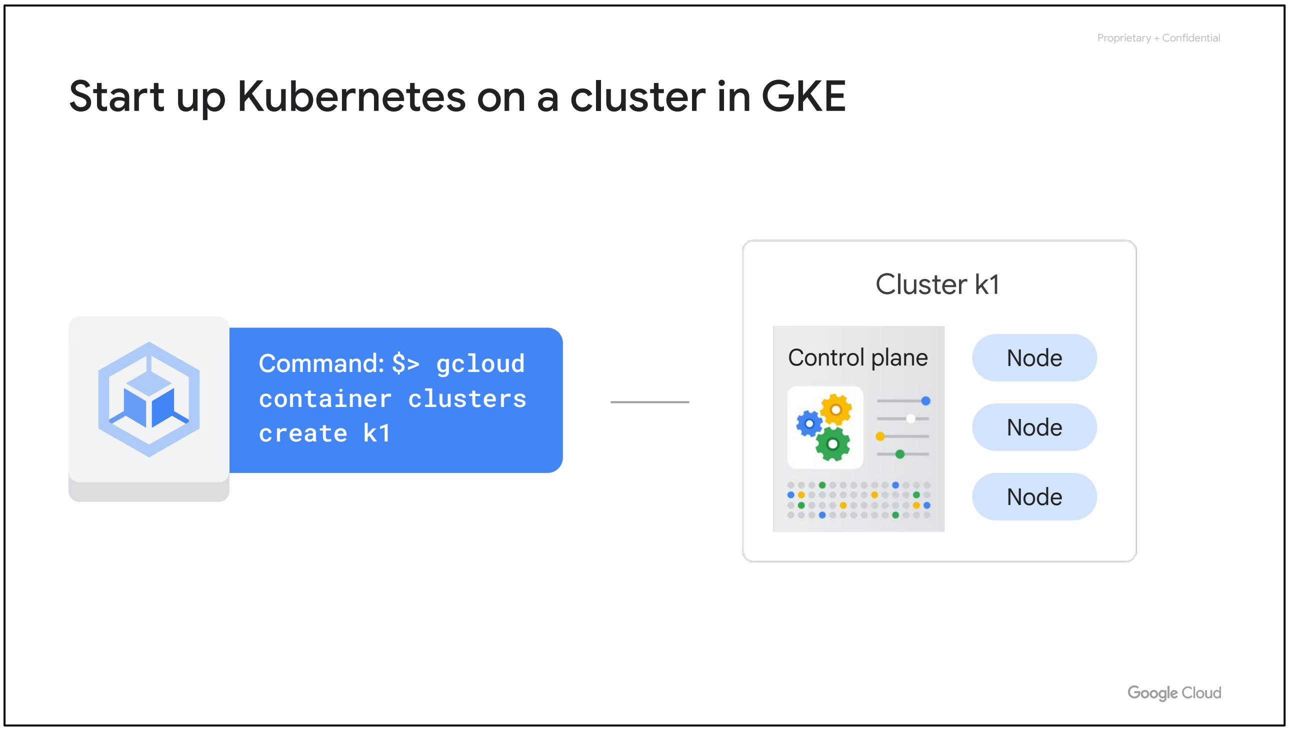
- Google Cloud's load-balancing for Compute Engine instances
- Node pools to designate subsets of nodes within a cluster
- Automatic scaling of your cluster's node instance count
- Automatic upgrades for your cluster's node software
- Node auto-repair to maintain node health and availability
- Logging and monitoring with Google Cloud's operations suite

Google Cloud

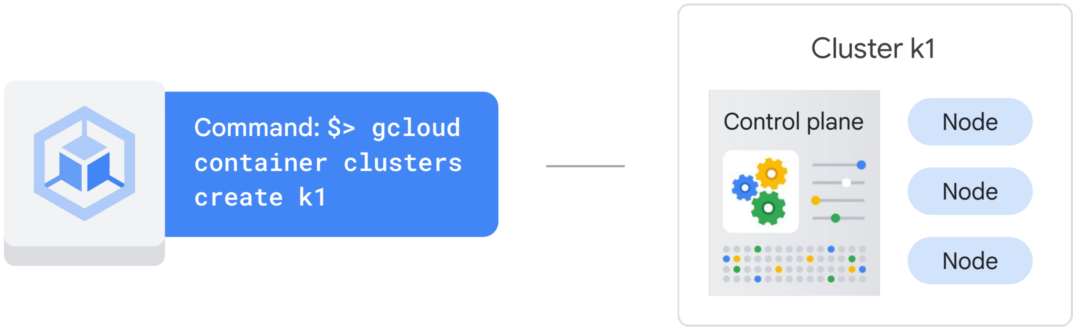
Running a GKE cluster comes with the benefit of advanced cluster management features that Google Cloud provides. These include:

- Google Cloud's load-balancing for Compute Engine instances
- Node pools to designate subsets of nodes within a cluster for additional flexibility
- Automatic scaling of your cluster's node instance count
- Automatic upgrades for your cluster's node software
- Node auto-repair to maintain node health and availability
- Logging and monitoring with Google Cloud's operations suite for visibility into your cluster

Running your application in GKE clusters is also a good foundation for moving your application into Anthos, if you will need to bridge your on-prem and cloud resources. We'll look more at that scenario in the next lesson.

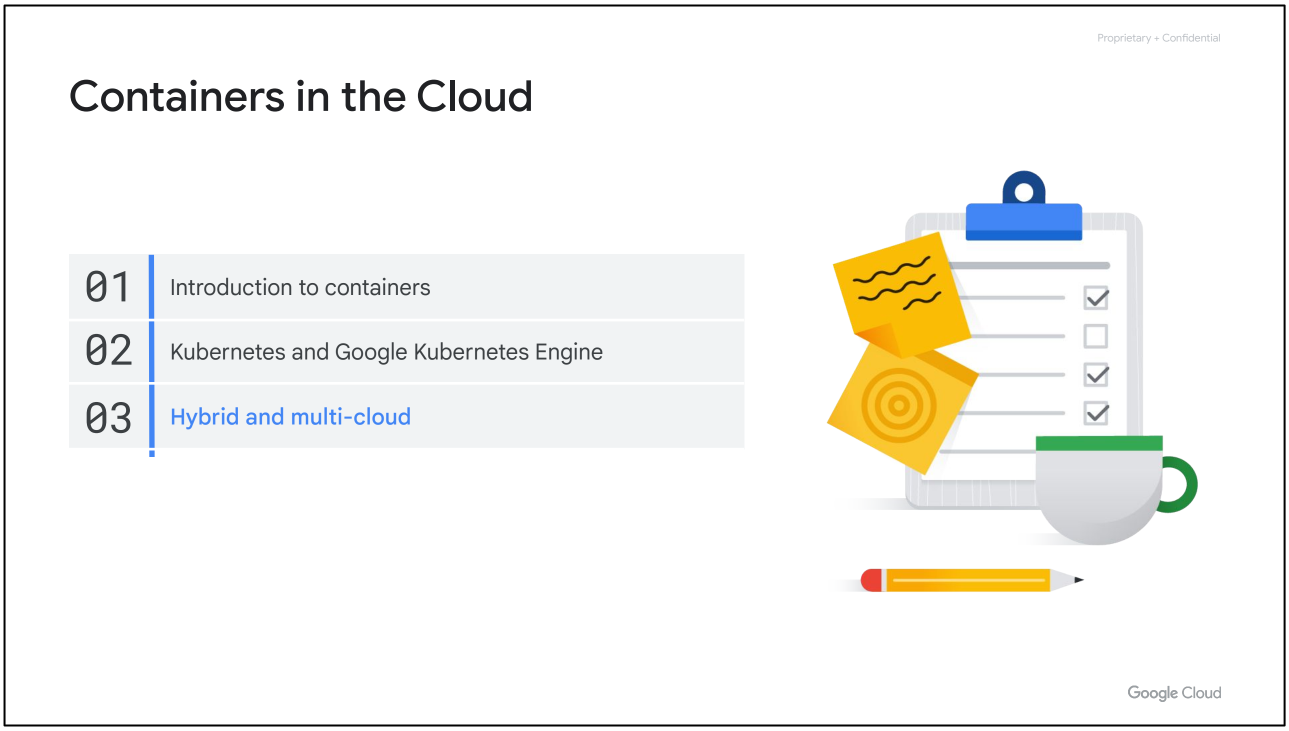


Start up Kubernetes on a cluster in GKE



To start up Kubernetes on a cluster in GKE, all you do is run this command:

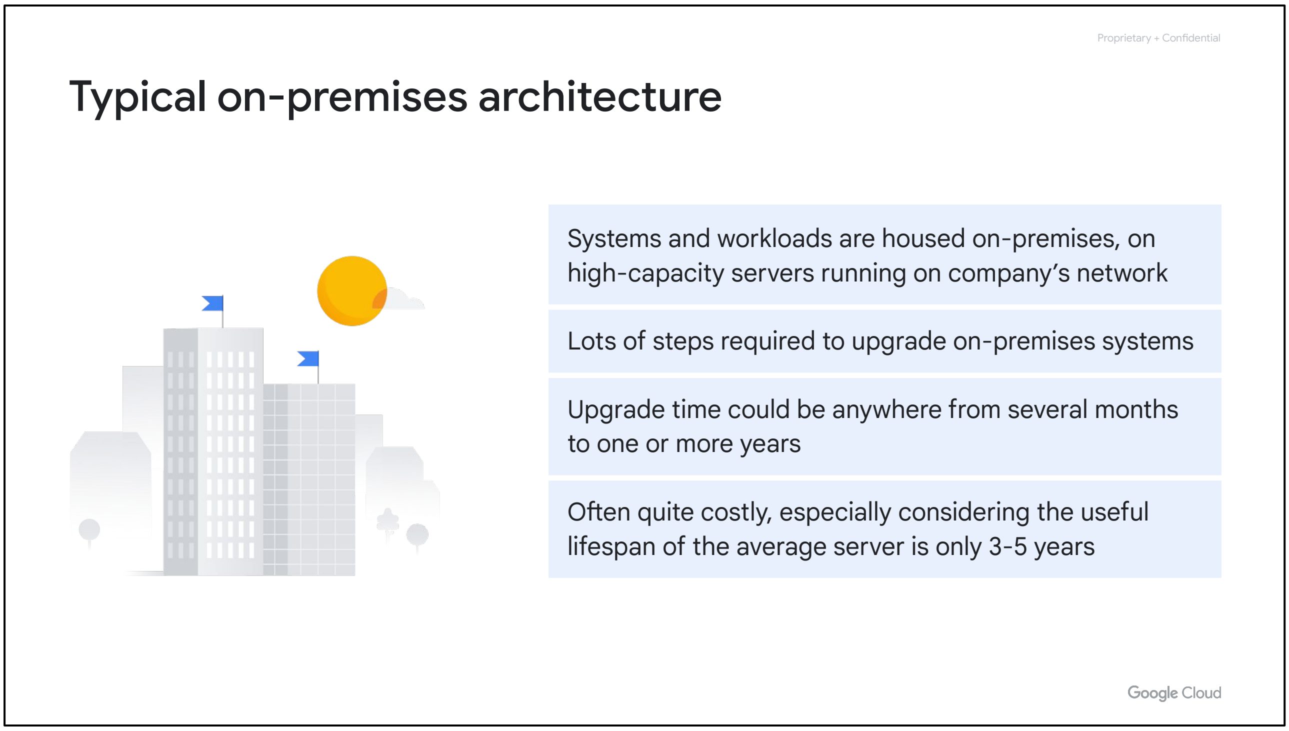
```
$> gcloud container clusters create k1
```



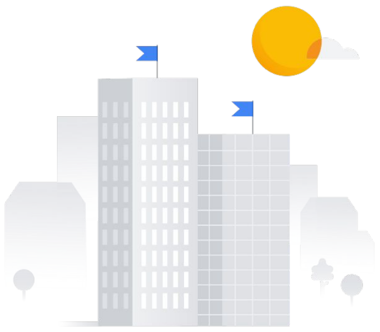
Containers in the Cloud

Google Cloud

Now that you've seen how containers work, we're going to take that information a step further and explore using them in a modern hybrid cloud and multi-cloud architecture.



Typical on-premises architecture



- Systems and workloads are housed on-premises, on high-capacity servers running on company's network
- Lots of steps required to upgrade on-premises systems
- Upgrade time could be anywhere from several months to one or more years
- Often quite costly, especially considering the useful lifespan of the average server is only 3-5 years

Google Cloud

But before we do that, let's look at a typical on-premises distributed systems architecture, which is how businesses traditionally met their enterprise computing needs before cloud computing.

As you may know, most enterprise-scale applications are designed as distributed systems, spreading the computing workload required to provide services over two or more networked servers. Over the past few years, containers have become a popular way to break these workloads down into "microservices" so they can be more easily maintained and expanded.

Traditionally, these enterprise systems–and their workloads, containerized or not–have been housed on-premises, which means they are housed on a set of high-capacity servers running somewhere within the company's network or within a company-owned data center.

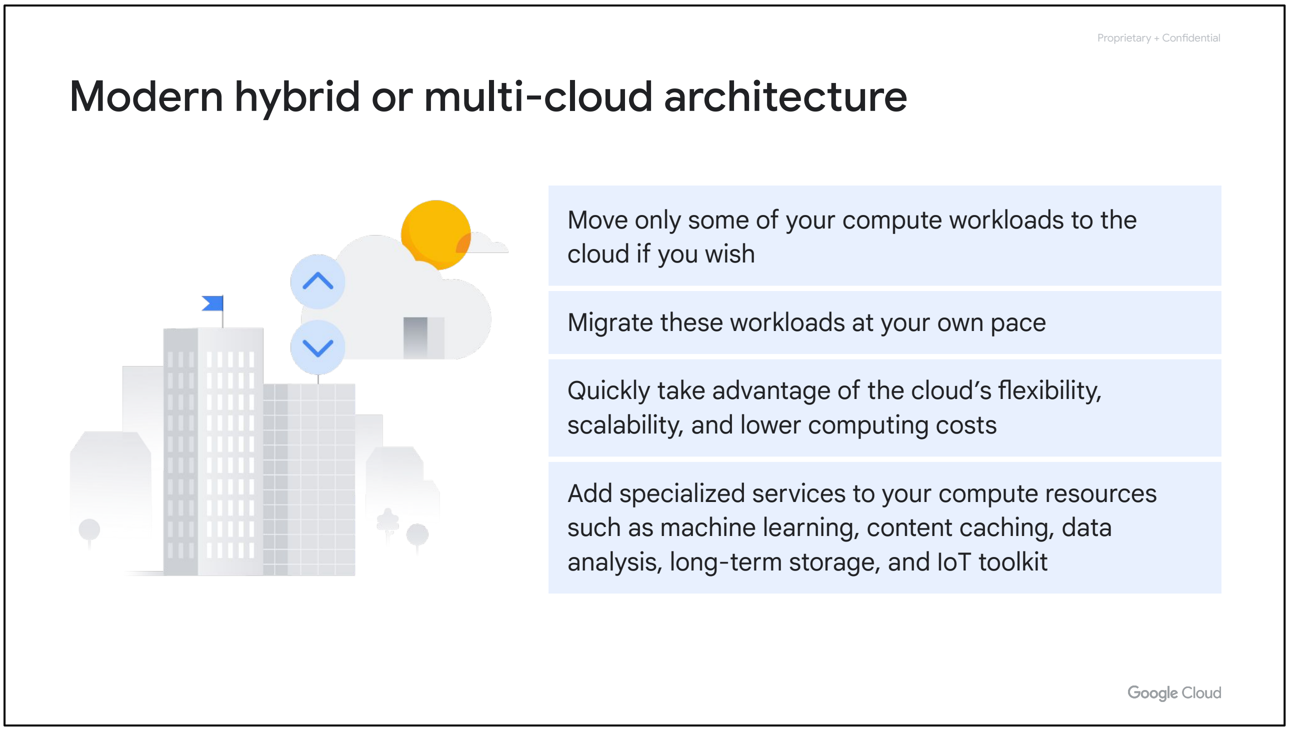
When an application's computing needs begin to outstrip its available computing resources, a company using on-premises systems would need to requisition more (or more powerful) servers, install them on the company network (after any necessary network changes or expansion), configure the new servers, and finally load the application and its dependencies onto the new servers, before resource bottlenecks could be resolved.

The time required to complete an on-premises upgrade of this kind could be anywhere from several months to one or more years. It may also be quite costly, especially when you consider the useful lifespan of the average server is only 3-5 years.

But what if you need more computing power *now*, not months from now?

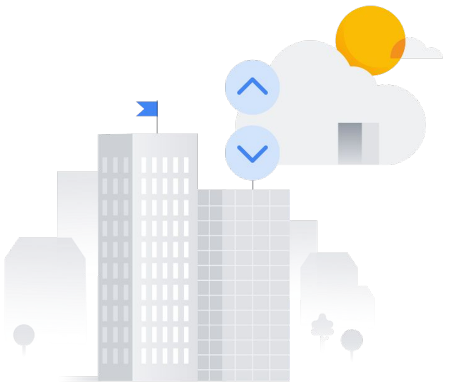
What if your company wants to relocate some workloads away from on-premises, to the cloud, to take advantage of lower costs and higher availability, but is unwilling (or unable) to move the entire enterprise application from the on-premises network?

And what if you want to use specialized products and services that are only available in the cloud?



Proprietary + Confidential

Modern hybrid or multi-cloud architecture



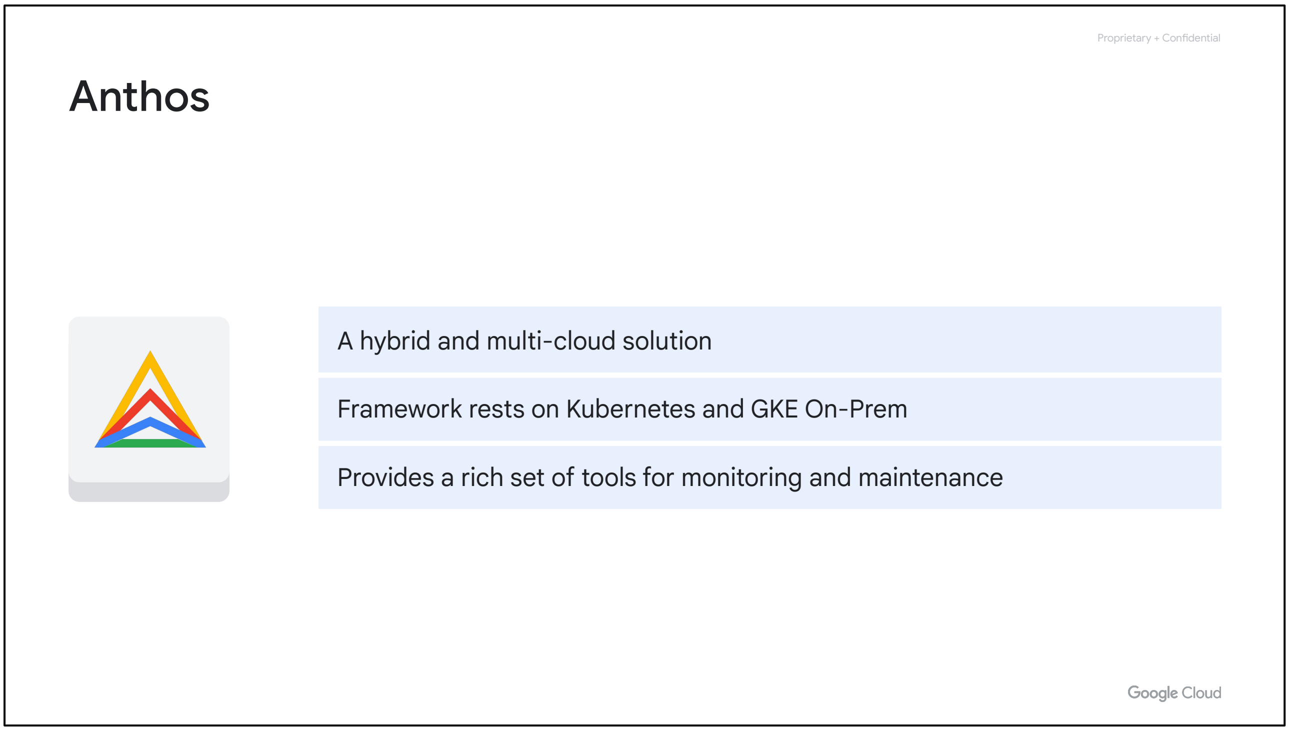
- Move only some of your compute workloads to the cloud if you wish
- Migrate these workloads at your own pace
- Quickly take advantage of the cloud's flexibility, scalability, and lower computing costs
- Add specialized services to your compute resources such as machine learning, content caching, data analysis, long-term storage, and IoT toolkit

Google Cloud

This is where a modern hybrid or multi-cloud architecture can help.

It allows you to:

- Keep parts of your systems infrastructure on-premises, while moving other parts to the cloud, creating an environment that is uniquely suited to your company's needs.
- Move only specific workloads to the cloud at your own pace, because a full scale migration is not required for it to work.
- Take advantage of the flexibility, scalability and lower computing costs offered by cloud services for running the workloads you decide to migrate.
- Add specialized services such as machine learning, content caching, data analysis, long-term storage, and IoT to your computing resources toolkit.



Proprietary + Confidential

Anthos

A hybrid and multi-cloud solution

Framework rests on Kubernetes and GKE On-Prem

Provides a rich set of tools for monitoring and maintenance

Google Cloud

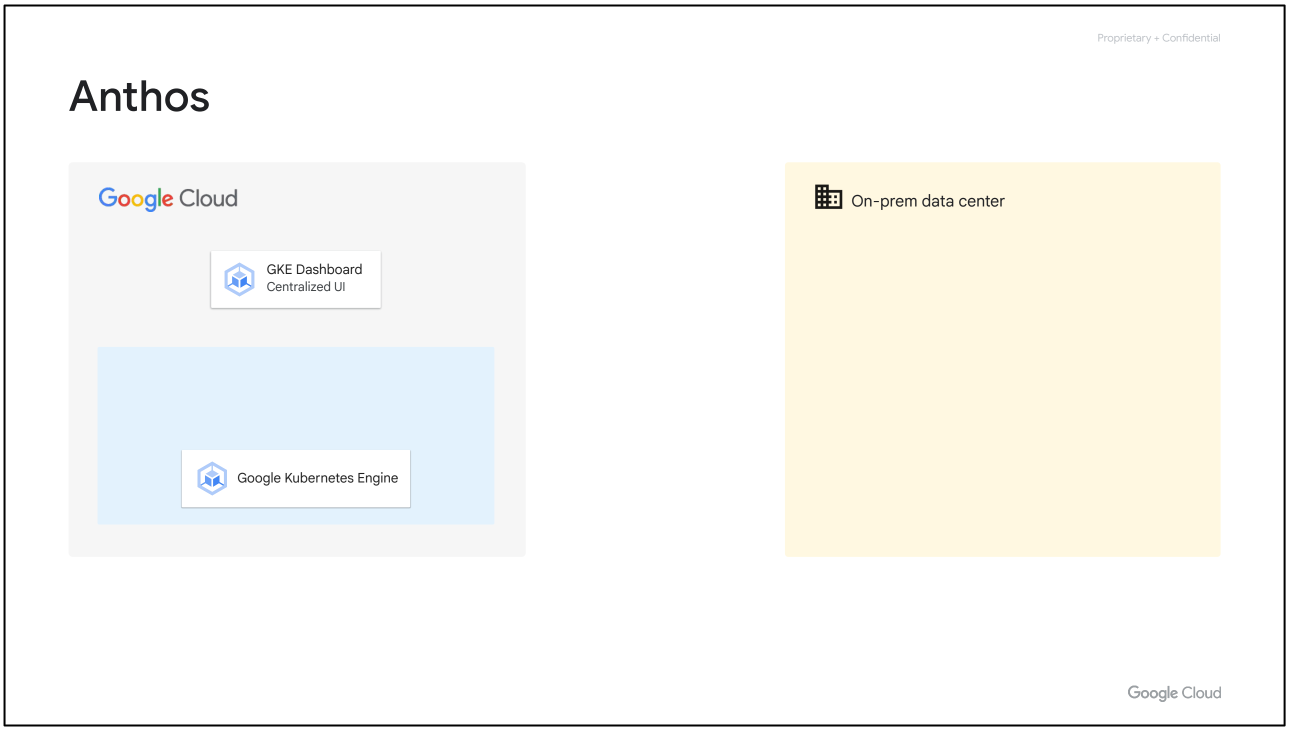
You might have heard a lot recently concerning the adoption of “hybrid” architecture for powering distributed systems and services. You might have even heard about Google’s answer to modern hybrid and multi-cloud distributed systems and services management, called **Anthos**.

But, what exactly is Anthos?

Anthos is a hybrid and multi-cloud solution powered by the latest innovations in distributed systems and service management software from Google. The Anthos framework rests on Kubernetes and GKE On-Prem, which provides the foundation for an architecture that is fully integrated, and has centralized management through a central control plane that supports policy-based application lifecycle delivery across hybrid and multiple cloud environments.

Anthos also provides a rich set of tools for monitoring and maintaining the consistency of your applications across all of your network, whether on-premises, in the cloud, or in multiple clouds.

Let’s take a deeper look at this framework, as we build a modern hybrid infrastructure stack, step by step, with Anthos.



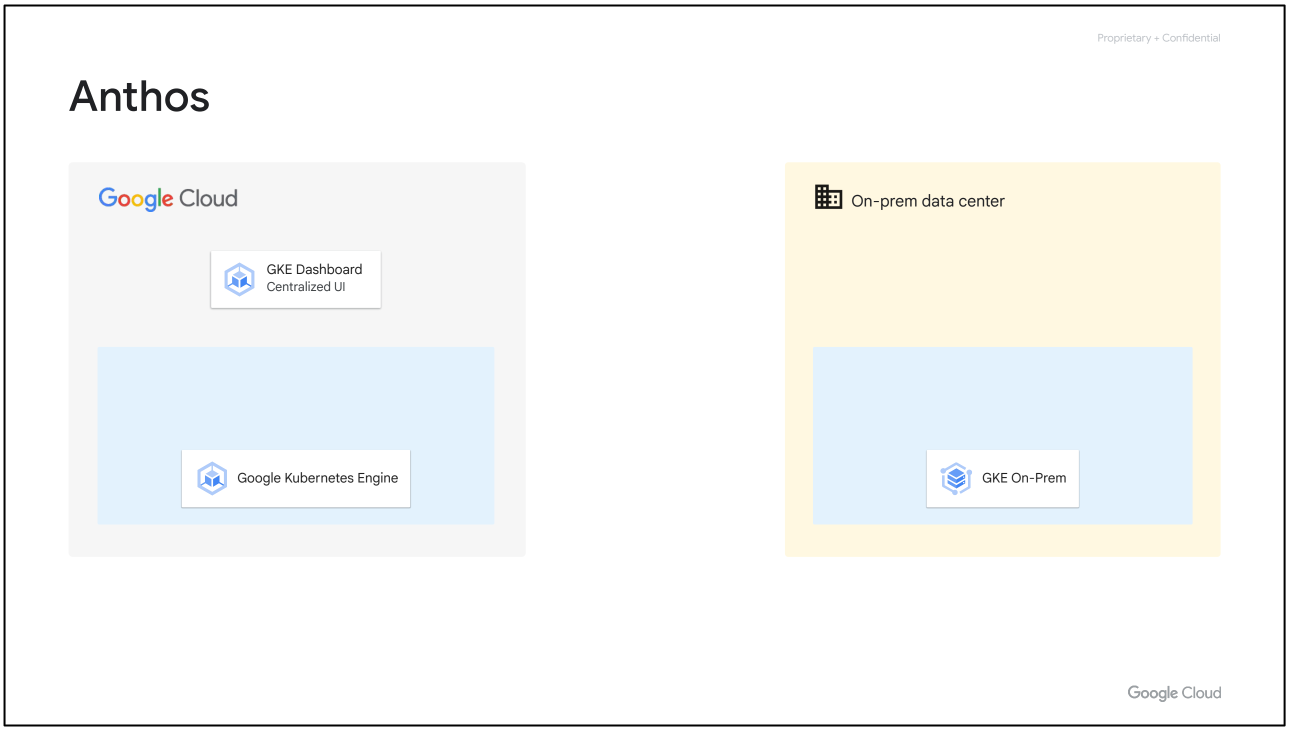
Proprietary + Confidential

Anthos

First, let’s look at Google Kubernetes Engine on the cloud side of our hybrid network.

Google Kubernetes Engine:

- Is a managed, production-ready environment for deploying containerized applications.
- Operates seamlessly with high availability and an SLA.
- Runs Certified Kubernetes, thus ensuring portability across clouds and on-premises.
- Includes auto node repair, auto upgrade, and auto scaling.
- Uses regional clusters for high availability with multiple control planes and node storage replication across multiple zones.


Proprietary + Confidential

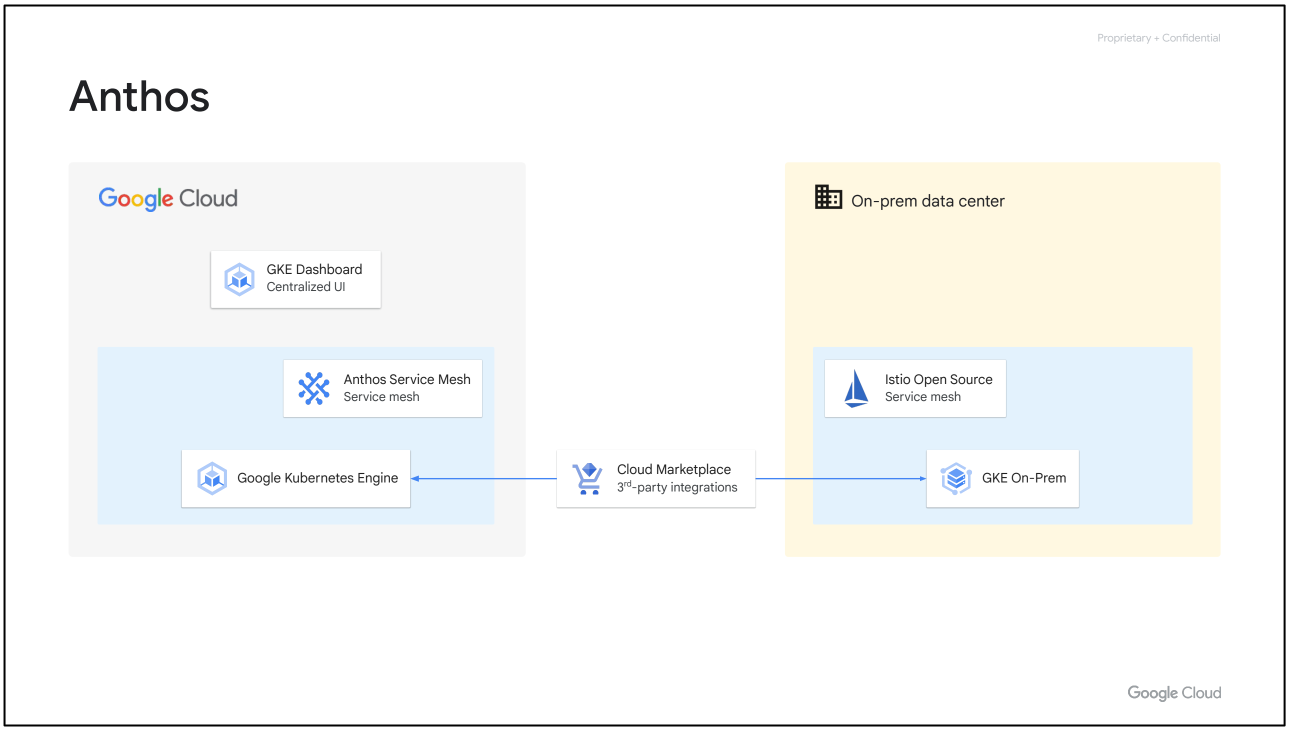
Anthos

Google Cloud

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

GKE On-Prem:

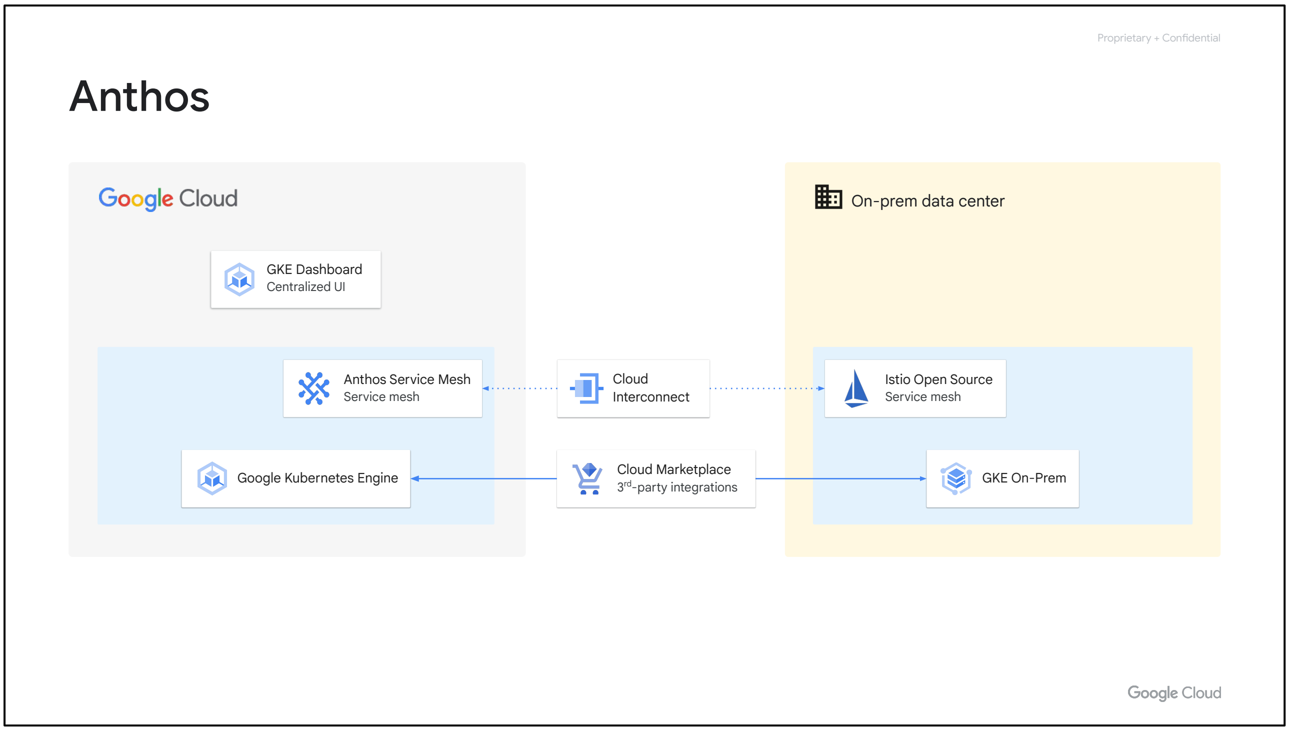
- Is a turn-key, production-grade, conformant version of Kubernetes with a best-practice configuration pre-loaded.
- Provides an easy upgrade path to the latest Kubernetes releases that have been validated and tested by Google.
- Provides access to container services on Google Cloud, such as Cloud Build,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



Anthos

Both Google Kubernetes Engine and GKE On-Prem integrate with **Marketplace** so that all of the clusters in your network, whether on-premises or in the cloud, have access to the same repository of containerized applications.

This allows you to use the same configurations on both sides of the network, which reduces time spent maintaining conformity between your clusters. You also spend less time developing applications because of a write-once/replicate-anywhere approach.

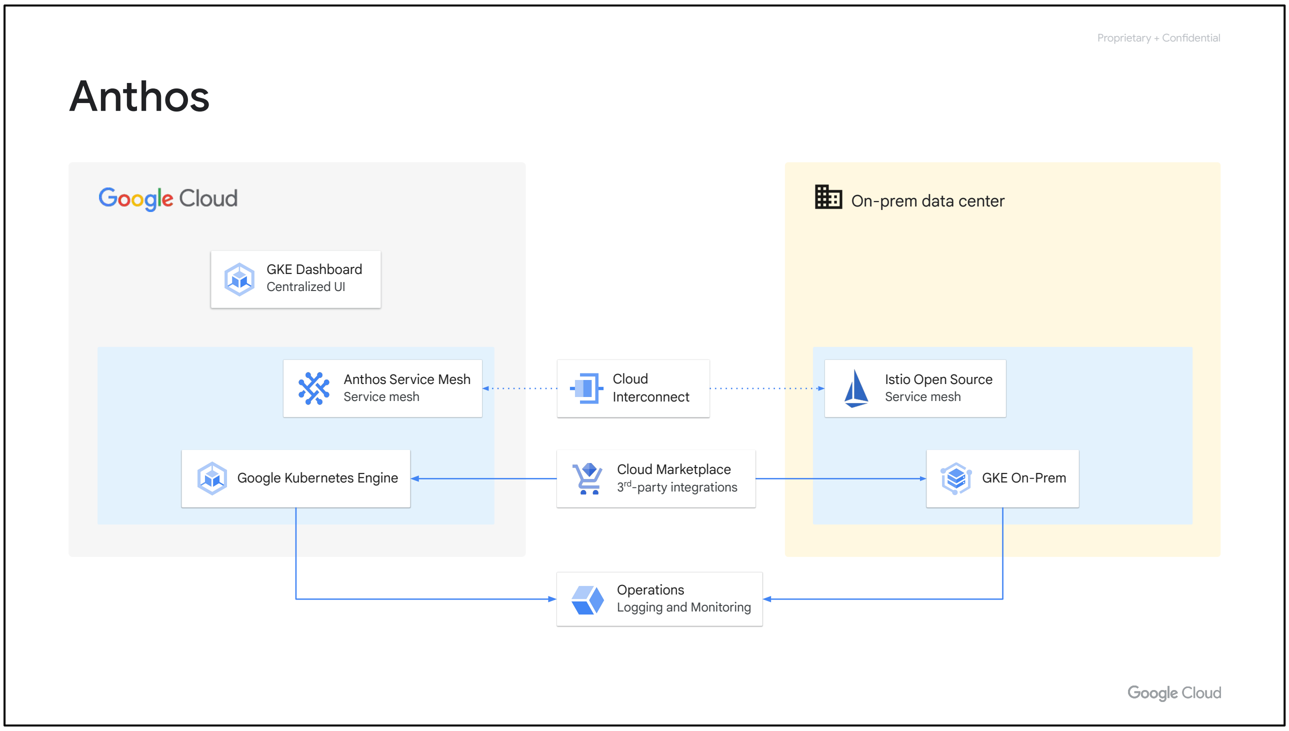


Anthos

Google Cloud

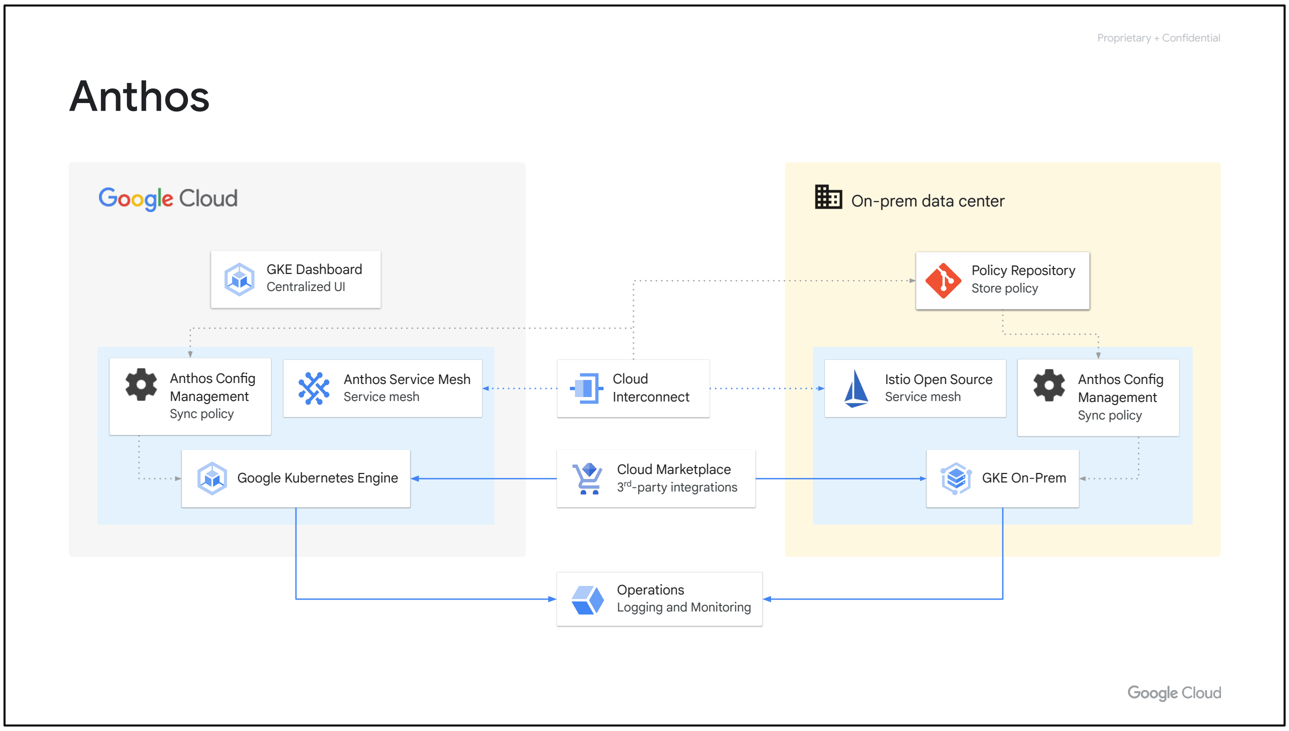
Enterprise applications might use hundreds of microservices to handle computing workloads. Keeping track of all of these services and monitoring their health can quickly become a challenge.

Anthos Service Mesh and **Istio Open Source Service Mesh** take all of the guesswork out of managing and securing your microservices. These service mesh layers communicate across the hybrid network using **Cloud Interconnect** to sync and pass their data.



Anthos

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.



Anthos

Finally, **Anthos Configuration Management** provides a single authoritative source of truth for your clusters' configuration. That authoritative source of truth is kept in the Policy Repository, which is actually a git repository.The repository can be located on-premises or hosted in the cloud. The Anthos Configuration Management agents use the Policy Repository to enforce configurations locally in each environment, thus managing the complexity of owning clusters across environments.

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

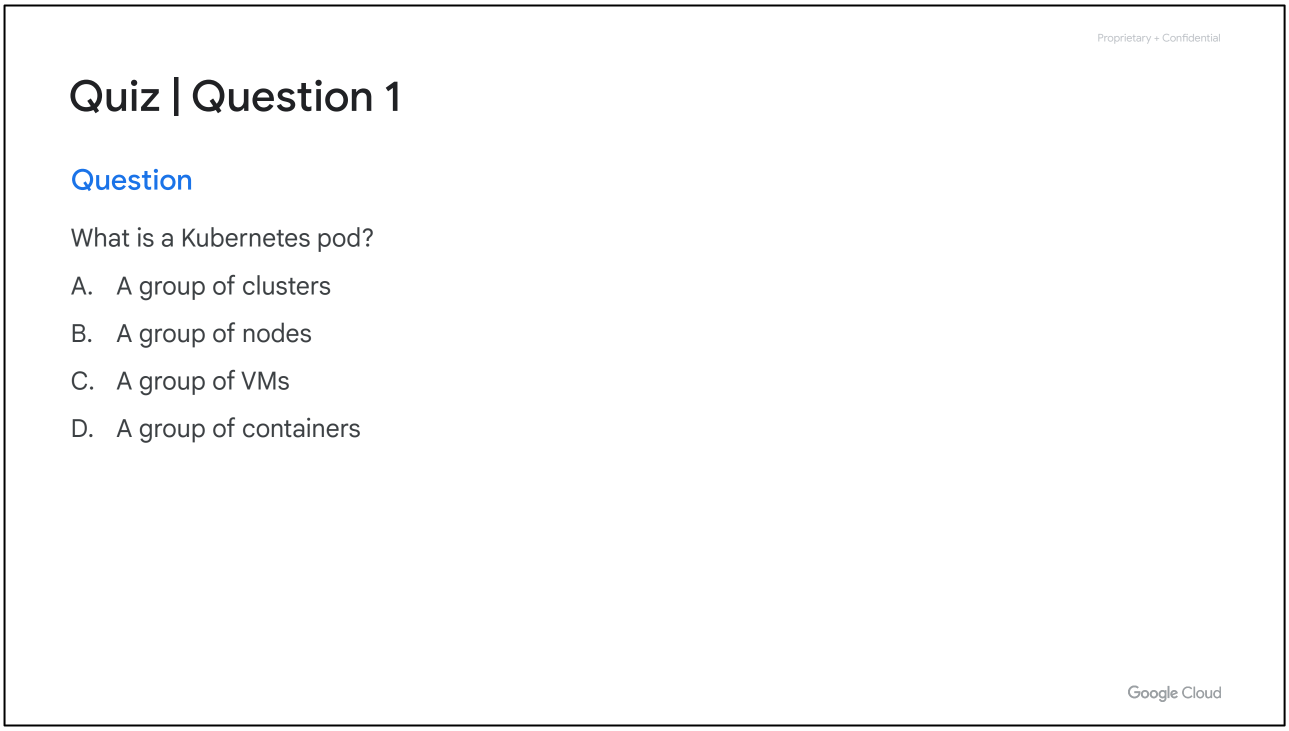
Proprietary + Confidential

cloud.google.com/anthos

Google Cloud

You can learn more about Anthos at **cloud.google.com/anthos**.

Module Quiz



Quiz | Question 1

Question

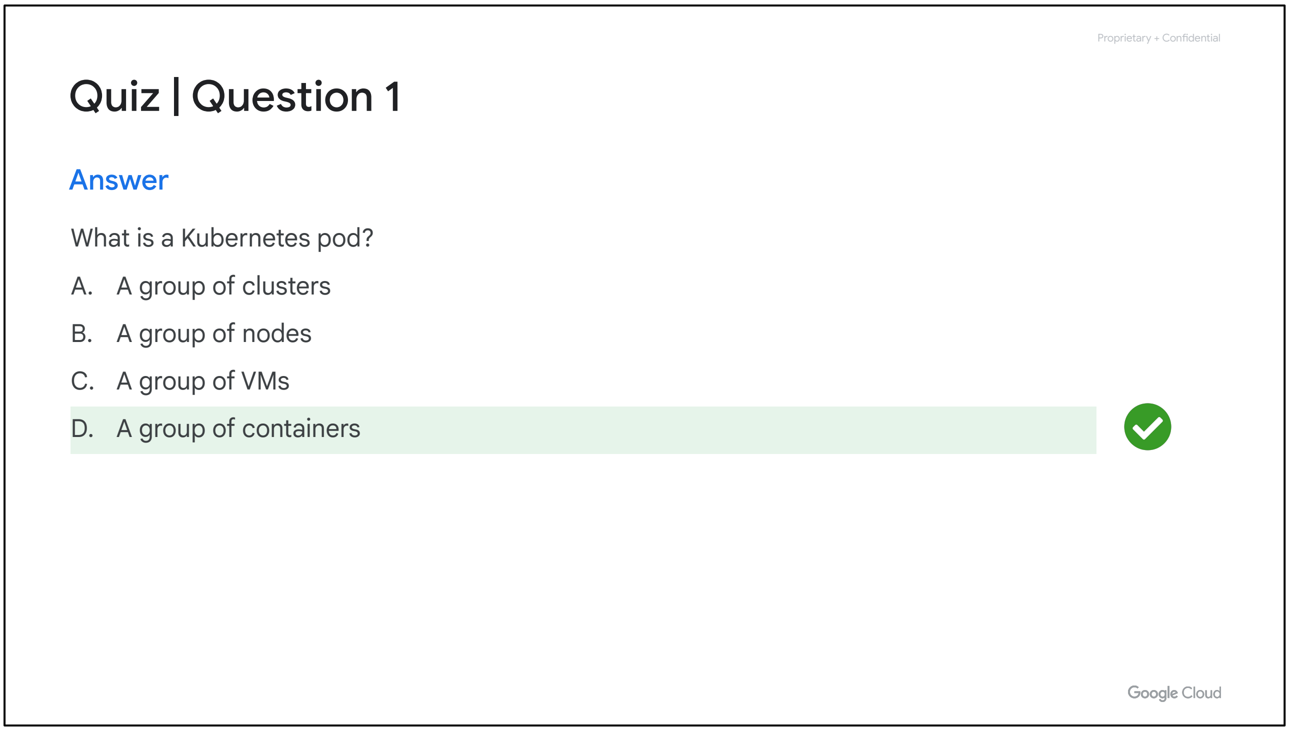
What is a Kubernetes pod?

A. A group of clusters

B. A group of nodes

C. A group of VMs

D. A group of containers



Quiz | Question 1

Answer

What is a Kubernetes pod?

A. A group of clusters

B. A group of nodes

C. A group of VMs

D. A group of containers ✓

What is a Kubernetes pod?

A: A group of clusters
Feedback: Sorry, that's not correct. In Kubernetes, a group of one or more containers is called a pod. Containers in a pod are deployed together. They are started, stopped, and replicated as a group.
The simplest workload that Kubernetes can deploy is a pod that consists only of a single container.

B: A group of nodes
Feedback: Sorry, that's not correct. In Kubernetes, a group of one or more containers is called a pod. Containers in a pod are deployed together. They are started, stopped, and replicated as a group.
The simplest workload that Kubernetes can deploy is a pod that consists only of a single container.

C: A group of VMs
Feedback: Sorry, that's not correct. In Kubernetes, a group of one or more containers is called a pod. Containers in a pod are deployed together. They are started, stopped, and replicated as a group.
The simplest workload that Kubernetes can deploy is a pod that consists only of a single container.

D: A group of containers

Feedback: That's correct. In Kubernetes, a group of one or more containers is called a pod. Containers in a pod are deployed together. They are started, stopped, and replicated as a group.
The simplest workload that Kubernetes can deploy is a pod that consists only of a single container.

Quiz | Question 2

Question

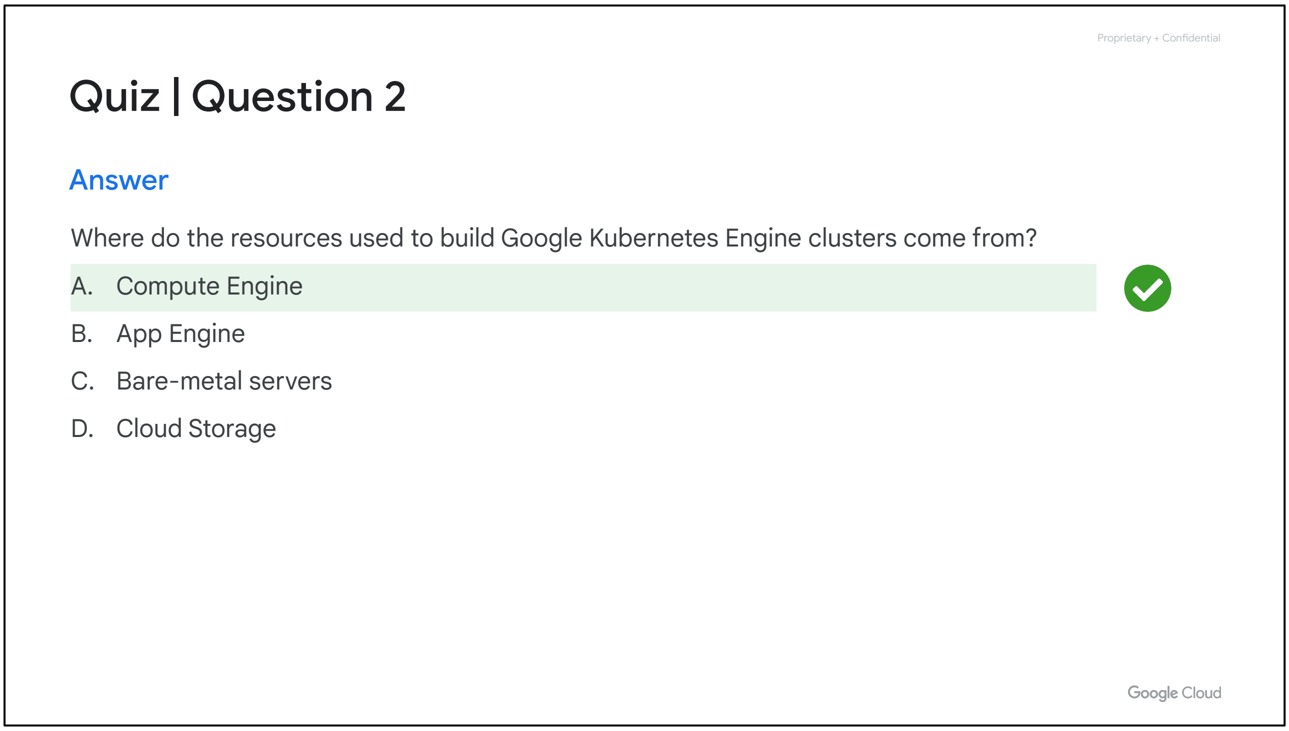
Where do the resources used to build Google Kubernetes Engine clusters come from?

A. Compute Engine

B. App Engine

C. Bare-metal servers

D. Cloud Storage



Quiz | Question 2

Answer

Where do the resources used to build Google Kubernetes Engine clusters come from?

A. Compute Engine ✓

B. App Engine

C. Bare-metal servers

D. Cloud Storage

Google Cloud

Where do the resources used to build Google Kubernetes Engine clusters come from?

A: Compute Engine
Feedback: Correct! Because the resources used to build Google Kubernetes Engine clusters come from Compute Engine, Google Kubernetes Engine gets to take advantage of Compute Engine’s and Google VPC’s capabilities.

B: App Engine
Feedback: Sorry, that's not correct. Because the resources used to build Google Kubernetes Engine clusters come from Compute Engine, Google Kubernetes Engine gets to take advantage of Compute Engine’s and Google VPC’s capabilities.

C: Bare-metal servers
Feedback: Sorry, that's not correct. Because the resources used to build Google Kubernetes Engine clusters come from Compute Engine, Google Kubernetes Engine gets to take advantage of Compute Engine’s and Google VPC’s capabilities.

D: Cloud Storage
Feedback: Sorry, that's not correct. Because the resources used to build Google Kubernetes Engine clusters come from Compute Engine, Google Kubernetes Engine gets to take advantage of Compute Engine’s and Google VPC’s capabilities.

Lab Intro

Google Cloud Fundamentals: Getting Started with GKE

In this lab, you create a Google Kubernetes Engine cluster containing several containers, each containing a web server. You place a load balancer in front of the cluster and view its contents.