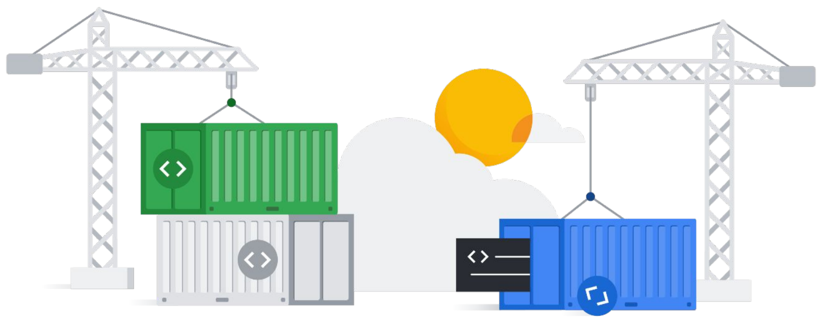
Google Cloud

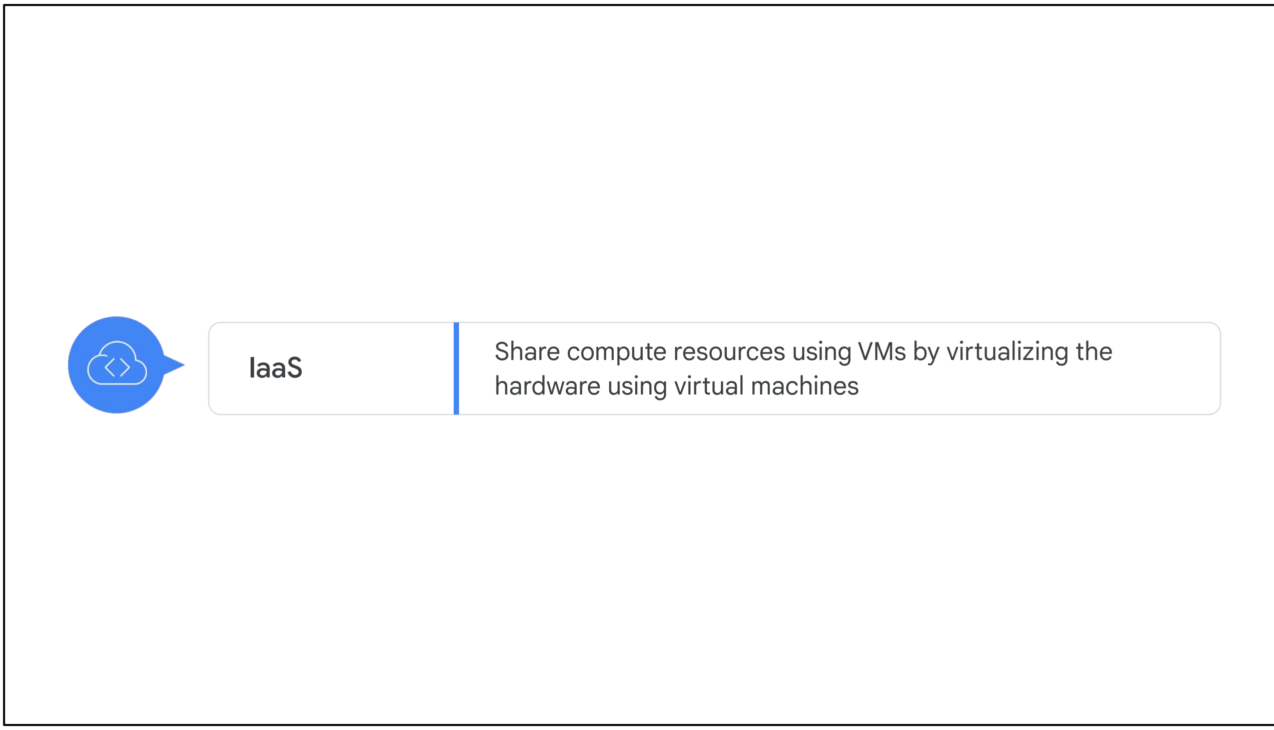
Google Cloud Core Infrastructure Module 5

On-demand course
March 2022

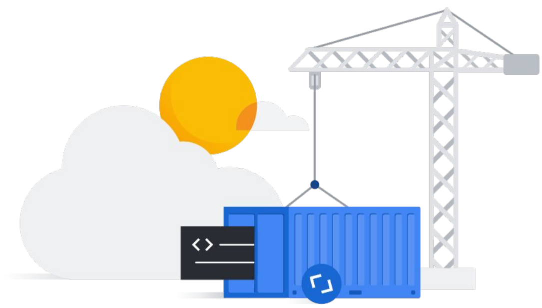


Containers

In this section of the course we'll explore containers and help you understand how they are used.



Infrastructure as a service, or IaaS, allows you to share compute resources with other developers by using virtual machines to virtualize the hardware. This lets each developer deploy their own operating system (OS), access the hardware, and build their applications in a self-contained environment with access to RAM, file systems, networking interfaces, etc.



- Independent workload scalability
- OS and hardware abstraction layer

This is where **containers** come in.

The idea of a container is to give the independent scalability of workloads in PaaS and an abstraction layer of the OS and hardware in IaaS.

A *configurable* system lets you install your favorite runtime, web server, database, or middleware, configure the underlying system resources, such as disk space, disk I/O, or networking, and build as you like.

But flexibility comes with a cost. The smallest unit of compute is an app with its VM. The guest OS might be large, even gigabytes in size, and take minutes to boot.

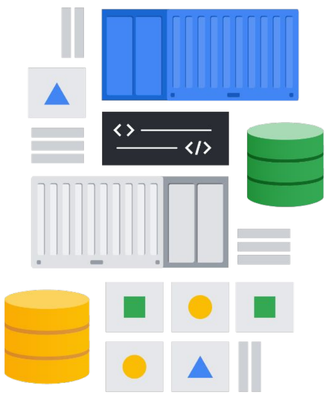
As demand for your application increases, you have to copy an entire VM and boot the guest OS for each instance of your app, which can be slow and costly.

App Engine

- Access to programming services to write your code
- Seamless, independent and rapid scaling
- No fine-tuning the underlying architecture to save cost

Now, with App Engine, you get access to programming services, so you only need to write your code in self-contained workloads that use these services and include any dependent libraries. This means that as demand for your app increases, the platform scales your app seamlessly and independently by workload and infrastructure.

This scales rapidly, but there's no option to fine-tune the underlying architecture to save cost.



A container:

Is 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

Scales like PaaS, but gives nearly the same flexibility as IaaS

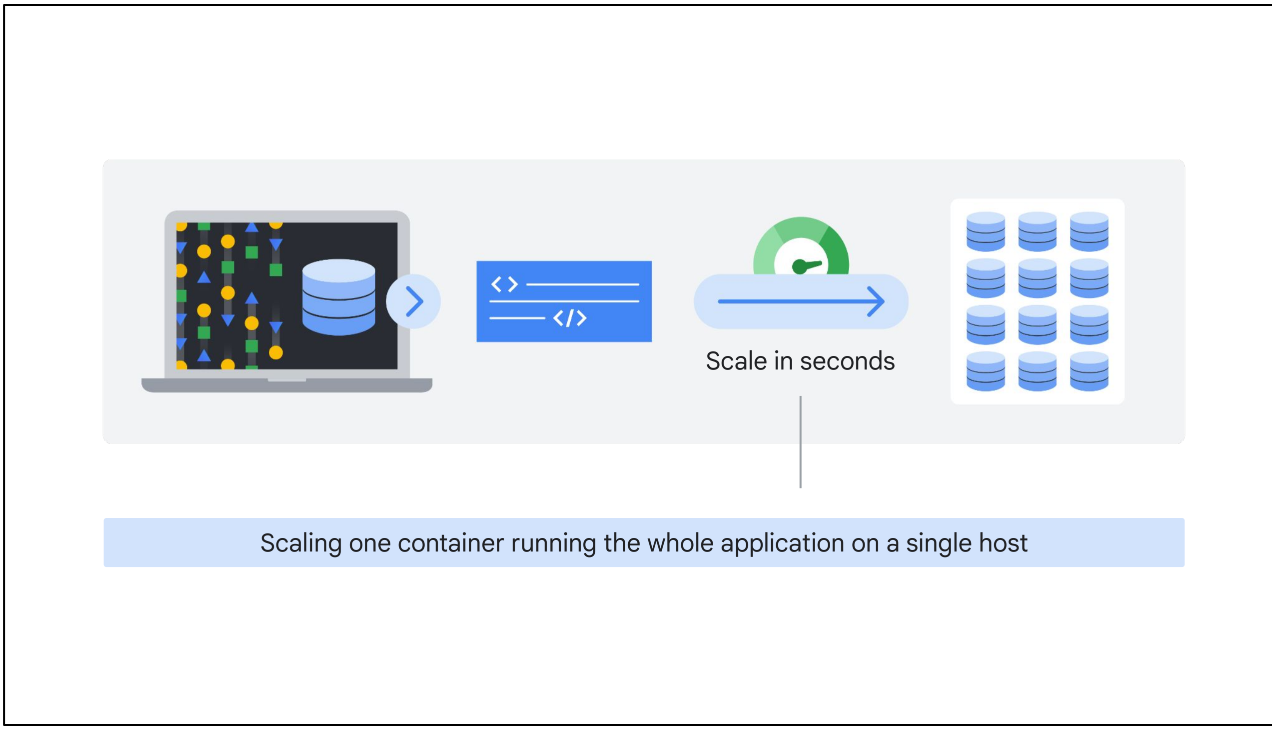
A **container** is an invisible box around your code and its dependencies with limited access to its own partition of the file system and hardware.

It only requires a few system calls to create and it starts as quickly as a process.

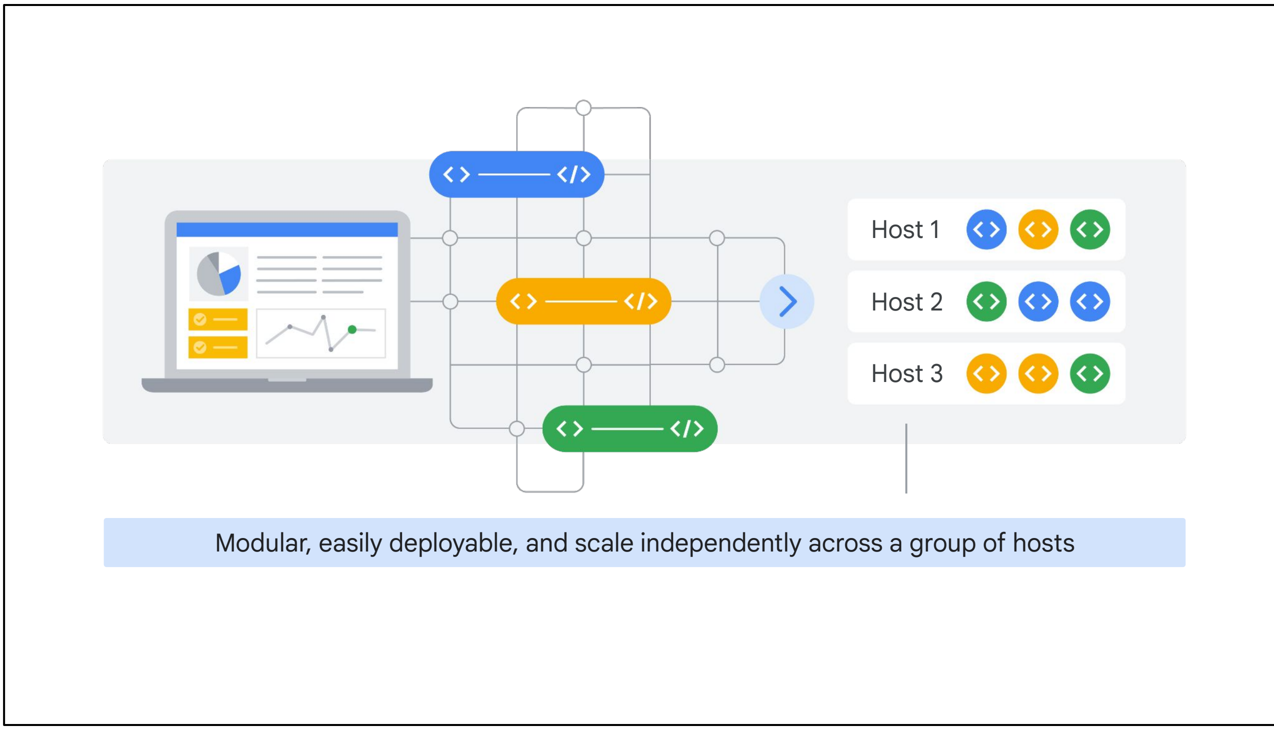
All that's needed on each host is an OS kernel that supports containers and a container runtime.

In essence, the OS is being virtualized. It scales like PaaS but gives you nearly the same flexibility as IaaS.

This makes 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.

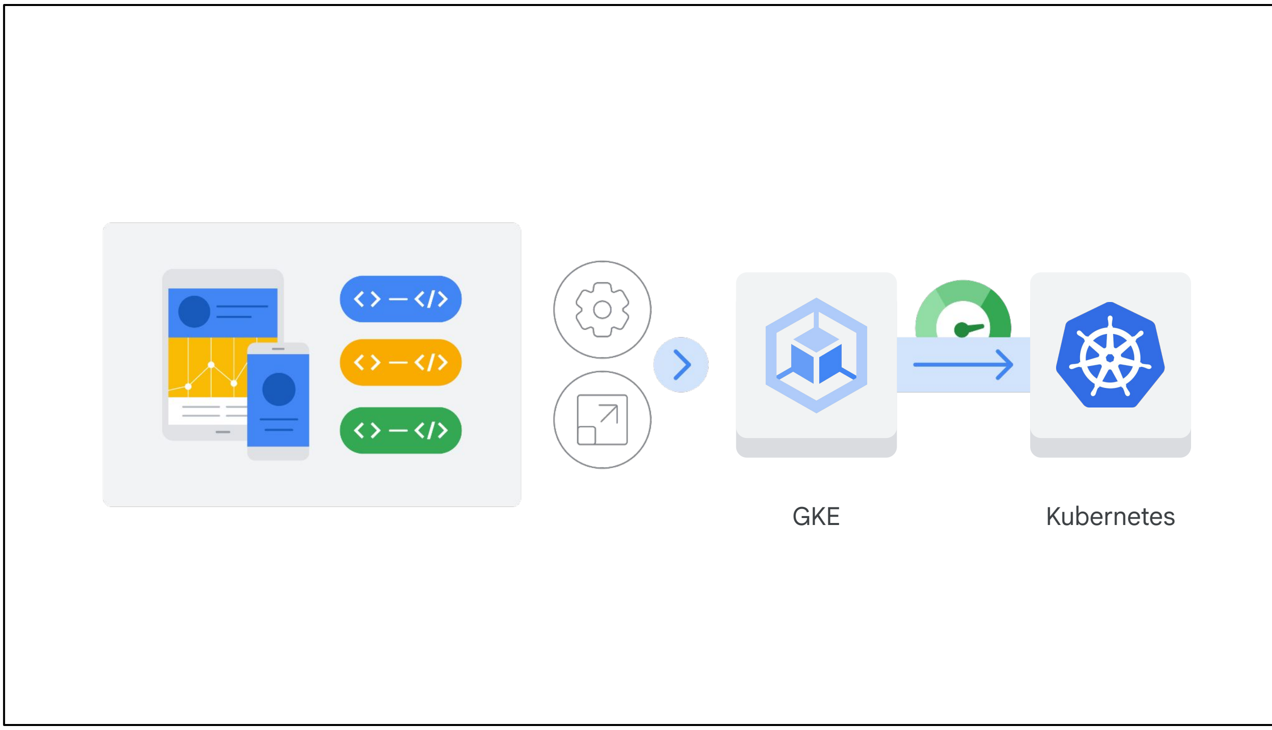


That's just a simple example of scaling one container running the whole application on a single host.



make them modular, deploy easily, and scale independently across a group of hosts.

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



So to save time and effort when scaling applications and workloads, Kubernetes can be bootstrapped using **Google Kubernetes Engine** or GKE.

What is Kubernetes?

So, what is Kubernetes?

Kubernetes:

- 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

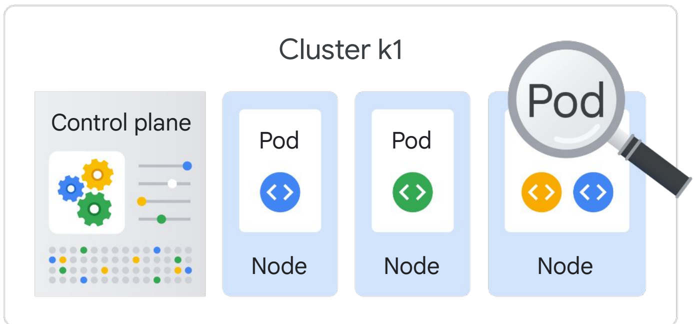
Kubernetes is an open-source platform for managing containerized workloads and services.

It makes it easy to orchestrate many containers on many hosts, scale them as microservices, and easily deploy rollouts and rollbacks.

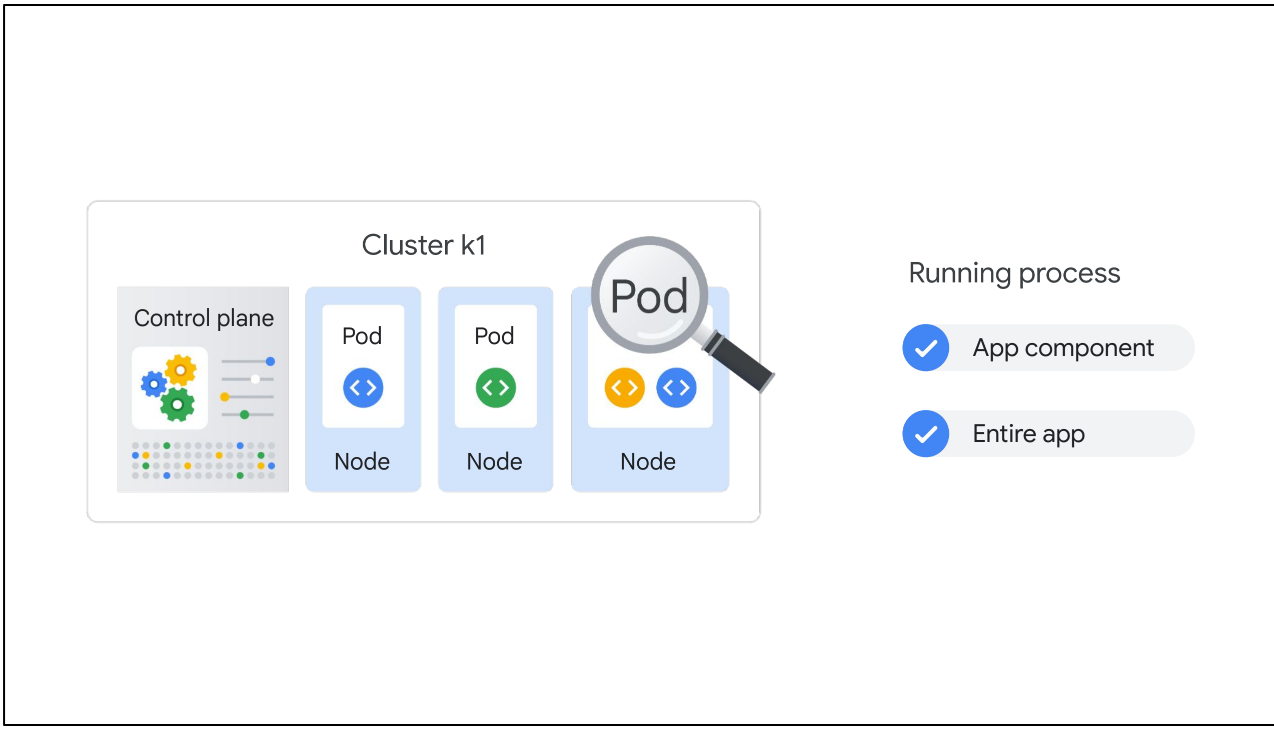
At the highest level, Kubernetes is a set of APIs that you can use to deploy containers on a set of nodes called a *cluster*.

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.

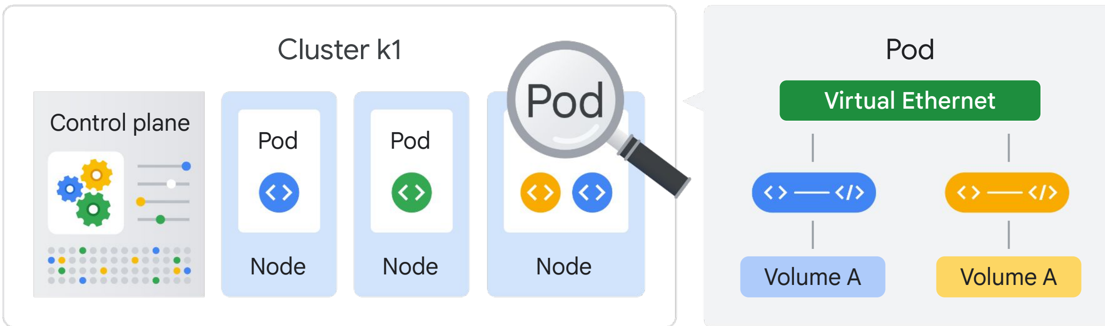
You can describe a set of applications and how they should interact with each other, and Kubernetes determines how to make that happen.



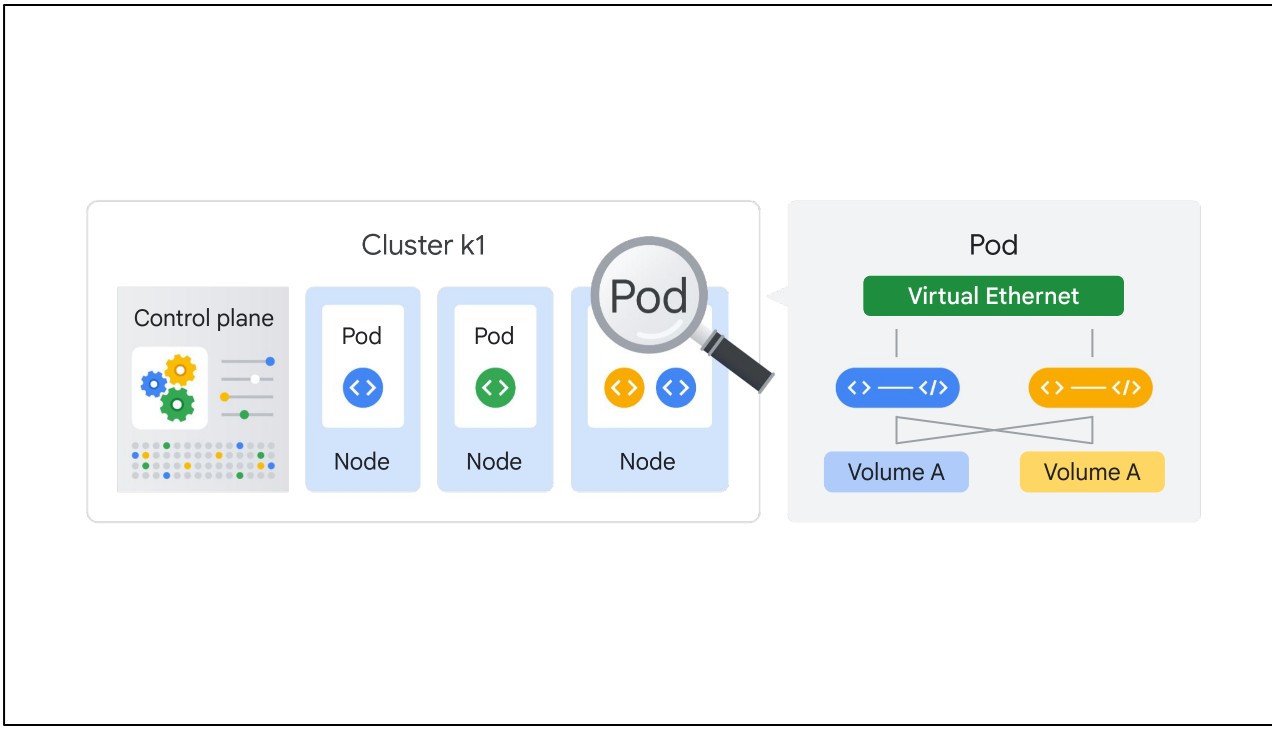
Deploying containers on nodes by using a wrapper around one or more containers is what defines a Pod.



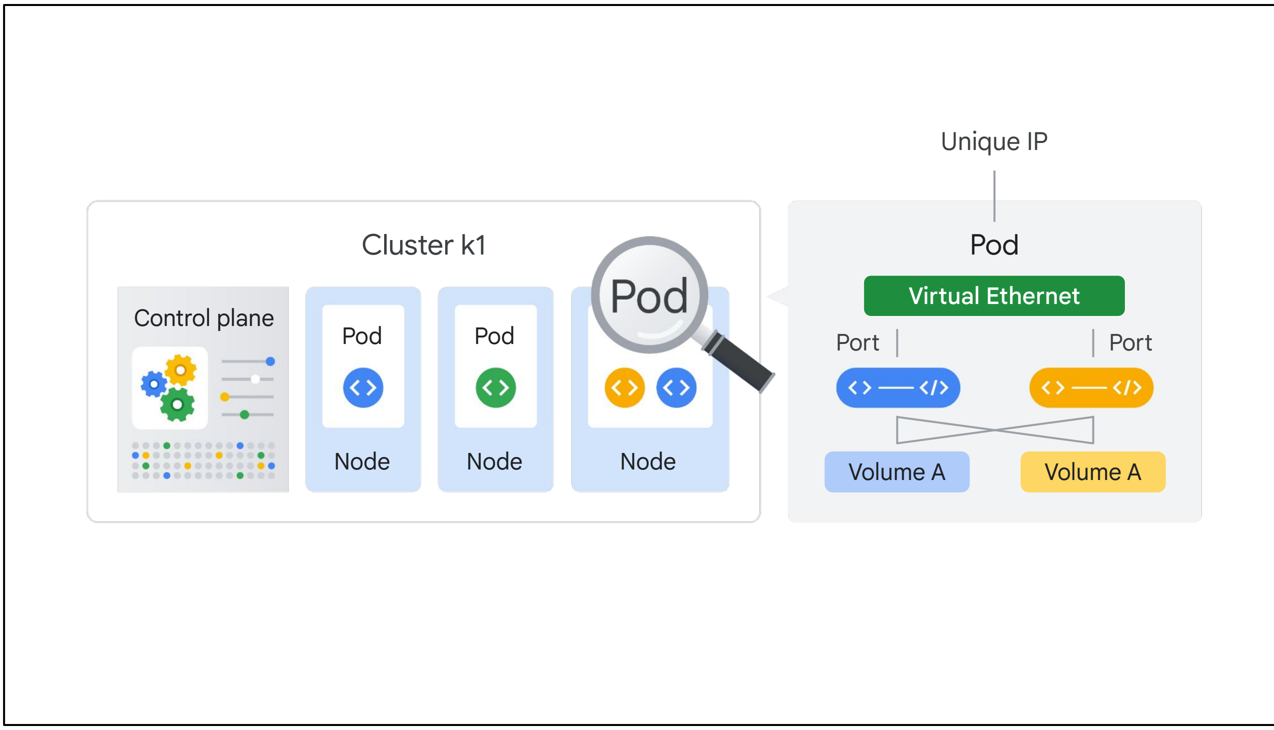
It represents a running process on your cluster as either a component of your application or an entire app.



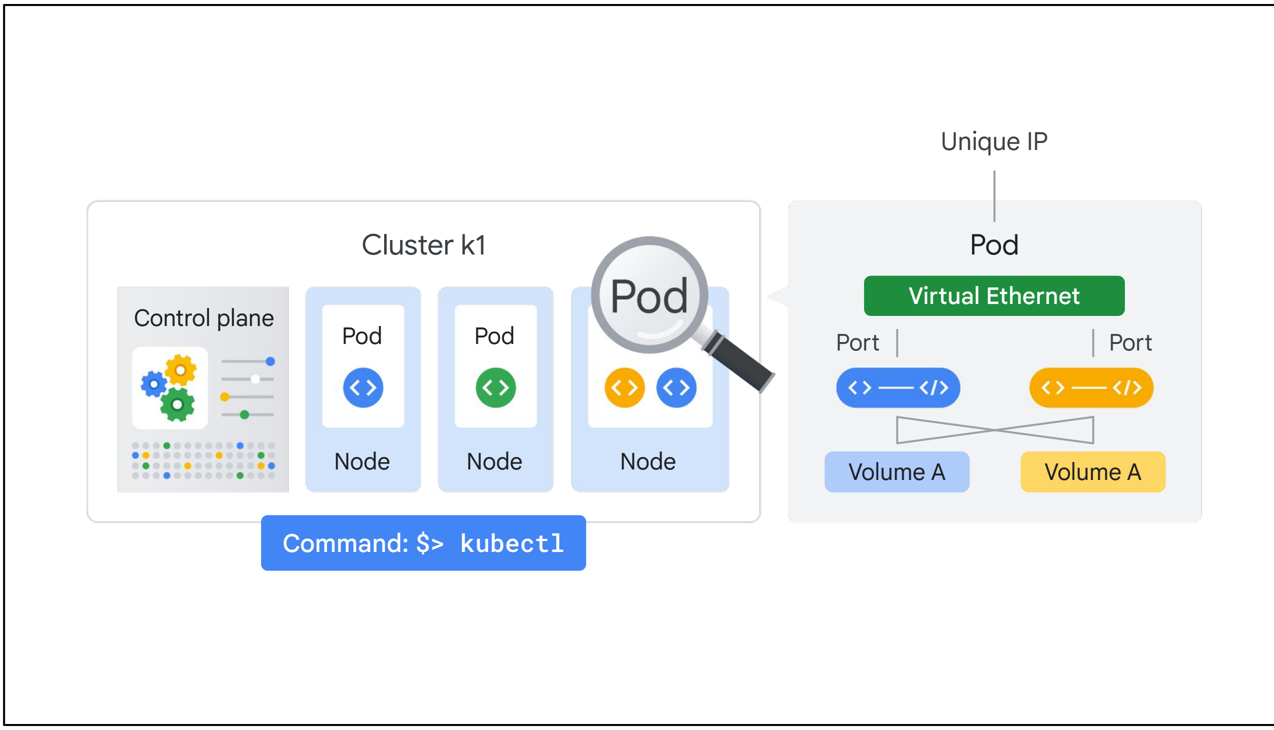
Generally, you only have one container per Pod, but if you have multiple containers with a hard dependency, you can package them into a single Pod



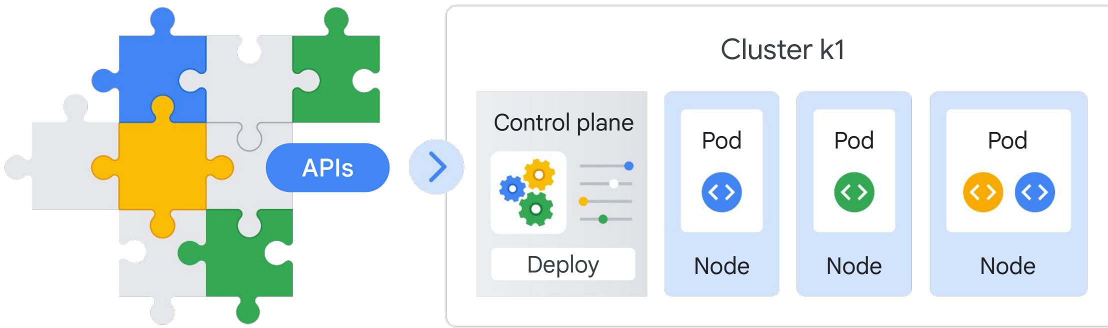
and share networking and storage resources between them.



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

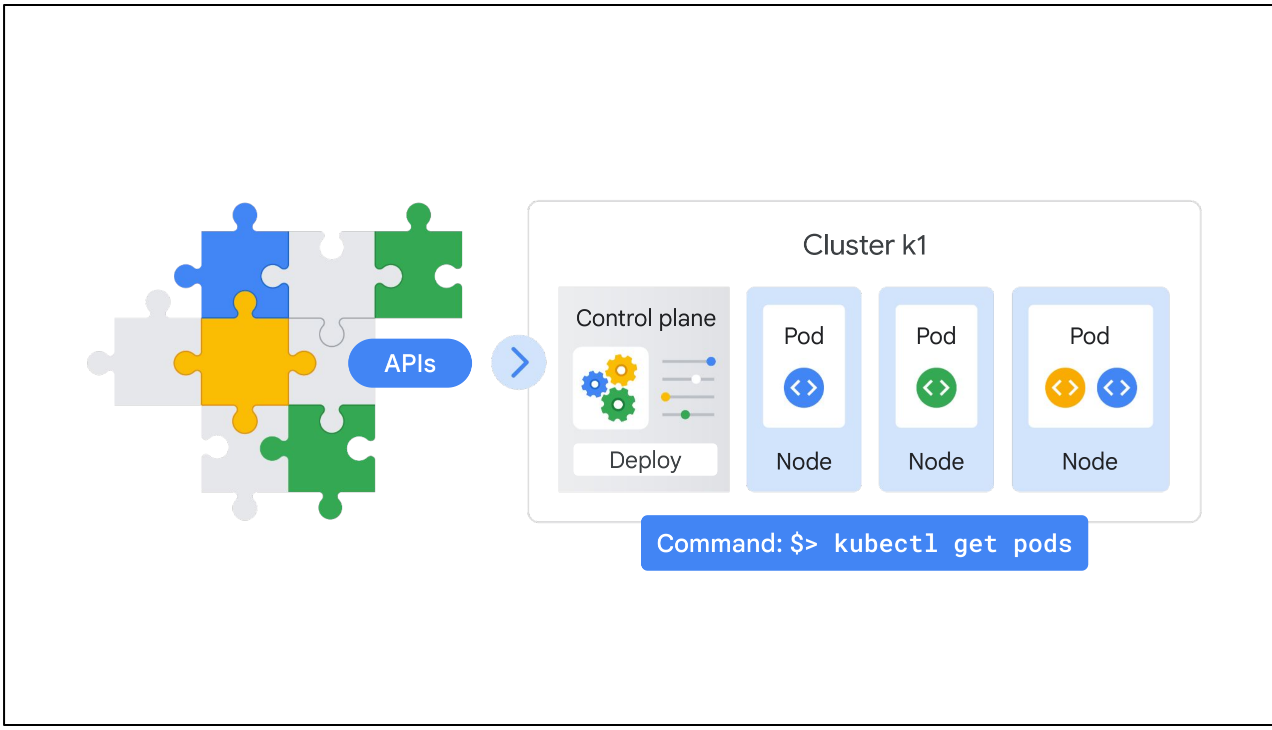


`kubectl` run command, which starts a Deployment with a container running inside a Pod.



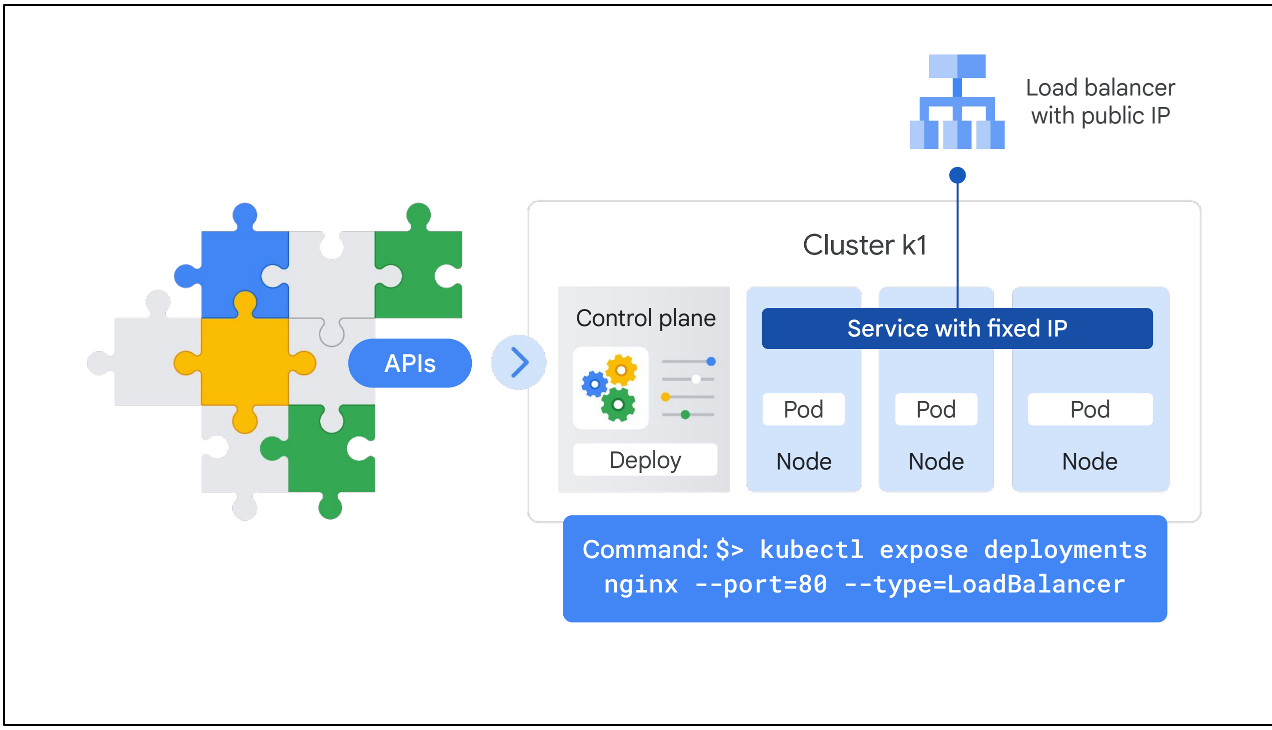
A Deployment represents a group of replicas of the same Pod and keeps your Pods running even when the nodes they run on fail. A Deployment could represent a component of an application or even an entire app.

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



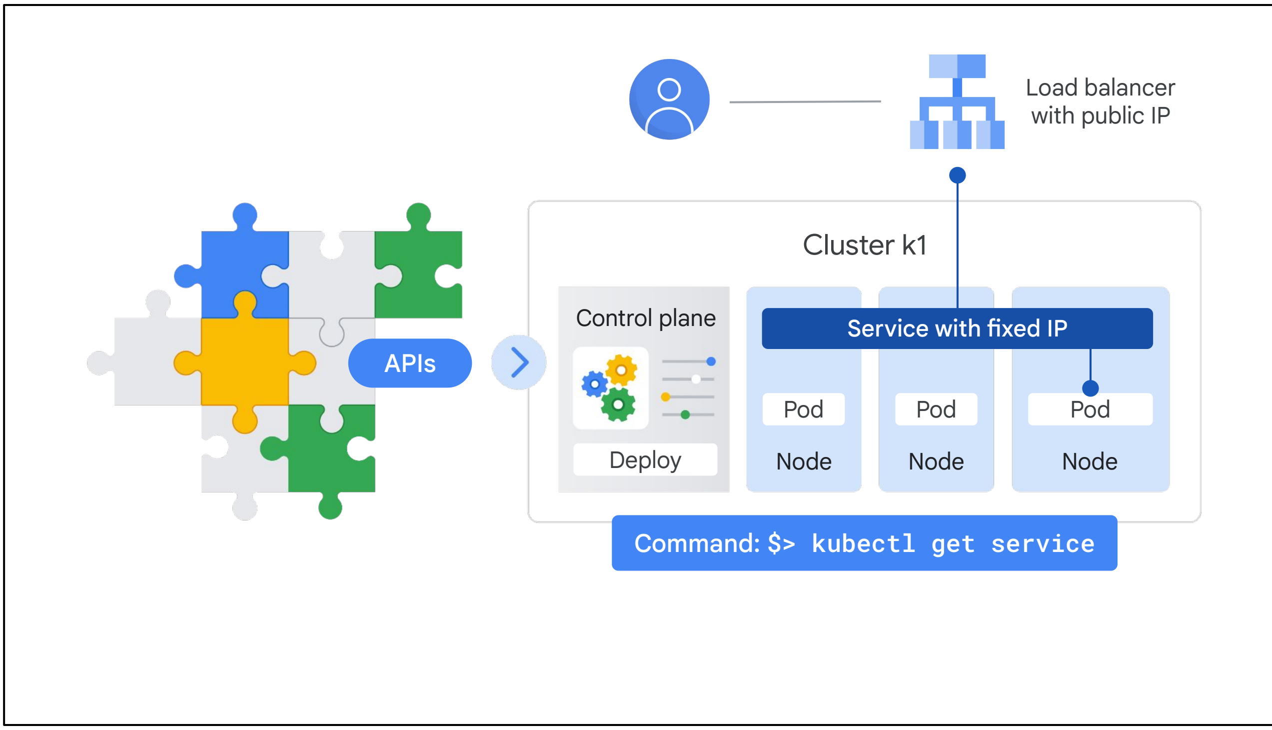
run the command:

```
$ kubectl get pods
```



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.

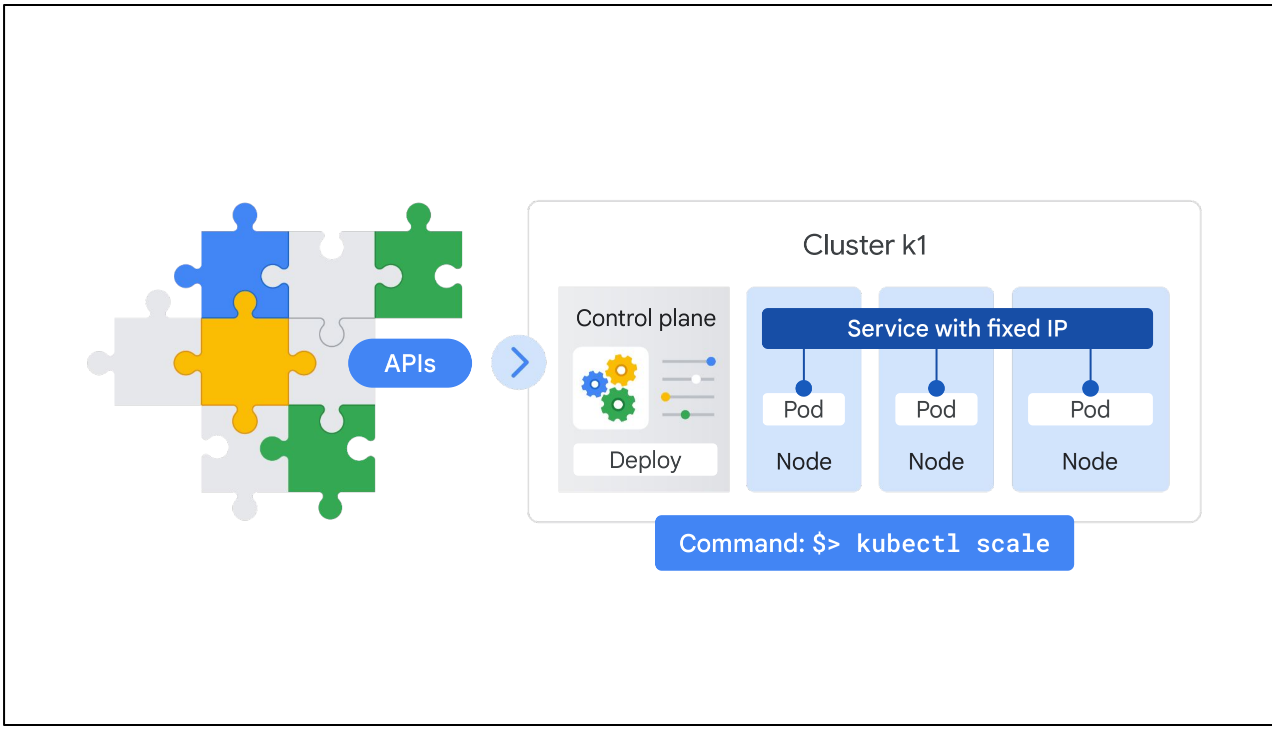


Any client that reaches that IP address will be routed to a Pod behind the Service. A Service is an abstraction which defines a logical set of Pods and a policy by which to access them.

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.

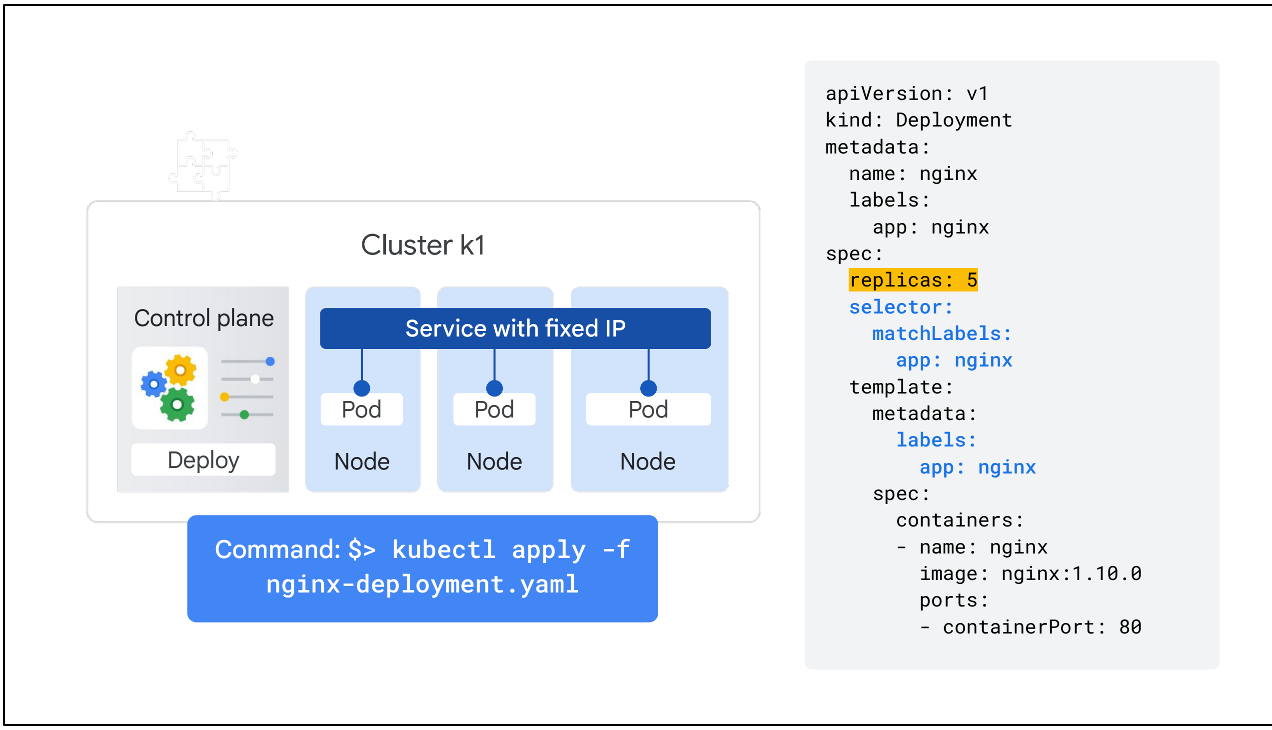
For example, if you create two sets of Pods called `frontend` and `backend` and put them behind their own Services, the `backend` Pods might change, but `frontend` Pods are not aware of this. They simply refer to the backend Service.



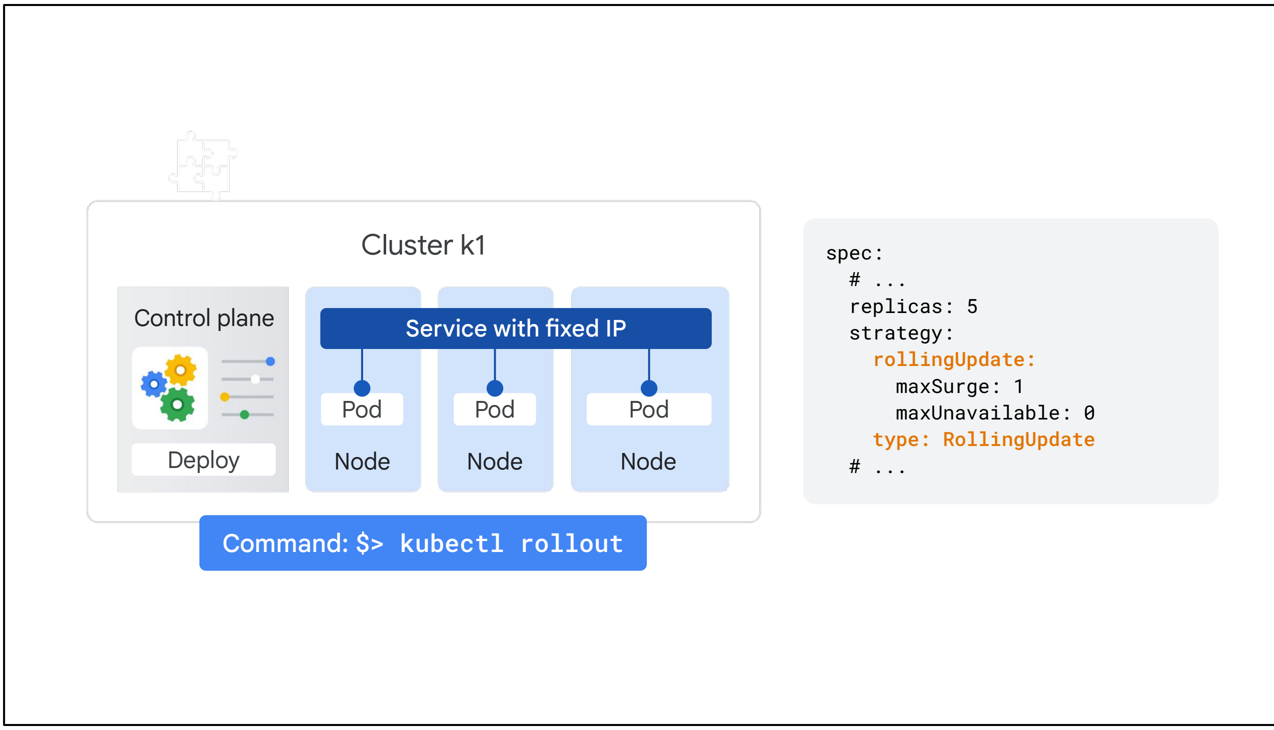
To scale a Deployment, run the `kubectl scale` command.

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.



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.

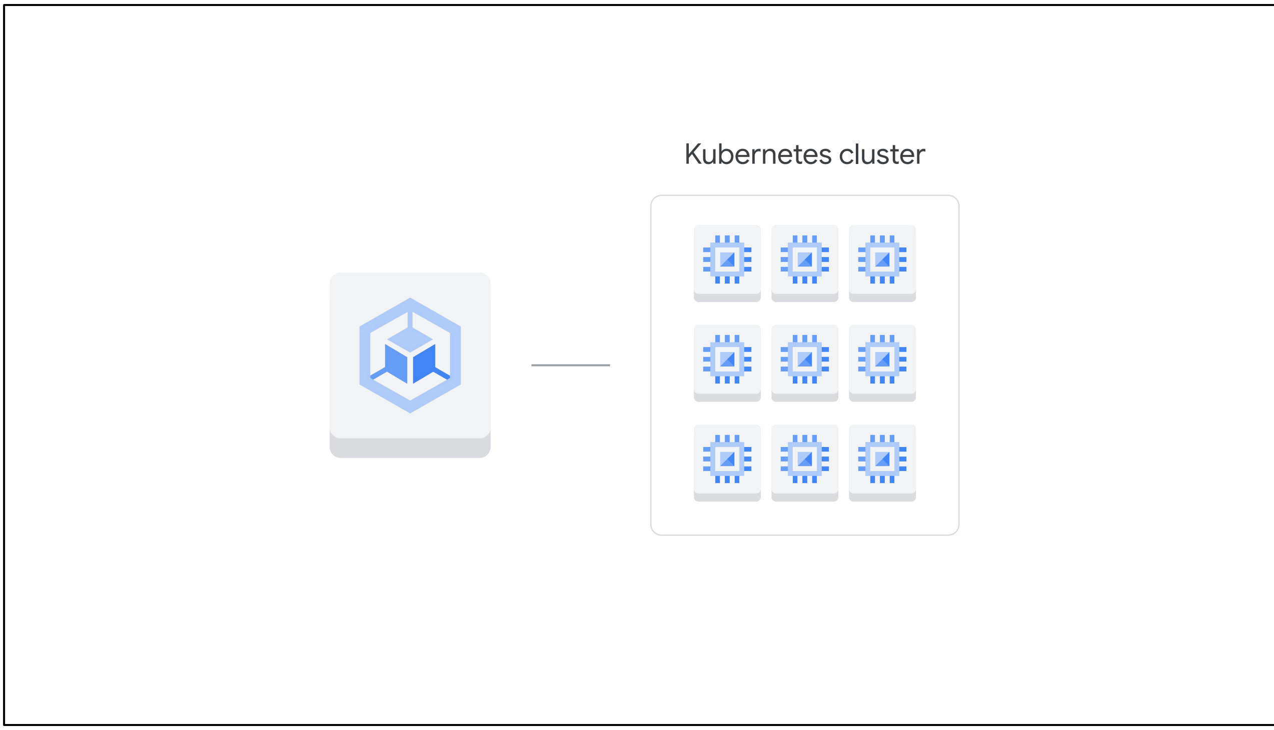


The last question is, what happens when you want to update a new version of your app?

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`.

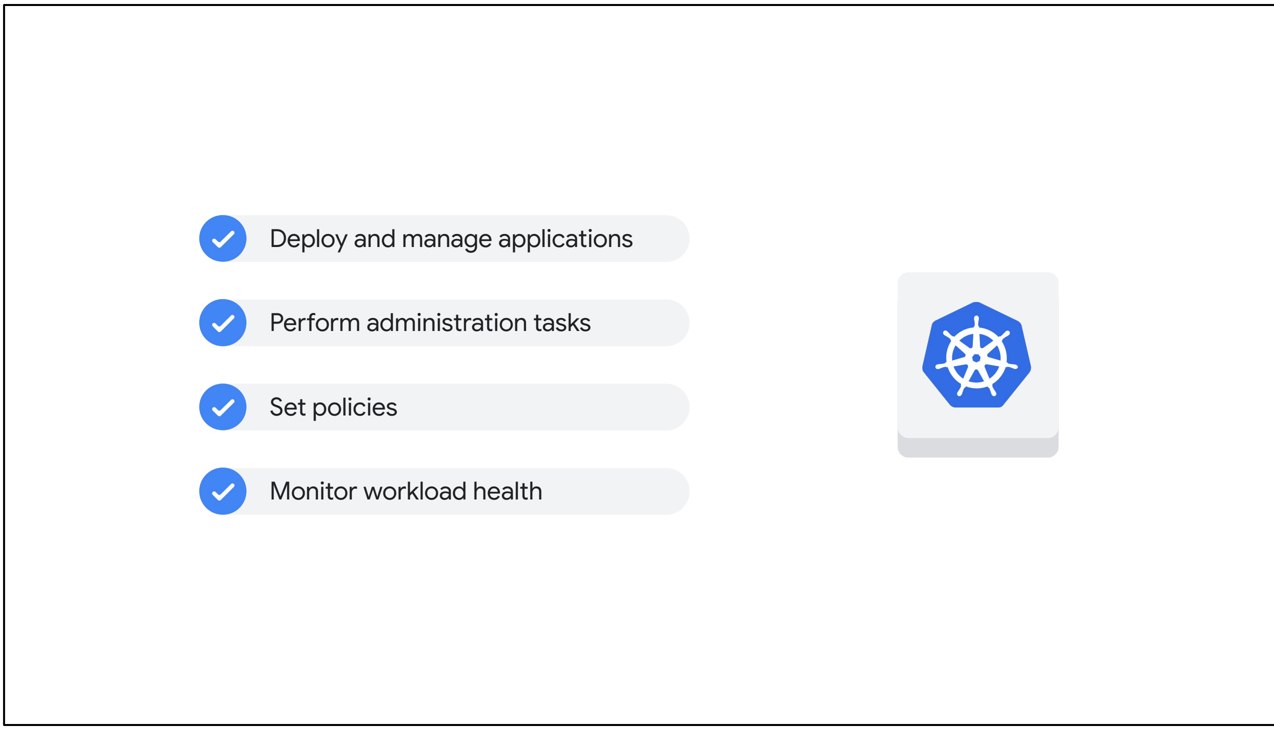
New Pods will then be created according to your new update strategy. Here's an example configuration that will create new version Pods individually and wait for a new Pod to be available before destroying one of the old Pods.



You can create a Kubernetes cluster with Kubernetes Engine

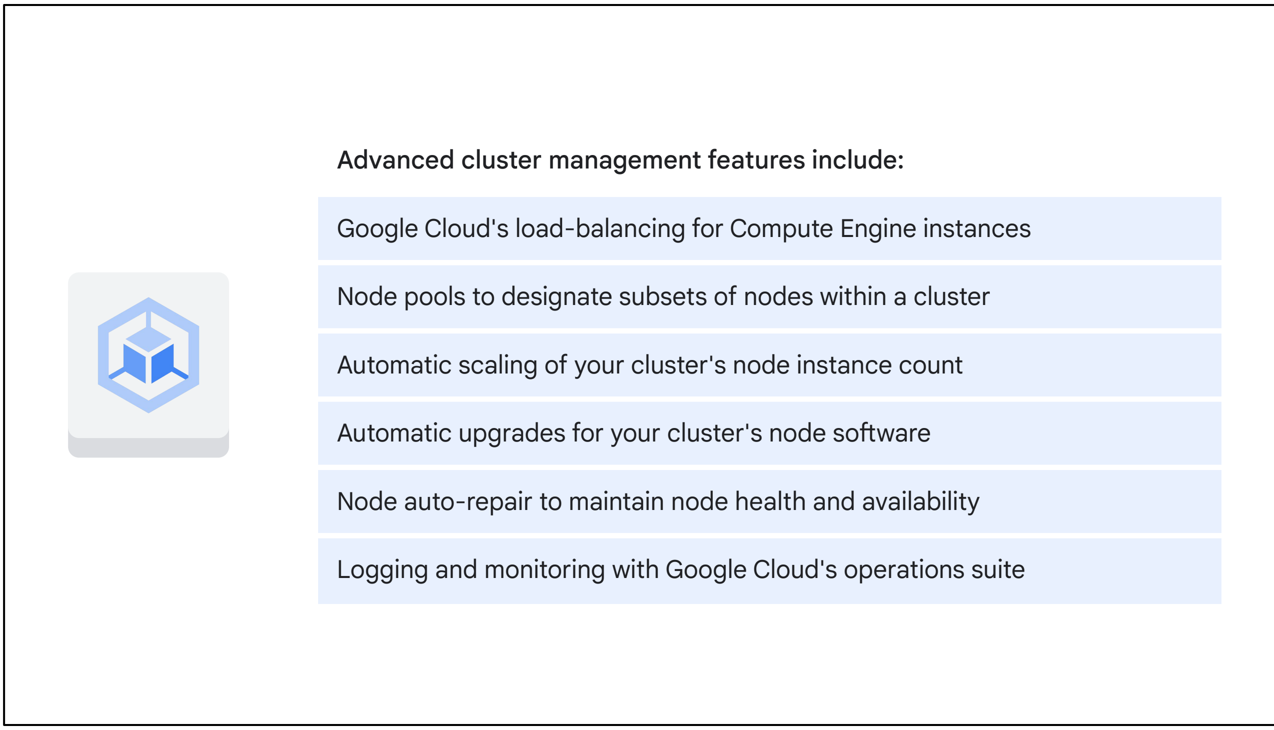
Kubernetes provides the mechanisms through which you **interact with your cluster**

Kubernetes provides the mechanisms through which you interact with your cluster.



Kubernetes commands and resources are used to:

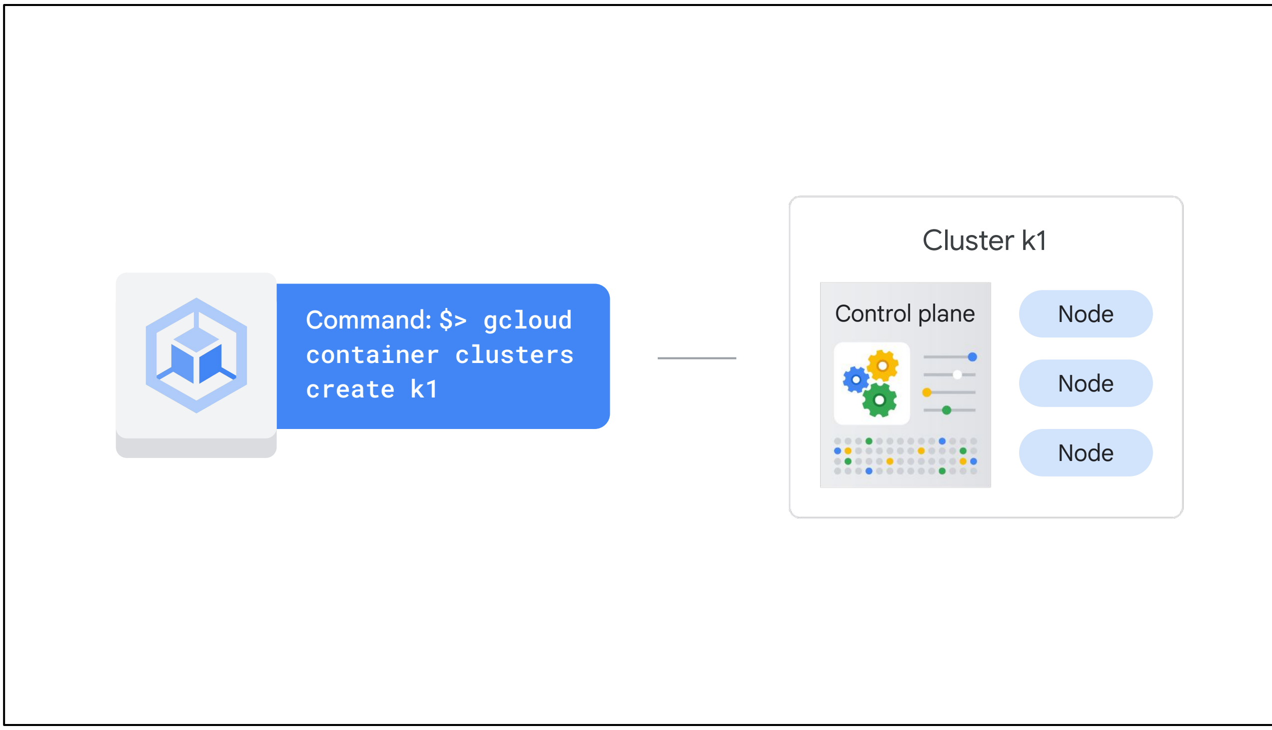
- deploy and manage applications,
- perform administration tasks,
- set policies,
- and monitor the health of deployed workloads.



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

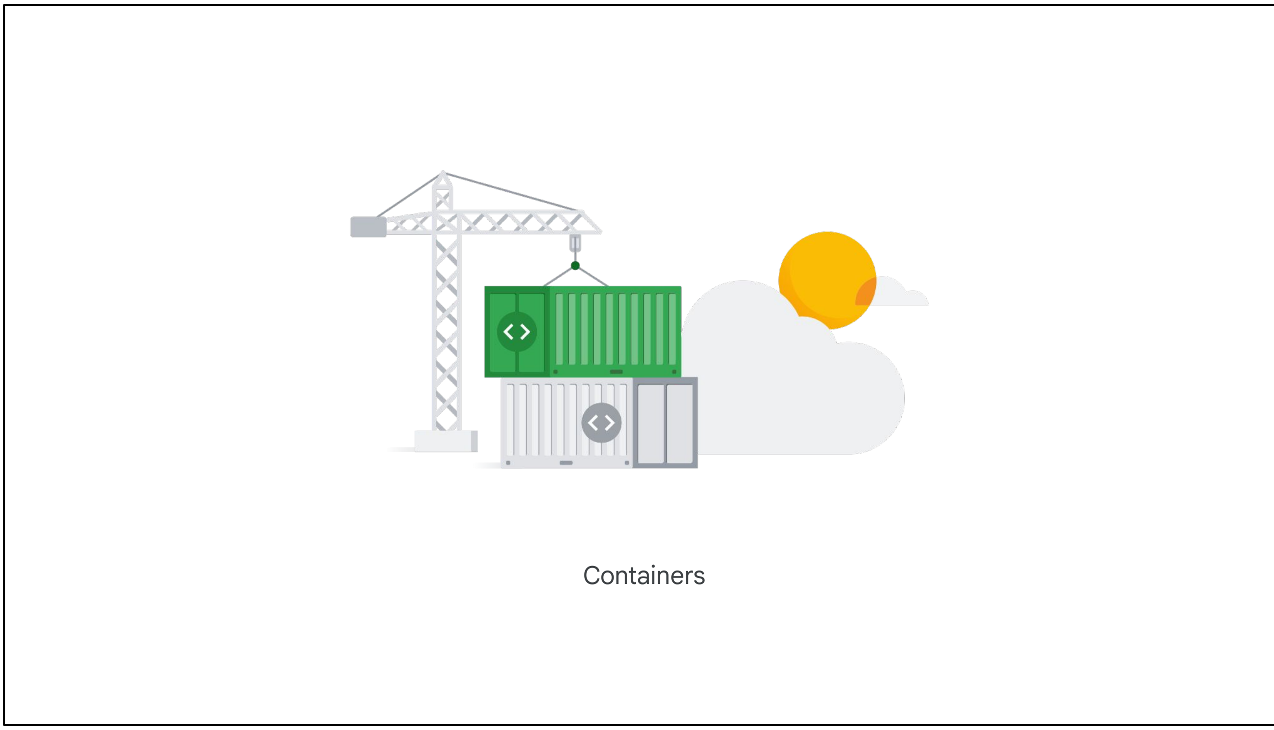
- Google Cloud's load-balancing for Compute Engine instances
- Node pools to designate subsets of nodes within a cluster for additional flexibility
- Automatic scaling of your cluster's node instance count
- Automatic upgrades for your cluster's node software
- Node auto-repair to maintain node health 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 to have if you'll need to bridge your on-prem and cloud resources.

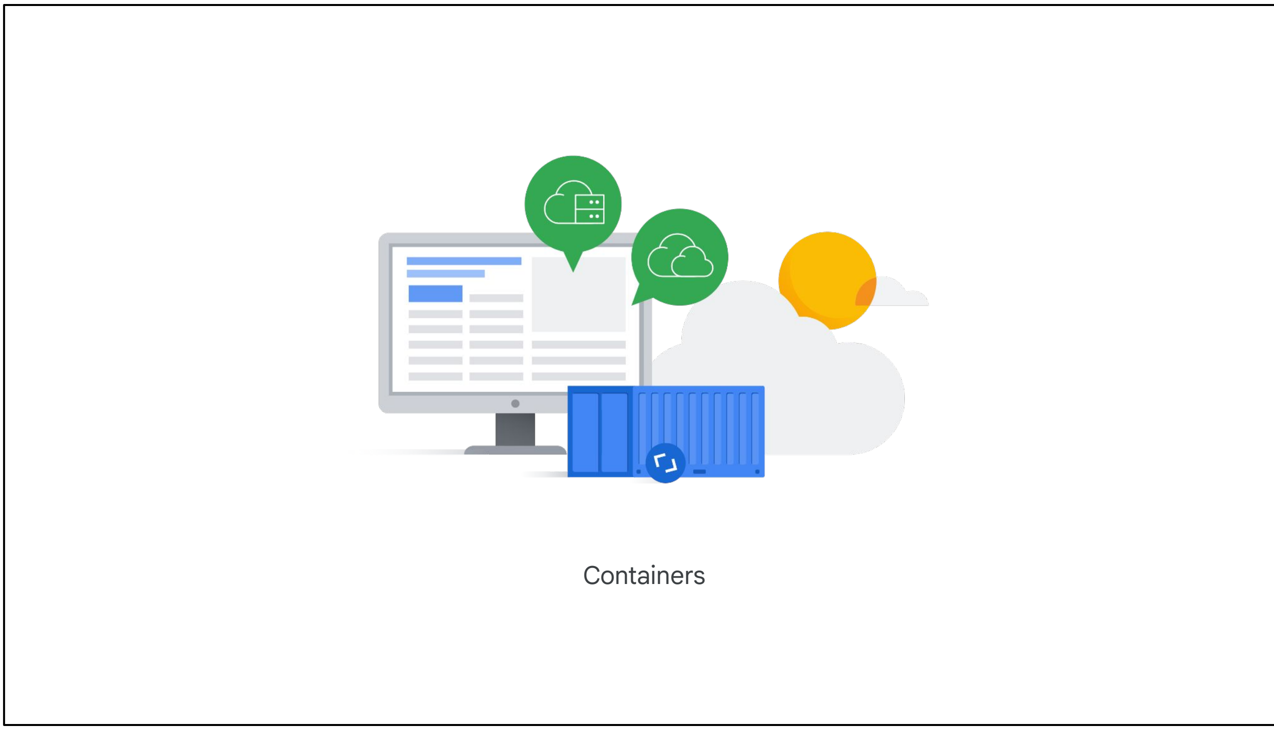


To start up Kubernetes on a cluster in GKE, all you do is run this command:

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



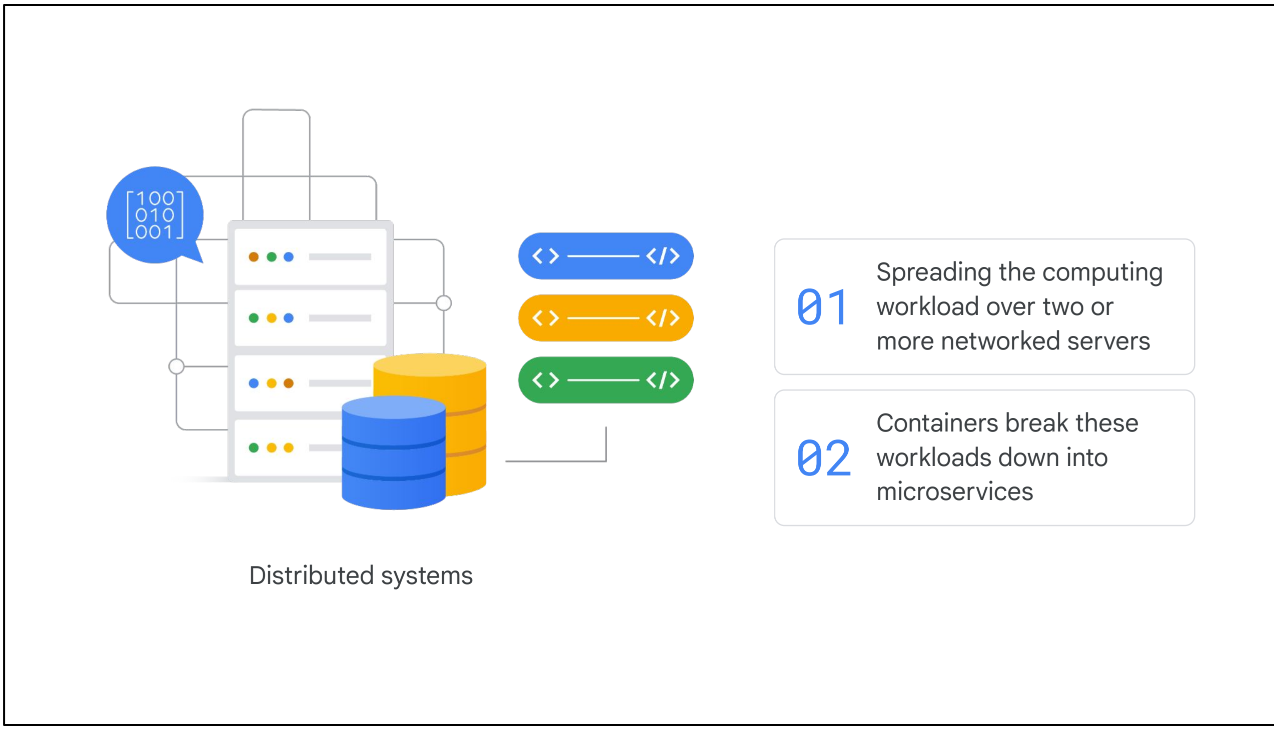
Now that you’ve seen how containers work, we’re going to take that information



a step further and explore how they can be used in a modern hybrid cloud and multi-cloud architecture.

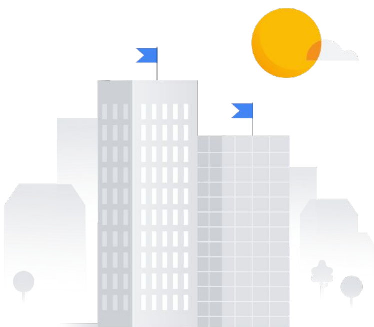
Distributed systems

Let’s begin by looking at a typical on-premises distributed systems architecture, which is



Most enterprise-scale applications are designed as distributed systems, spreading the computing workload required to provide services over two or more networked servers.

In recent years, containers have become a popular way to break these workloads down into “microservices” so they can be more easily maintained and expanded.



Typical on-premises architecture:

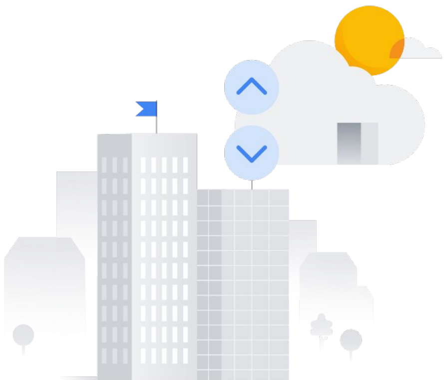
- Systems and workloads are housed on-premises, on high-capacity servers running on company’s network
- Lots of steps required to upgrade on-premises systems
- Upgrade time could be anywhere from several months to one or more years
- Often quite costly, especially considering the useful lifespan of the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

Traditionally, these enterprise systems–and their workloads, containerized or not–have been located on-premises, which means they’re 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 might also be quite costly, especially when you consider the useful lifespan of the average server is only 3-5 years.



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

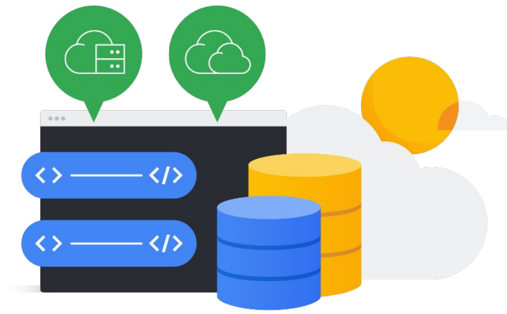
This is where a modern hybrid or multi-cloud architecture can help.

- It allows you to keep parts of your systems infrastructure on-premises while moving other parts to the cloud, thus 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.

- And add specialized services, such as machine learning, content caching, data analysis, long-term storage, and IoT, to your computing resources toolkit.



Hybrid and multi-cloud

Google Cloud

In the next video you'll learn about Google Cloud's answer to modern hybrid and multi-cloud distributed systems and services management.