Containers in the Cloud

Containers in the Cloud

01 Introduction to containers

02 Kubernetes and Google Kubernetes Engine

We've already explore Compute Engine–which is Google Cloud's infrastructure as a service offering, with access to servers, file systems, and networking–and App Engine, which is Google Cloud's platform as a service offering.

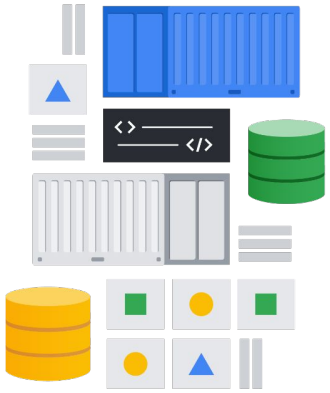
Containers in the Cloud

01 Introduction to containers

02 Kubernetes and Google Kubernetes Engine

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

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

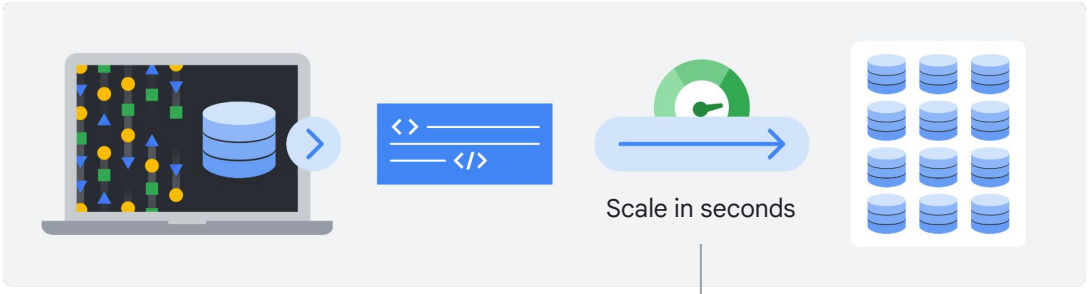
What are containers?

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."

You can scale by duplicating single containers

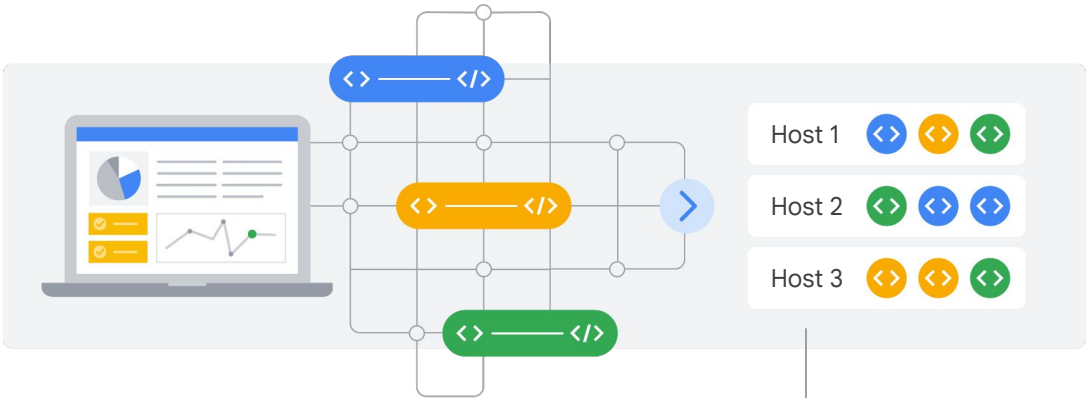


Scaling one container running the whole web server on a single host

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.

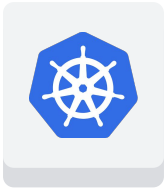
Containers in the Cloud

01 Introduction to containers

02 Kubernetes and Google Kubernetes Engine

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



It's 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 deploy rollouts and rollbacks.

At the highest level, it's a set of APIs to deploy containers on a set of nodes called a cluster.

It's 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.

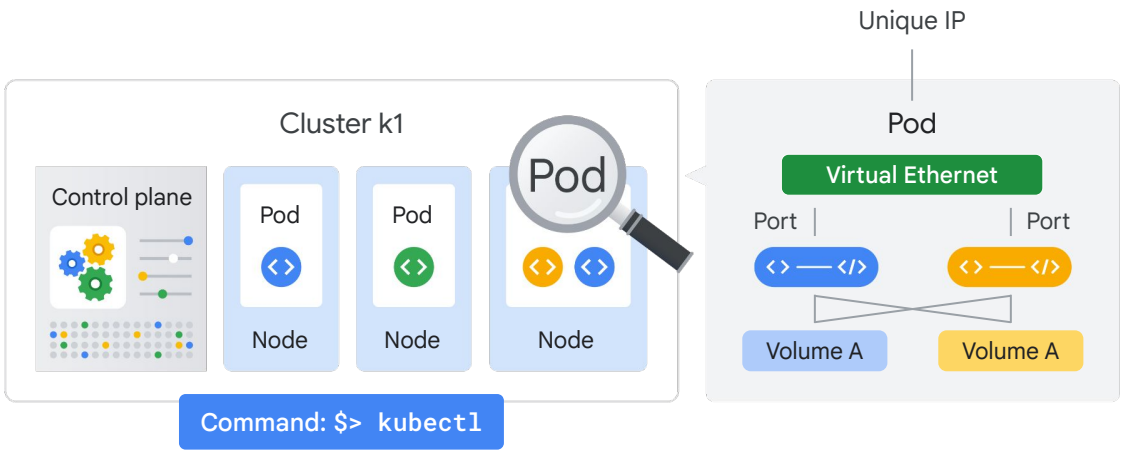
So, what is Kubernetes? Kubernetes is an open-source platform for managing containerized workloads and services. It makes it easy to orchestrate many containers on many hosts, scale them as microservices, and easily deploy rollouts and rollbacks.

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”

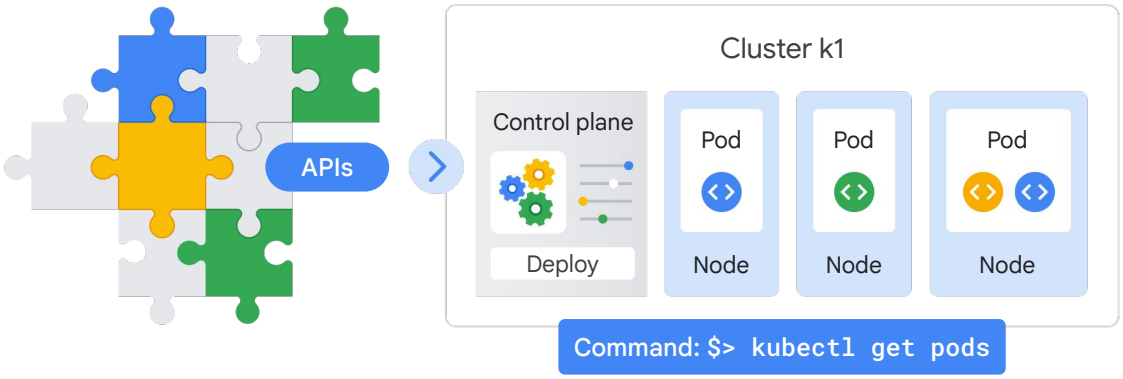


Deploying containers on nodes by using a wrapper around one or more containers is what defines a Pod. A Pod is the smallest unit in Kubernetes that you create or deploy. It represents a running process on your cluster as either a component of your application or an entire app.

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.

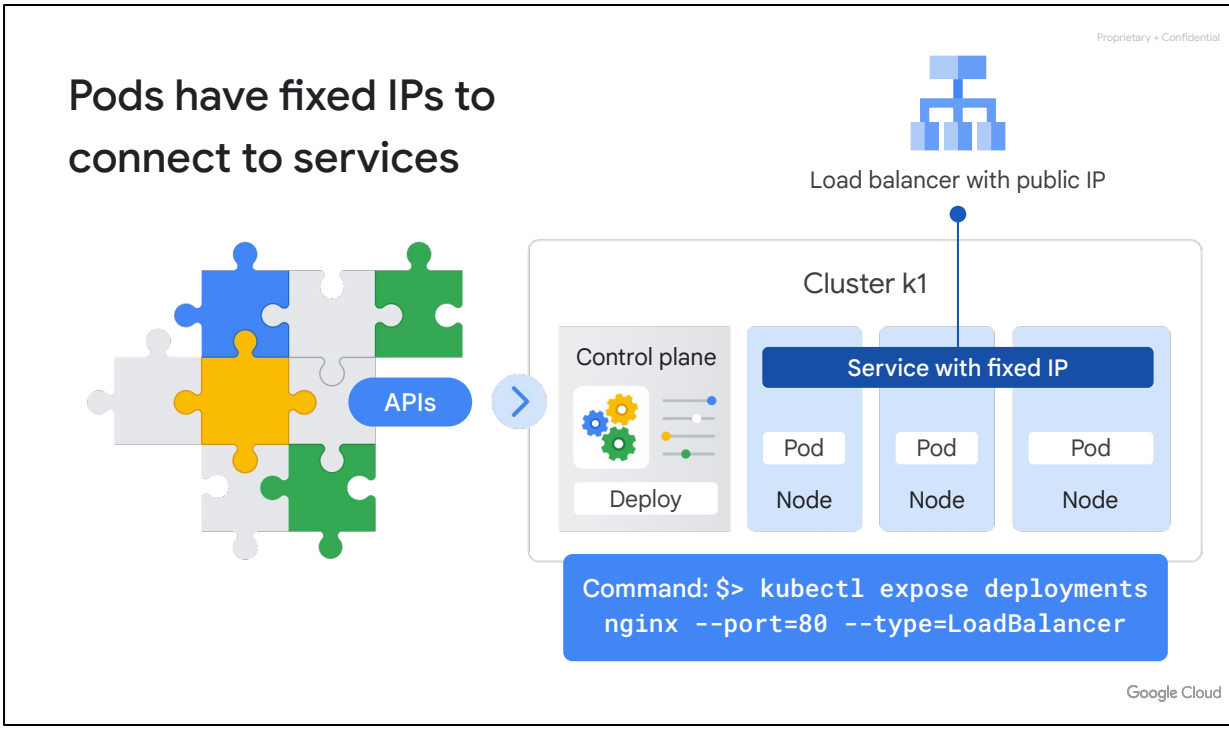
Deployments are replicas of a specific pod



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.

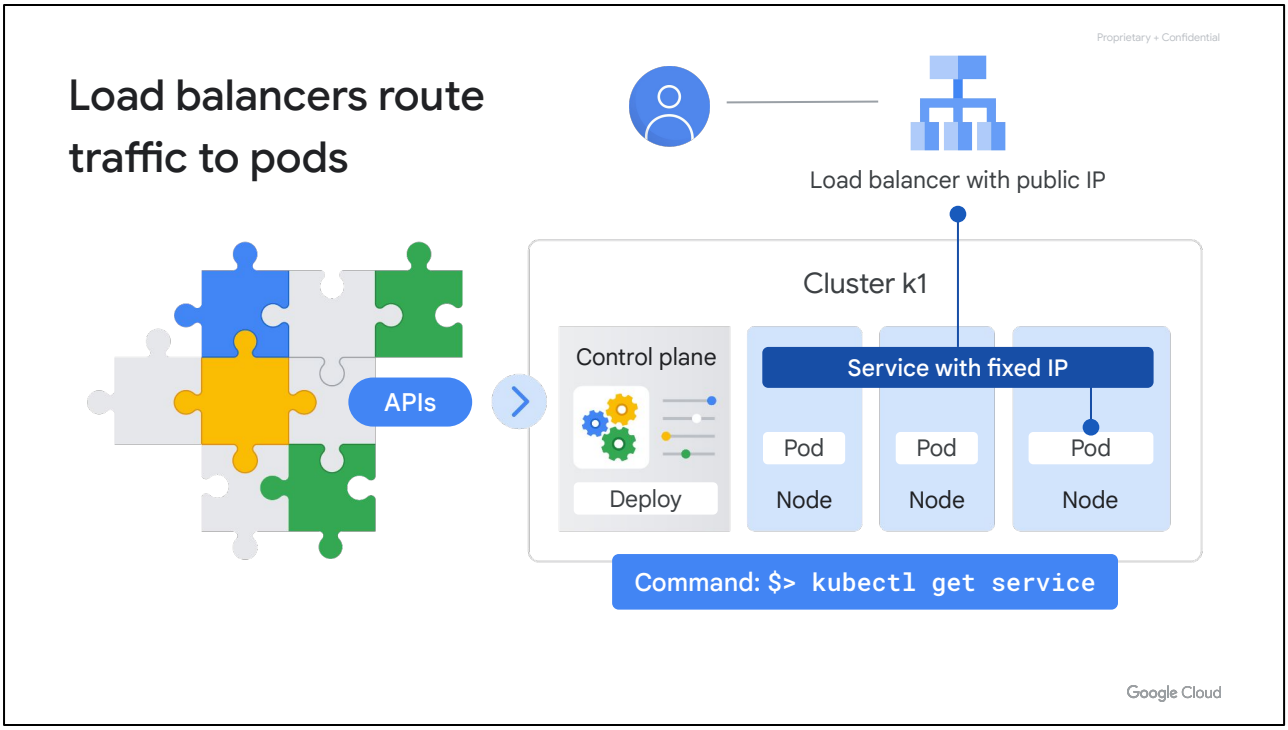
To see a list of the running Pods in your project, run the command:

```
$ 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.



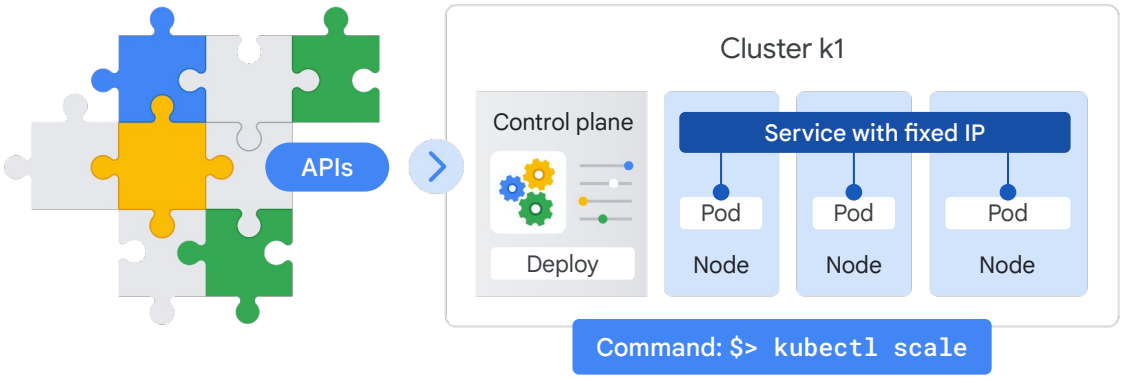
Any client that reaches that IP address will be routed to a Pod behind the Service. A Service is an abstraction which defines a logical set of Pods and a policy by which to access them.

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.

Deployments can be scaled on command

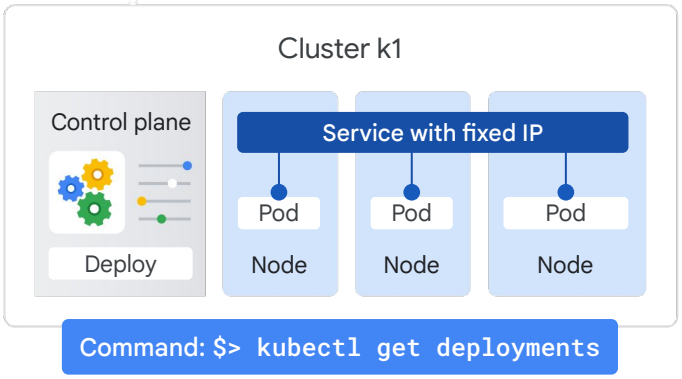


To scale a Deployment, run the `kubectl scale` command.

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
```

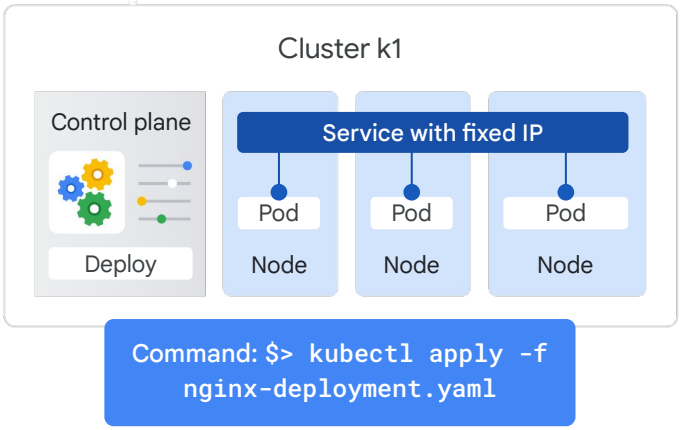

So far, we've seen how to run imperative commands like expose and scale. This works well to learn and test Kubernetes step-by-step.

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.

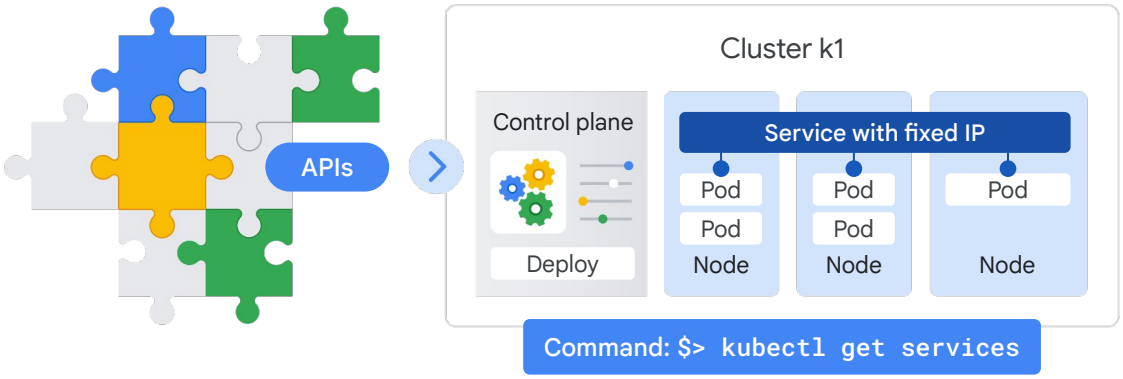
Deployments can be modified by changing configuration files




```
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
```

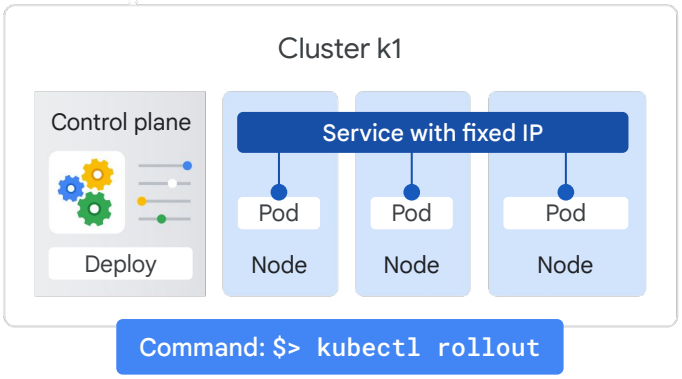

You can check your Deployment to make sure the proper number of replicas is running by using either `kubectl get deployments` or `kubectl describe deployments`. To run five replicas instead of three, all you do is update the Deployment config file and run the `kubectl apply` command to use the updated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

Endpoints can be discovered using a kubectl command



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
  # ...
```

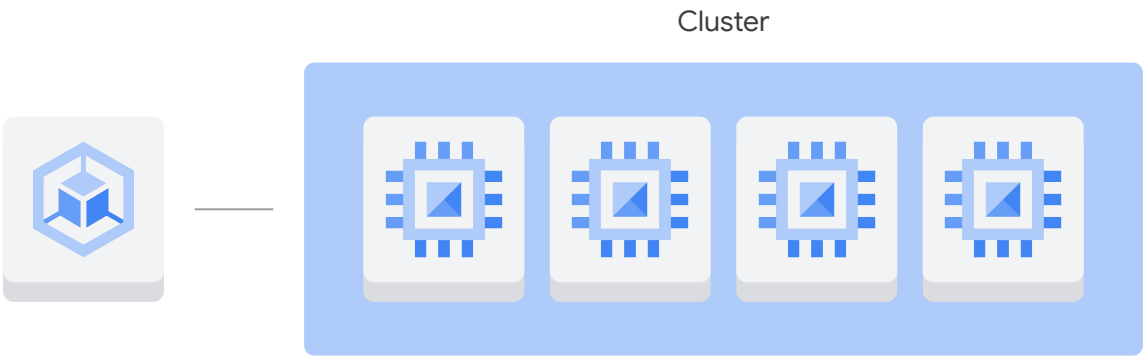

The last question is, what happens when you want to update a new version of your app?

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.

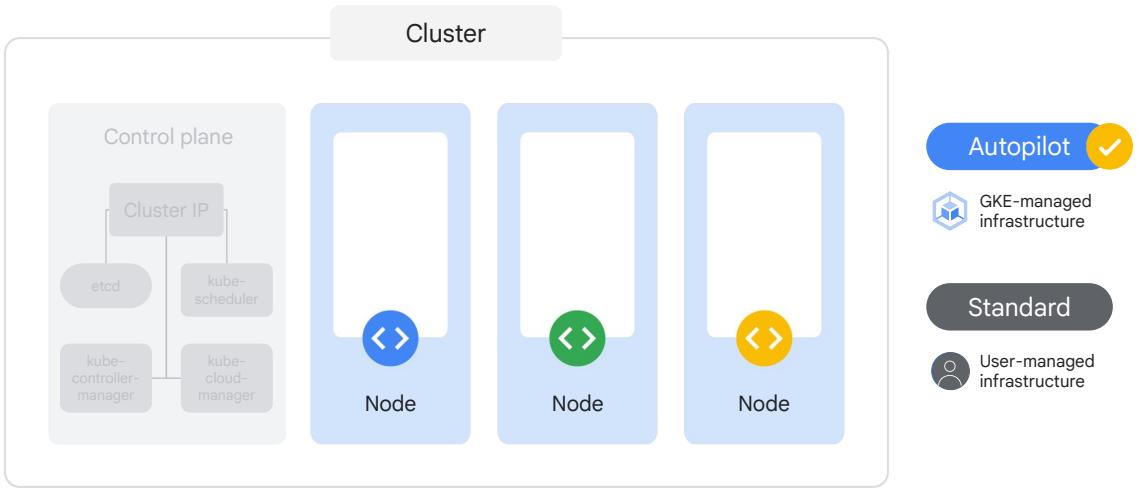
GKE is Google's managed Kubernetes service



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.

Node configuration and management



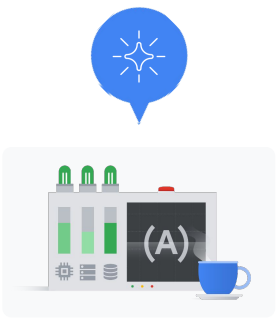
But how is GKE different from Kubernetes? From the user's perspective, it's a lot simpler. GKE manages all the control plane components for us. It still exposes an IP address to which we send all of our Kubernetes API requests, but GKE is responsible for provisioning and managing all the control plane infrastructure behind it. It also eliminates the need for a separate control plane.

Node configuration and management depends on the type of GKE mode you use.

With the **Autopilot mode**, which is recommended, GKE manages the underlying infrastructure such as node configuration, autoscaling, auto-upgrades, baseline security configurations, and baseline networking configuration.

With the **Standard mode**, you manage the underlying infrastructure, including configuring the individual nodes.

Autopilot



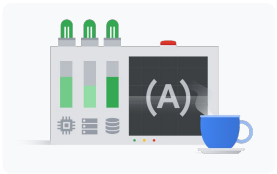
Autopilot

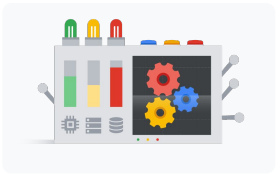
- Optimized for production
- Strong security posture
- Promotes operational efficiency

These are some benefits and functionality of Autopilot:

- It's optimized for **production**.
- Helps produce a strong security posture.
- Promotes operational efficiency.

Standard mode





Autopilot

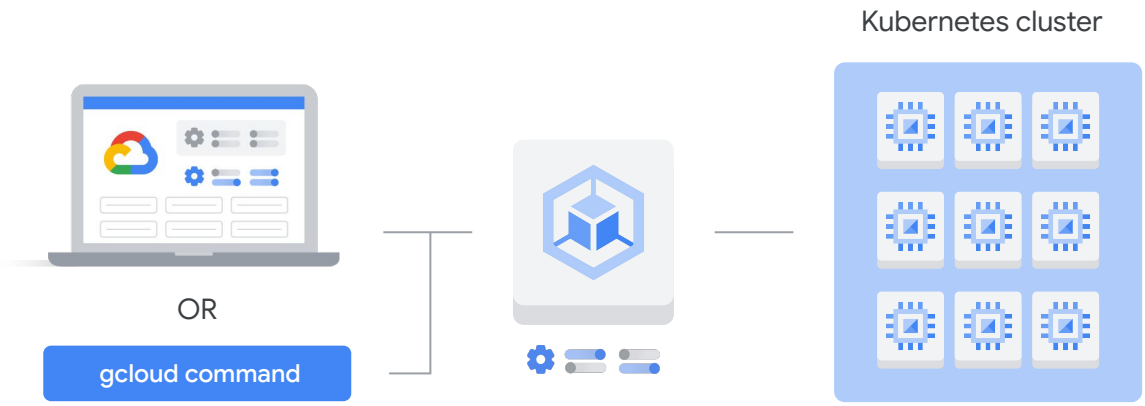
Standard

- Cluster configuration
- Cluster management
- Cluster optimization

The GKE Standard mode has the same functionality as Autopilot, but **you're** responsible for the configuration, management, and optimization of the cluster.

Unless you require the specific level of configuration control offered by GKE standard, it's recommended that you use Autopilot mode.

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.

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 Observability

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

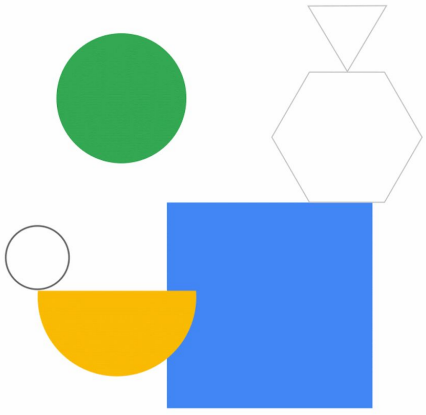
- Google Cloud's load-balancing for Compute Engine instances.
- Node pools to designate subsets of nodes within a cluster for additional flexibility.
- Automatic scaling of your cluster's node instance count.
- Automatic upgrades for your cluster's node software.
- Node auto-repair to maintain node health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

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
```

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

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.