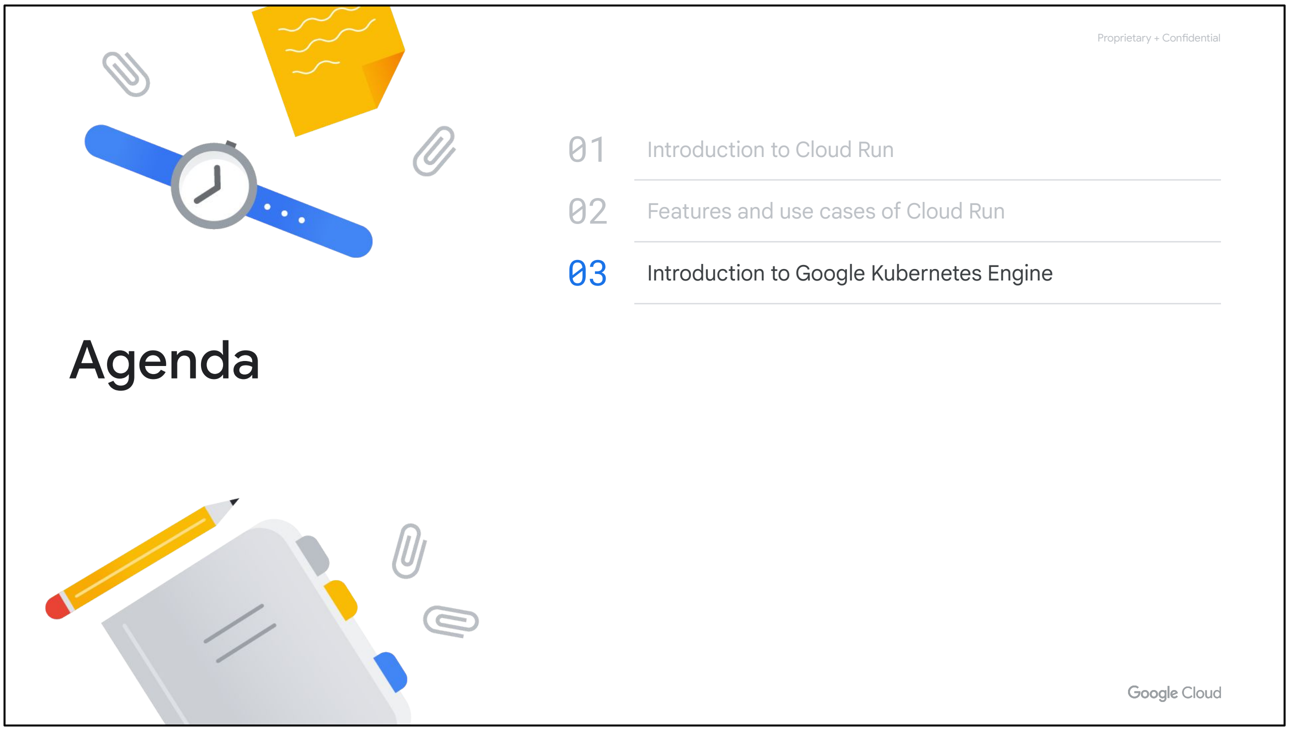
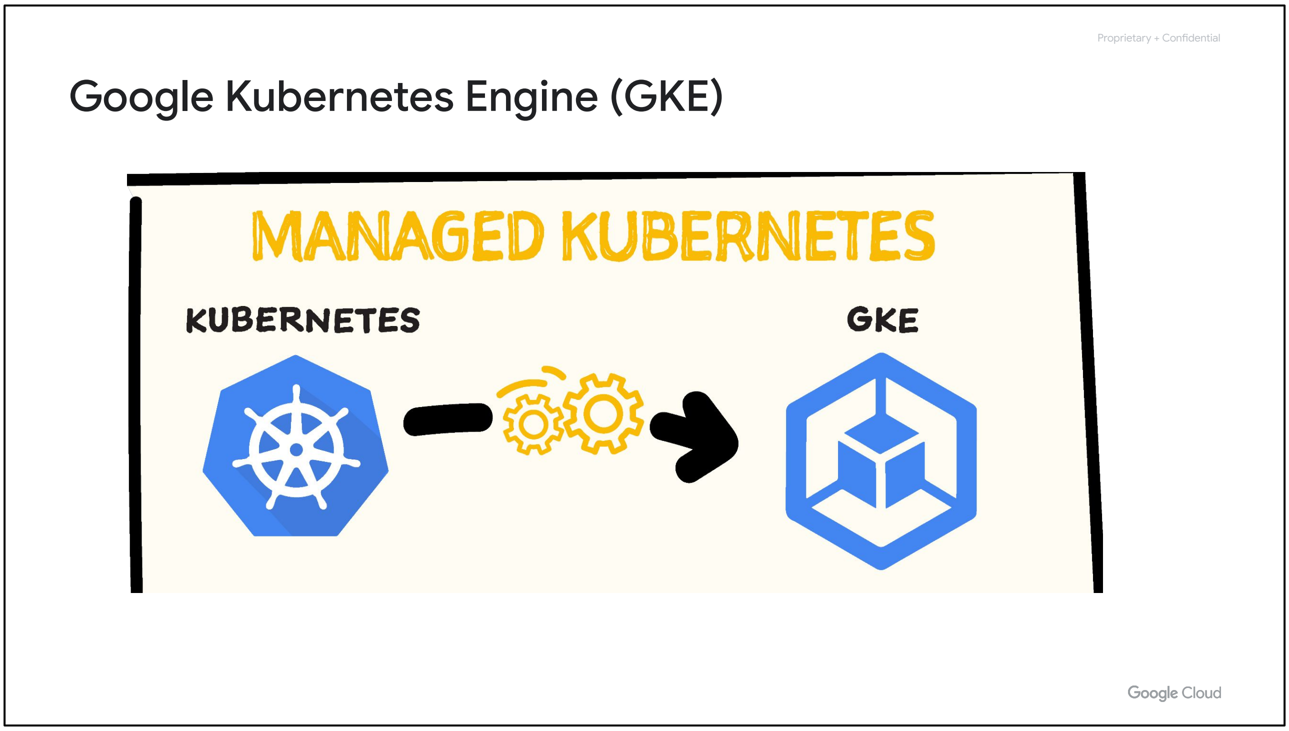


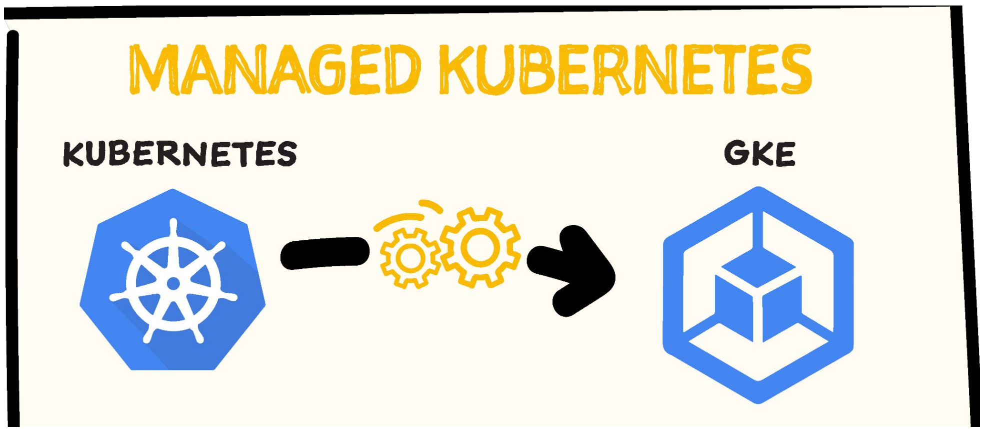
Agenda

Google Cloud

Let’s now discuss what is Google Kubernetes Engine.



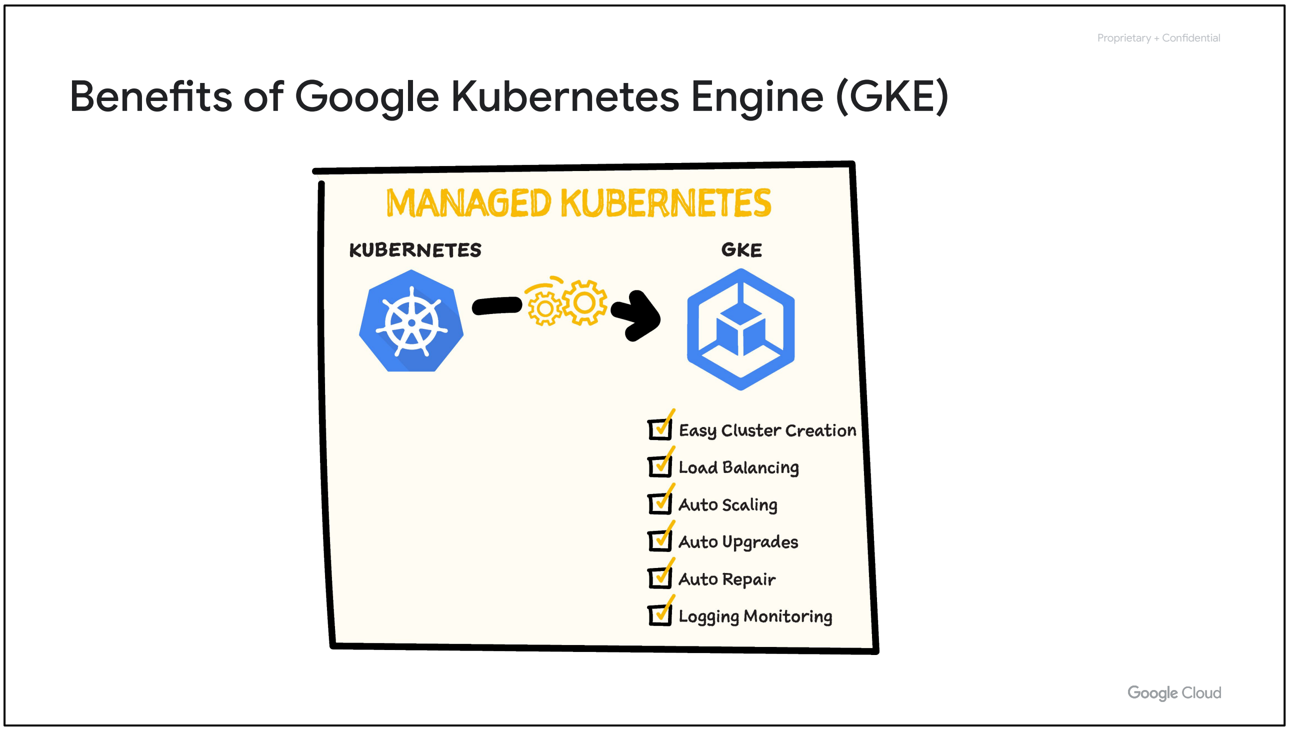
Google Kubernetes Engine (GKE)



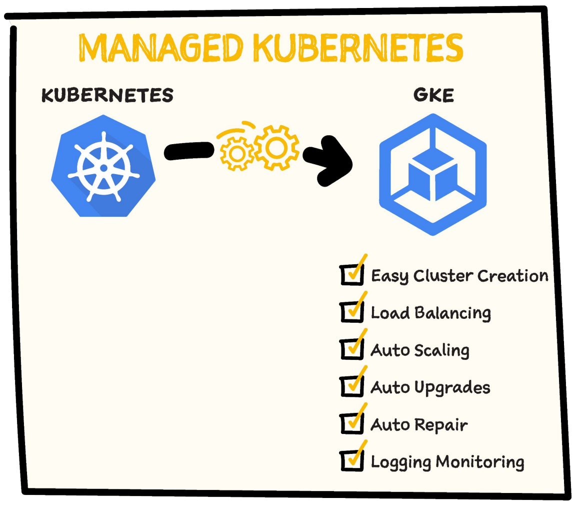
Google Kubernetes Engine (GKE) is a fully managed Kubernetes service. Kubernetes is an open source container orchestration system for automating software deployment, scaling, and management. Originally designed by Google, the project is now maintained by the Cloud Native Computing Foundation (CNCF).

Google Kubernetes Engine provides a managed environment for deploying, managing, and scaling your containerized applications on Google infrastructure.

The GKE environment consists of multiple machines or nodes (specifically, Compute Engine instances) that are grouped together to form a cluster.



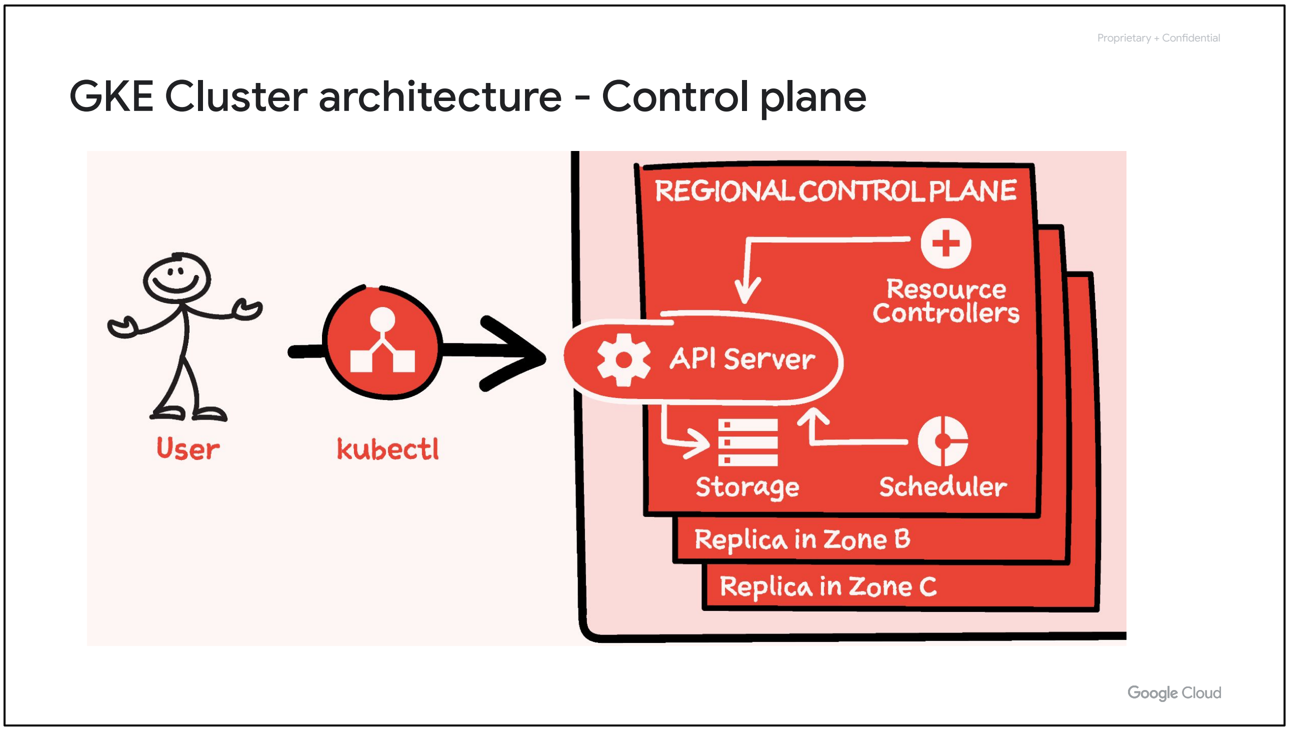
Benefits of Google Kubernetes Engine (GKE)



There is a lot of work that goes into managing a container orchestration system like Kubernetes, from installation and provisioning to upgrades, scaling and meeting service level agreements (SLAs).

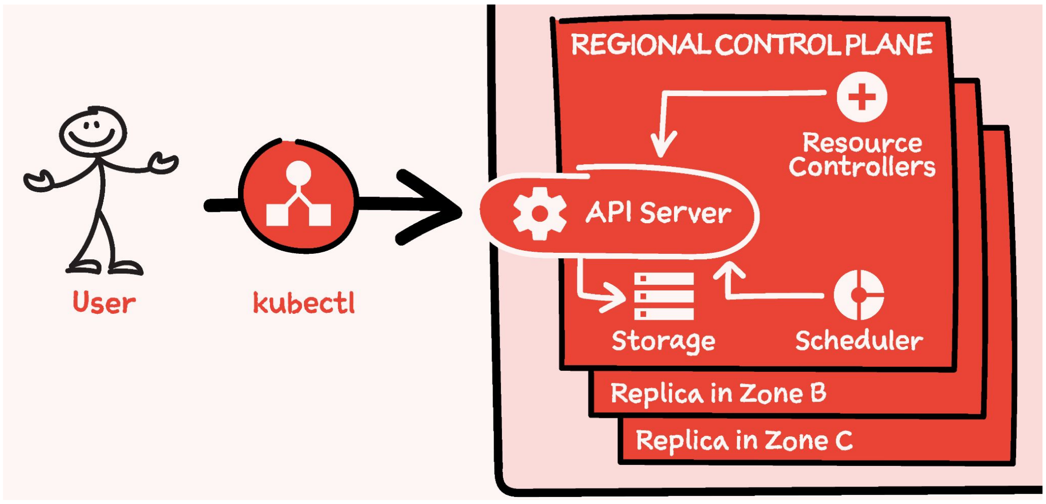
With Google Kubernetes Engine, you gain the benefit of advanced cluster management features that include:

- Easy cluster creation and management.
- Load balancing.
- Automatic scaling.
- Automatic upgrades of your cluster node software.
- Automatic repair to maintain node health and availability.
- Logging and monitoring with Google Cloud's operations suite for cluster visibility.



Proprietary + Confidential

GKE Cluster architecture - Control plane



Google Cloud

A GKE cluster consists of one or more control planes and worker machines called nodes. The control plane and nodes make up the Kubernetes cluster orchestration system. GKE manages the entire underlying infrastructure of clusters, including the control plane, nodes, and all system components.

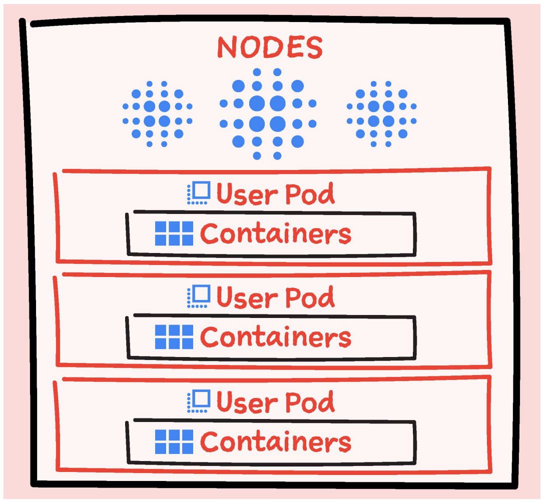
The control plane manages everything that runs on all of the cluster's nodes. The control plane schedules container workloads and manages the workloads' lifecycle, scaling, and upgrades. The control plane also manages network and storage resources for those workloads. The control plane and nodes communicate with each other using Kubernetes APIs.

The control plane is the unified endpoint for your cluster and runs the Kubernetes API server process (kube-apiserver) to handle API requests. To interact with the control plane, you make Kubernetes API calls using:

- HTTP/gRPC requests.
- Command-line clients such as *kubectl*, or the Google Cloud console.

The API server process is the hub for all communication for the cluster. All internal cluster components such as nodes, system processes, and application controllers act as clients of the API server.

GKE Cluster architecture - Nodes



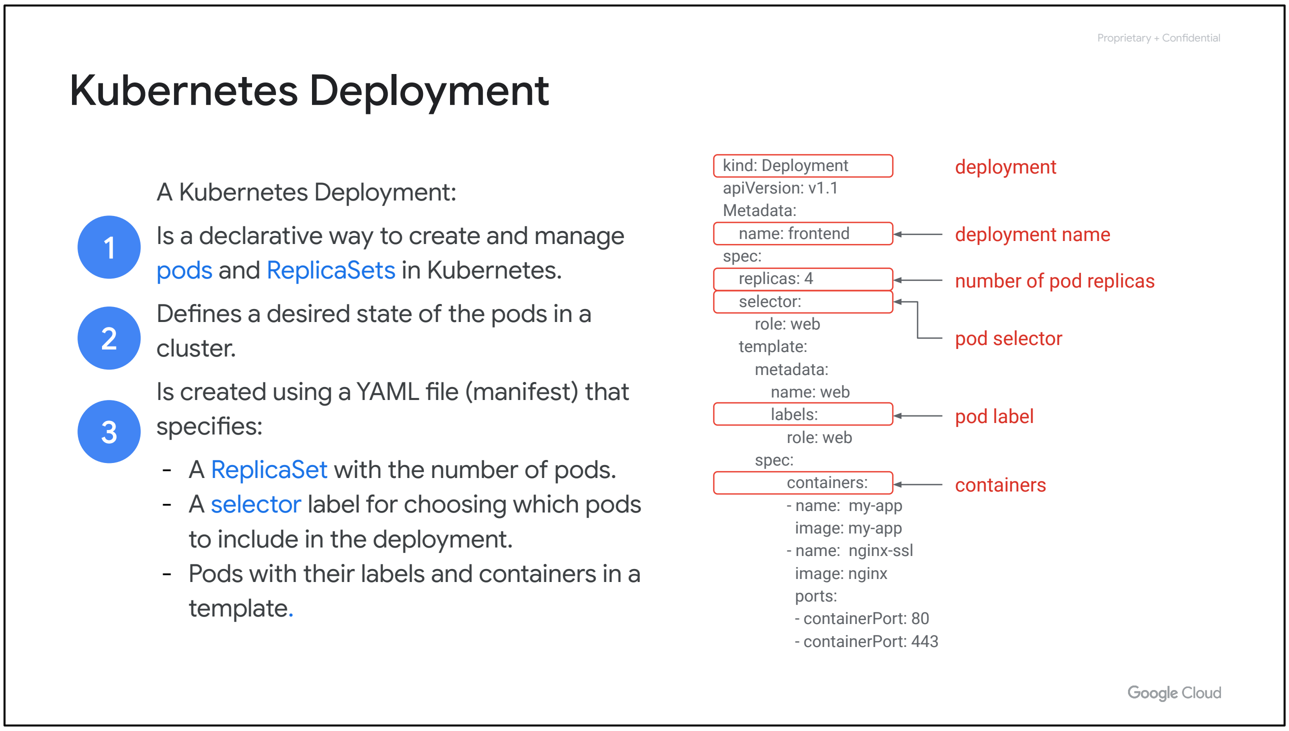
Nodes are Compute Engine virtual machines (VMs) that run your containerized applications and other workloads.

A node runs the services necessary to support the containers that make up your cluster's workloads. These include the runtime and the Kubernetes node agent (kubelet), which communicates with the control plane, and is responsible for starting and running containers that are scheduled on the node.

A pod is the smallest deployable compute unit that you can create and manage in Kubernetes. A pod is a group of one or more containers with shared storage and network resources and a specification for how to run the containers.

Pods are not usually created directly, but instead are created using Kubernetes workload resources such as Deployments or Jobs.

Pods are ephemeral, disposable entities. When a Pod is created, it’s scheduled to run on a node in the cluster. The Pod remains on that node until it finishes execution, the Pod object is deleted, the Pod is evicted for lack of resources, or the node fails.



Proprietary + Confidential

Kubernetes Deployment

A Kubernetes Deployment:

1. Is a declarative way to create and manage pods and ReplicaSets in Kubernetes.
2. Defines a desired state of the pods in a cluster.
3. Is created using a YAML file (manifest) that specifies:
   - A ReplicaSet with the number of pods.
   - A selector label for choosing which pods to include in the deployment.
   - Pods with their labels and containers in a template.

```
kind: Deployment
apiVersion: v1.1
Metadata:
    name: frontend
spec:
    replicas: 4
    selector:
        role: web
    template:
        metadata:
            name: web
            labels:
                role: web
        spec:
                containers:
                - name:  my-app
                  image: my-app
                - name:  nginx-ssl
                  image: nginx
                  ports:
                  - containerPort: 80
                  - containerPort: 443
```

With Kubernetes, you make API requests to specify the *desired state* for the objects in your cluster. Kubernetes attempts to constantly maintain that state.

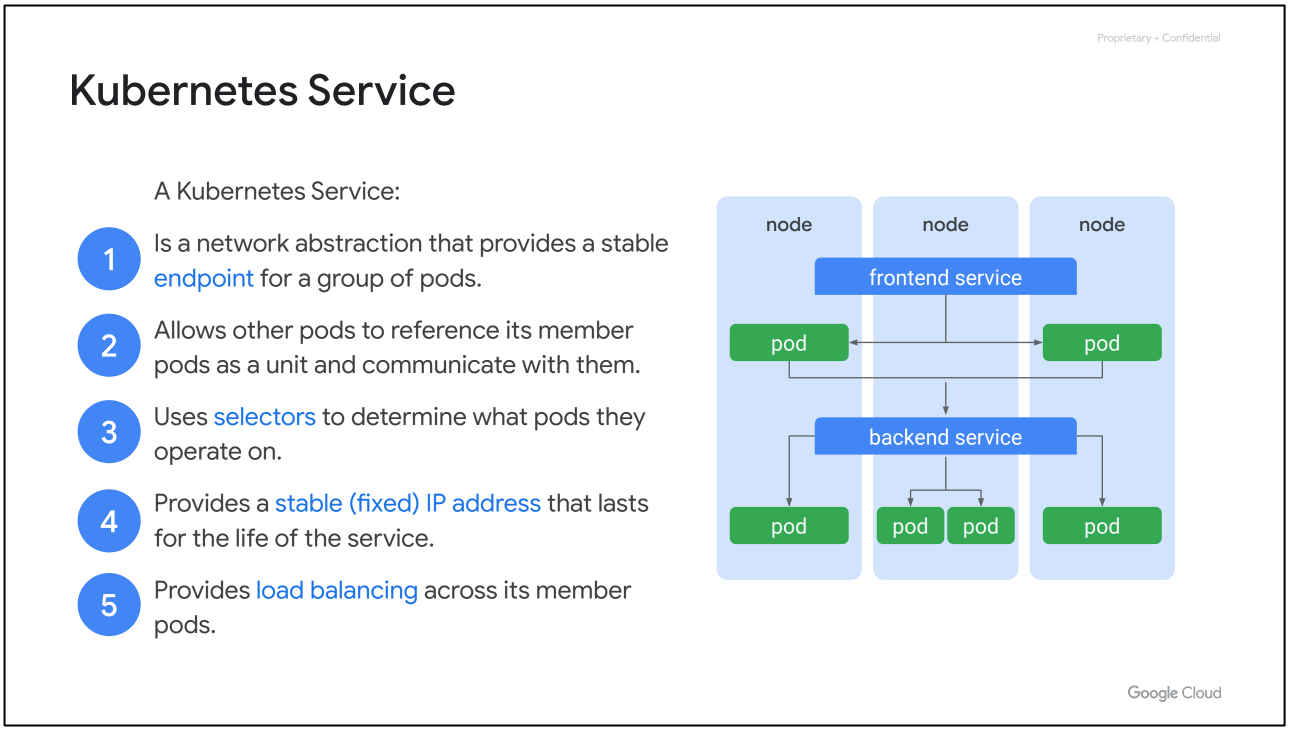
Kubernetes lets you configure objects in the API either imperatively or declaratively.

A Deployment is a declarative way to create and manage pods in Kubernetes. It defines a ReplicaSet that specifies the desired number of pod replicas needed. The purpose of a ReplicaSet is to maintain a stable set of replica pods running at any given time.

The Deployment Controller in Kubernetes changes the actual state of the deployment to the desired state at a controlled rate.

A deployment is defined using a YAML file that specifies the desired number of pods and a selector label that identifies the pods to be included in the deployment. It also includes a specification that identifies the containers that will run in the pod.

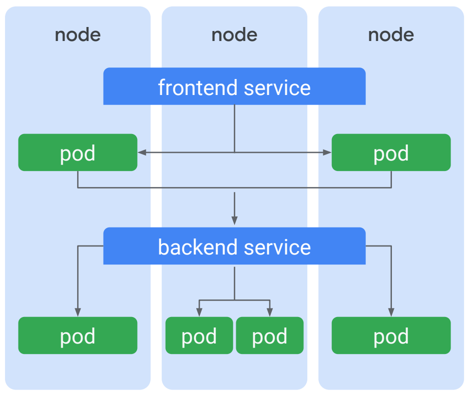
The example Deployment manages 4 pod replicas with the role label set to “web.”



Kubernetes Service

A Kubernetes Service:

1. Is a network abstraction that provides a stable endpoint for a group of pods.
2. Allows other pods to reference its member pods as a unit and communicate with them.
3. Uses selectors to determine what pods they operate on.
4. Provides a stable (fixed) IP address that lasts for the life of the service.
5. Provides load balancing across its member pods.

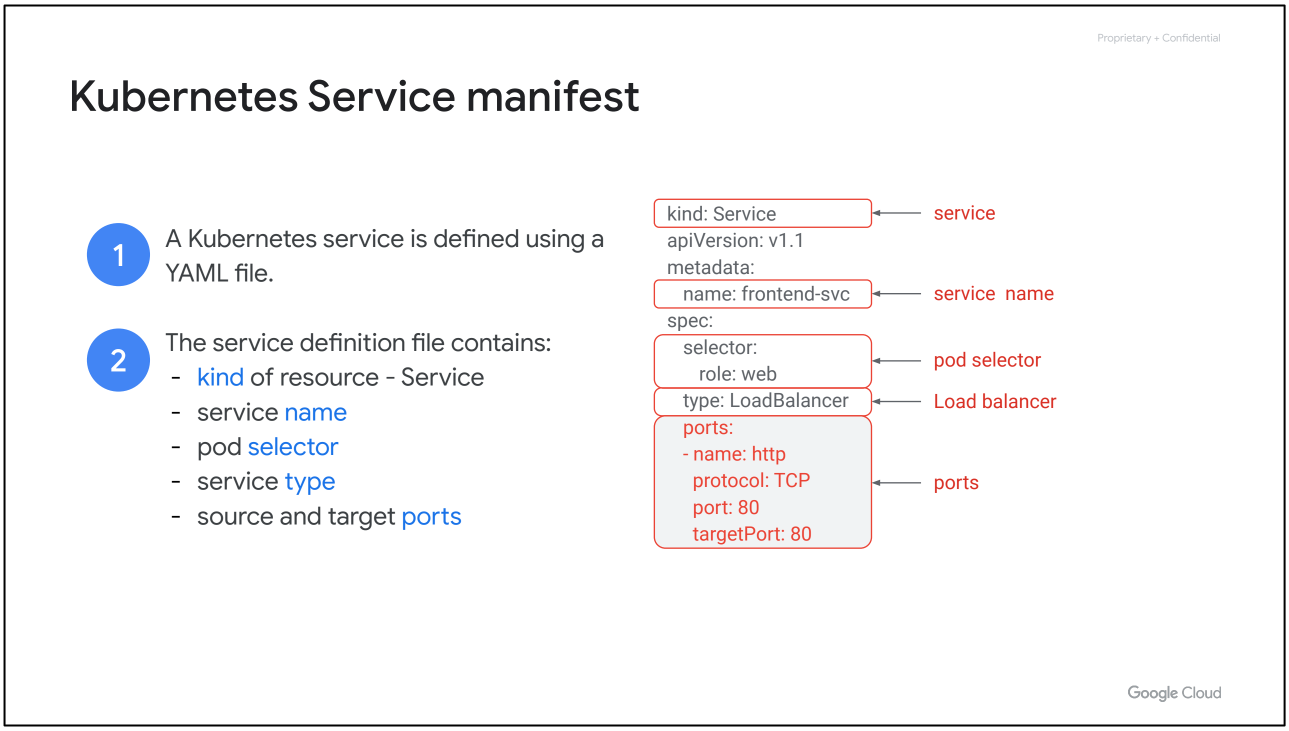


In Kubernetes, a service is a network abstraction that defines a logical set of pods and a policy by which to access them.
The set of pods targeted by a service is usually determined by a selector.

A service has a fixed IP address that lasts for the life of the service, even as the IP addresses of it’s member pods change.
Because a pod is ephemeral, its IP address changes as it is deleted and re-created. Therefore it doesn't make sense to use Pod IP addresses directly.

Clients call the service IP address instead, and their requests are load-balanced across the pods that are members of the service.

The example shows a frontend service and a backend service of an application that is running on Kubernetes cluster. The member pods of the frontend service communicate with the pods in the backend service using its’ fixed IP address.



Kubernetes Service manifest

1. A Kubernetes service is defined using a YAML file.
2. The service definition file contains:
   - kind of resource - Service
   - service name
   - pod selector
   - service type
   - source and target ports

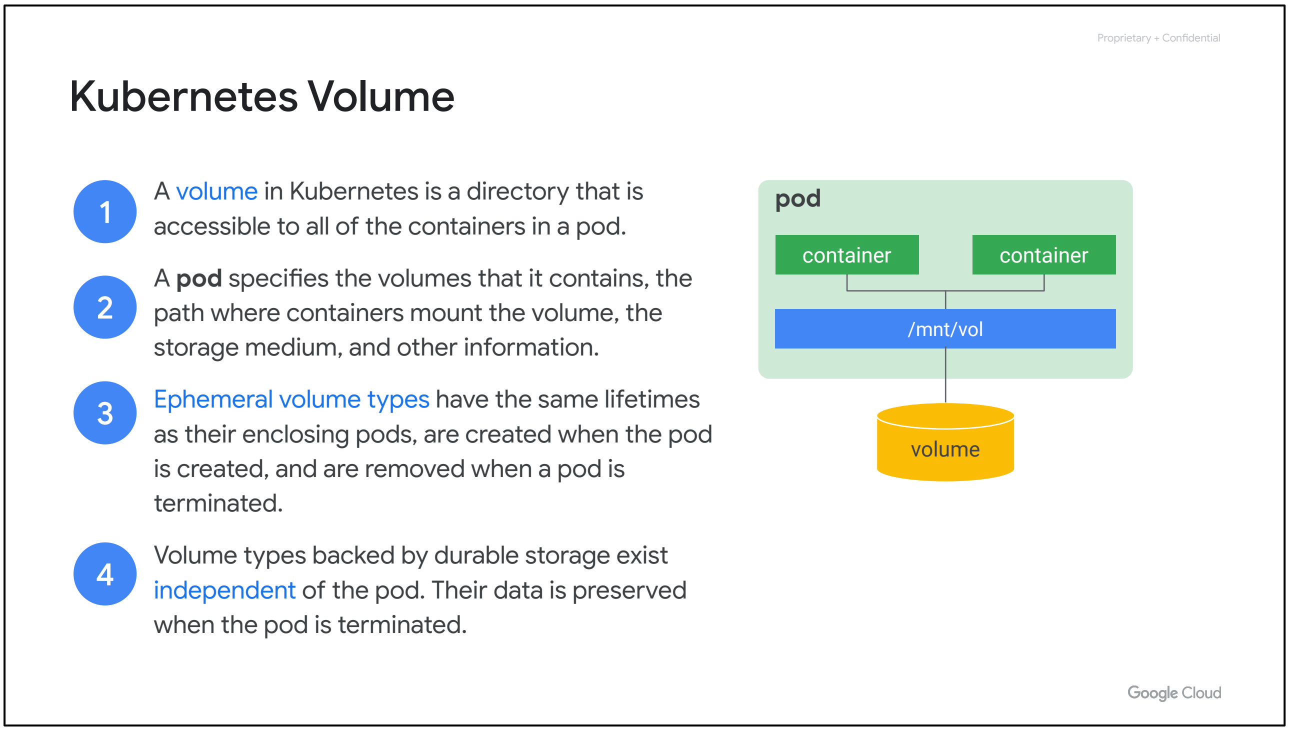
```
kind: Service
apiVersion: v1.1
metadata:
  name: frontend-svc
spec:
  selector:
    role: web
  type: LoadBalancer
  ports:
  - name: http
    protocol: TCP
    port: 80
    targetPort: 80
```

Like other Kubernetes resources, a service is defined in a manifest file in yaml format.

In the example shown, the kind of resource is specified as a Service. The service is given a name; in this example it's named frontend-svc.
The service also has a selector set to the role: web. This selects all pods that have a role label set to web to be part of the frontend service.

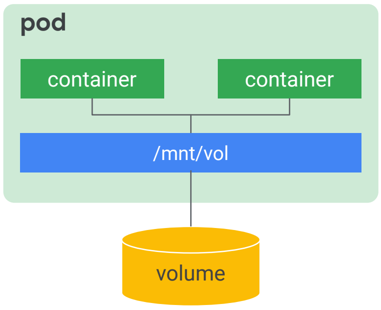
The service type defined is LoadBalancer. A Load balancer service is externally accessible and can be reached by clients that know the DNS name or IP address of the service. In addition to LoadBalancer, two other types of services that are commonly used in Kubernetes are ClusterIP and NodePort. For a full list of service types view the documentation.

External clients call the service by using the load balancer's IP address and the TCP port specified by port value in the yaml. The request is forwarded to one of the member pods on the TCP port specified by the targetPort value.



Kubernetes Volume

1. A volume in Kubernetes is a directory that is accessible to all of the containers in a pod.
2. A **pod** specifies the volumes that it contains, the path where containers mount the volume, the storage medium, and other information.
3. Ephemeral volume types have the same lifetimes as their enclosing pods, are created when the pod is created, and are removed when a pod is terminated.
4. Volume types backed by durable storage exist independent of the pod. Their data is preserved when the pod is terminated.

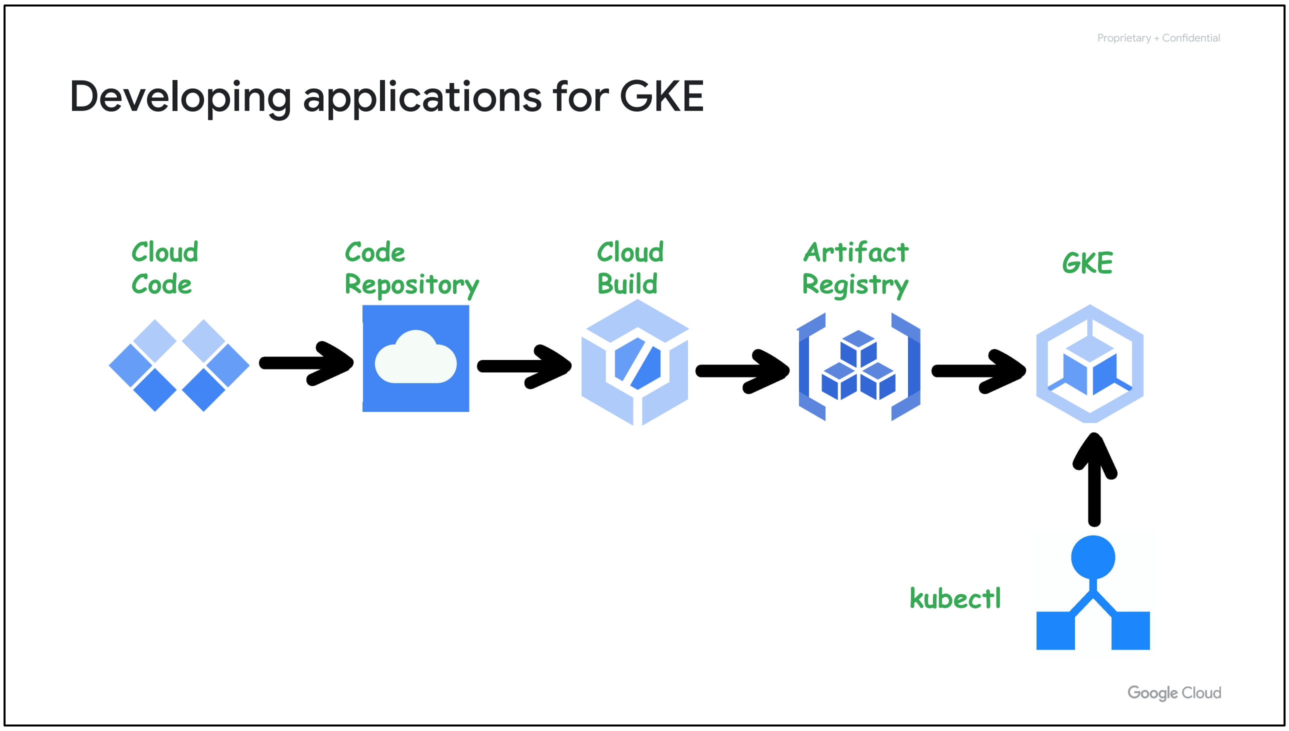


A Kubernetes Volume is a directory that is accessible to all of the containers in a pod. To use a volume, a pod specifies what volumes to provide for the pod and where to mount those into containers. After the volume is mounted, the containers in the pod are brought online and the rest of pod initialization is completed.

There are many different types of volumes in Kubernetes. ConfigMaps and Secrets, are types of volumes that are coupled to the life of the pod and cease to exist when the pod ceases to exist. Alternatively, a PersistentVolume has a lifecycle of its own, independent of the pod.

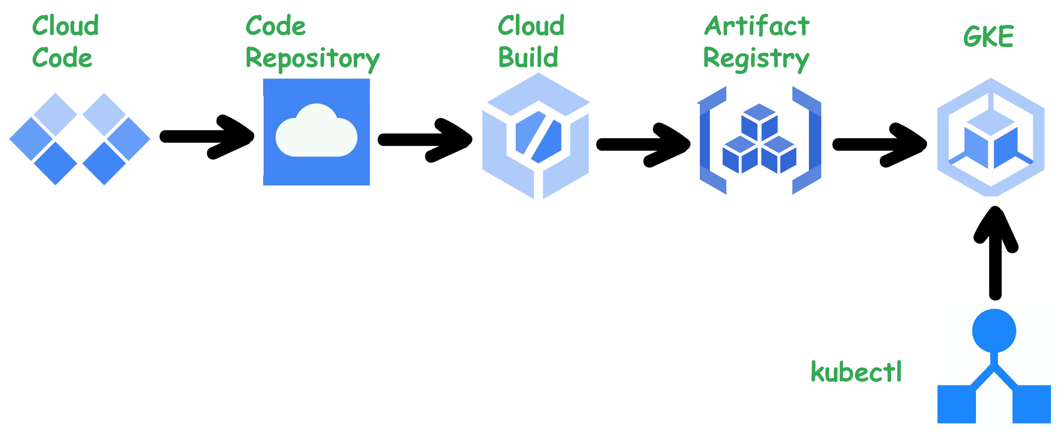
For full list of volume types and more information on their usage in GKE, view the documentation.

In addition to the Deployment, ReplicaSet, Service, Volume resource types, Kubernetes supports the DaemonSet, StatefulSet, Jobs, and others, including custom resource types. To learn more about these resource types, refer to the Kubernetes documentation.



Proprietary + Confidential

Developing applications for GKE



Google Cloud

Now that you have some knowledge of what Kubernetes and GKE is, let’s discuss how you can develop and run your applications on Google Kubernetes Engine.

We start with developing your application code. You can use any source code editor or IDE for this, but to make the development process more efficient, you should use consider using tools that let you validate your code changes in the developer workspace.

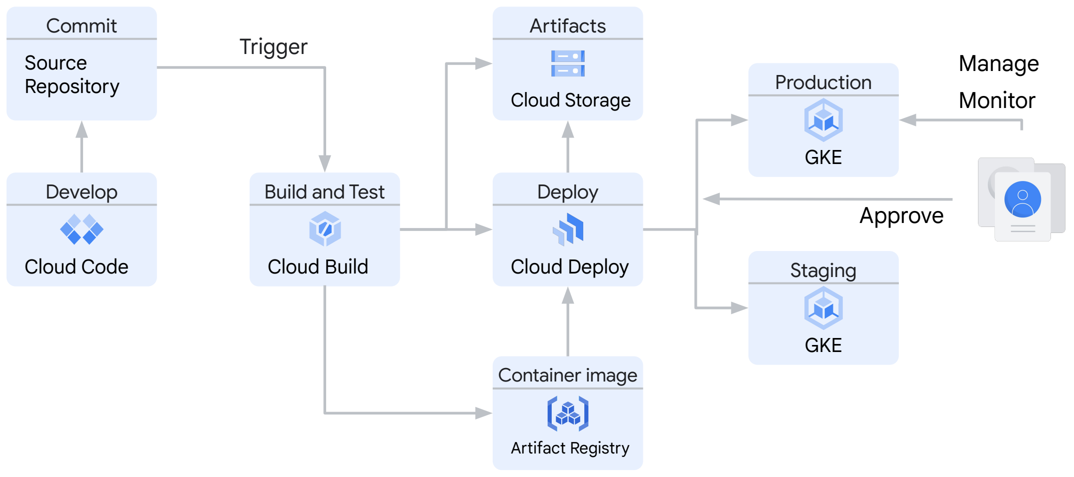
Cloud Code is a set of IDE plugins for popular IDEs that make it easier to create, deploy, and integrate applications with Google Cloud. Cloud Code has built-in capabilities such as Kubernetes and Cloud Run explorers which enable you to visualize, and monitor your cluster resources.

With Code Repository, Cloud Build, and Container or Artifact Registry, and other Google Cloud tools such as Google Cloud Deploy and Skaffold, you can set up a continuous integration and continuous delivery pipeline that deploys your application to Google Kubernetes Engine.

Using the kubectl CLI, you can manage your Kubernetes cluster resources and application workloads.

Proprietary + Confidential

Development and deployment workflow

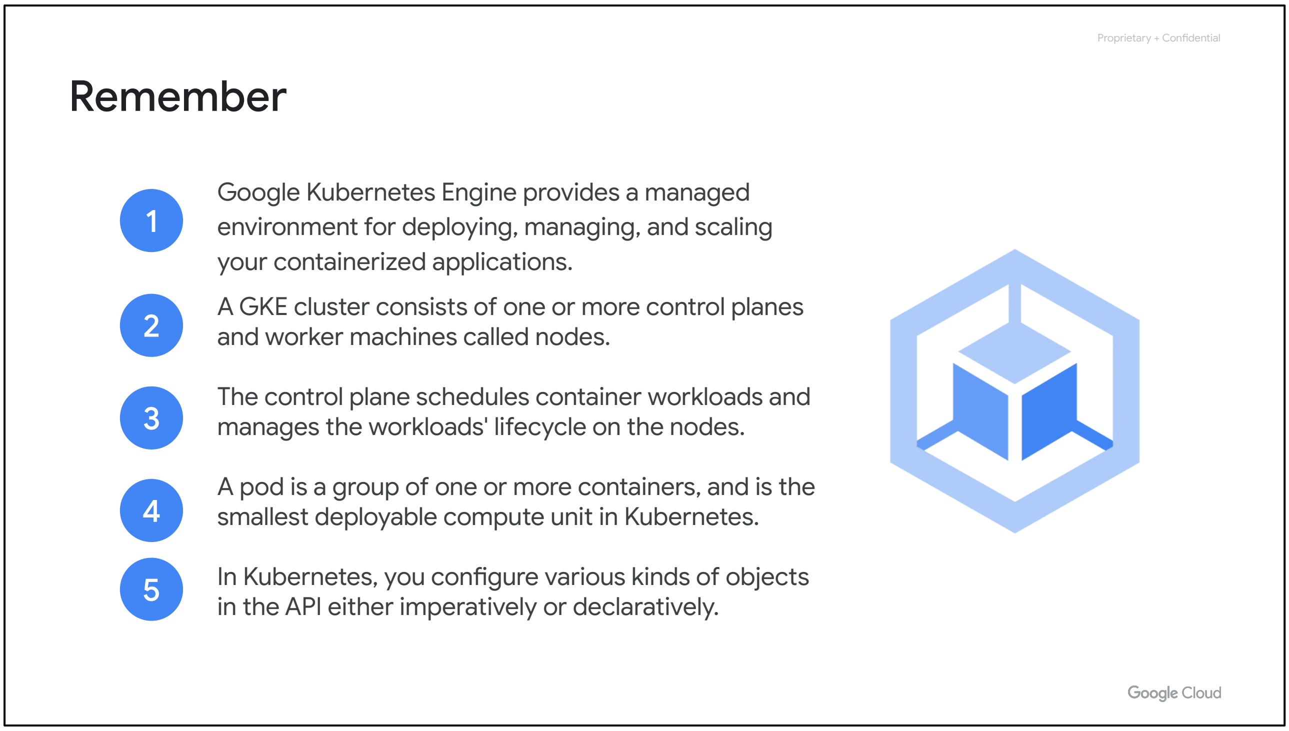


Google Cloud

Using Google Cloud tools, your development and deployment workflow would involve these steps:

1. Use Cloud Code or other editors to create your application source, and store your code in a source code repository such as Cloud Source Repositories, GitHub, or other.
2. Build your application using Cloud Build. Cloud Build lets you set up a CI pipeline that can be triggered from a commit to your source code repository. When a change is committed to the source repository, Cloud Build:
    i. Rebuilds the application container image.
    ii. Stores the image in Artifact Registry.
    iii. Stores any build artifacts in a Cloud Storage bucket.
    iv. Run tests on the container.
    v. Calls Google Cloud Deploy to deploy the container to a staging environment which contains a GKE cluster.
3. If the build and tests are successful, use Cloud Deploy to promote the container to a production environment after approval.
4. Manage your application on GKE.
5. Monitor the performance of your application with Google Cloud's operations suite that includes integrated monitoring and logging for your application.

https://cloud.google.com/architecture/app-development-and-delivery-with-cloud-code-gcb-cd-and-gke



Proprietary + Confidential

Remember

1. Google Kubernetes Engine provides a managed environment for deploying, managing, and scaling your containerized applications.
2. A GKE cluster consists of one or more control planes and worker machines called nodes.
3. The control plane schedules container workloads and manages the workloads' lifecycle on the nodes.
4. A pod is a group of one or more containers, and is the smallest deployable compute unit in Kubernetes.
5. In Kubernetes, you configure various kinds of objects in the API either imperatively or declaratively.

Google Cloud

In summary:

- Google Kubernetes Engine is a fully managed Kubernetes service that provides a managed environment for deploying, managing, and scaling your containerized applications on Google infrastructure.
- A GKE cluster consists of one or more control planes and worker machines called nodes. A zonal cluster has a single control plane running in one zone, while a regional cluster has multiple replicas of the control plane, running in multiple zones within a given region.
- The control plane schedules container workloads and manages the workloads' lifecycle, scaling, upgrades, network, and storage resources.
- A pod is a group of one or more containers, and is the smallest deployable compute unit that you can create and manage in Kubernetes.
- In Kubernetes, you configure various kinds of objects in the API either imperatively or declaratively.