

The first is a pod, and a pod is a mechanism for encapsulating and running a container.

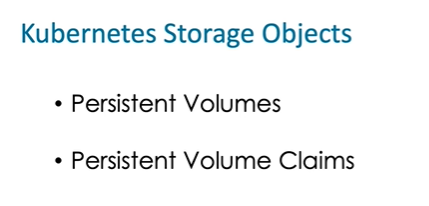
Now, it is possible for a pod to run multiple containers if the two containers are tightly coupled and have similar life cycles, but it's also just as convenient to think of pods as just running a container.

Now, sometimes a pod running a container can go down, for example. So sometimes we wanna have multiple pods running with the same container. We can do that by specifying something called a deployment and a deployment is basically a specification that says run a certain number of replicas of pods and it has some other parameters for defining the deployment. So a deployment is essentially a set of pods that are executing a particular container. When we wanna access a pod, for example, through an API call,

we need to know the IP address of that pod. Now, since pods run in deployments because there are multiple pods out there, any one of those pods in theory could respond to an API request. But because pods can go up and down, they can be recreated, for example, if they're not healthy, we need a way to basically identify

where the particular pods are running or what their IP addresses are. And we do that

by wrapping deployments and using deployments with services. And a service is basically a mechanism for service discovery. And in the case of Kubernetes, it's a way of associating a particular IP address, for example, with a load balancer, we can have a load balancer service and then our API makes a call to the load balancer or the service API and then the service determines what pods are available and what their IP addresses are and routes that traffic for us. So again, just to kind of summarize the process objects, the highest level or the farthest out in terms of our access to a computing object is a service that provides an endpoint with a stable IP address. And then a service wraps a deployment. And a deployment is a set of pods or replicated pods that are actually running the containers. So those are some of the compute abstractions.



Pods and their containers come and go. So that means we need a way to persist data

that we want to be available, for example, when we restart a pod in the case of an unhealthy pod. We want the data that existed before to still be available. The abstraction that makes that available is a persistent volume. So that's essentially some unit of storage that exists outside of a pod and will exist even when a pod shuts down. Now, a pod can make use of the data that's stored on a volume if it has something called a persistent volume claim. And that's basically a mechanism or a data structure that allows a pod to access a volume and work with the data on that volume. So those are the concepts around Kubernetes storage. Persistent volumes are the thing that actually does the storage and a persistent volume claim is sort of an abstraction or a pointer or a reference to that storage so that the pod can access it.

**Deploying cluster in kubernetes engine :**

Creating cluster in gcloud :

gcloud container clusters create cluster-name –zone=() –machine-type=() –disk-size=100GB

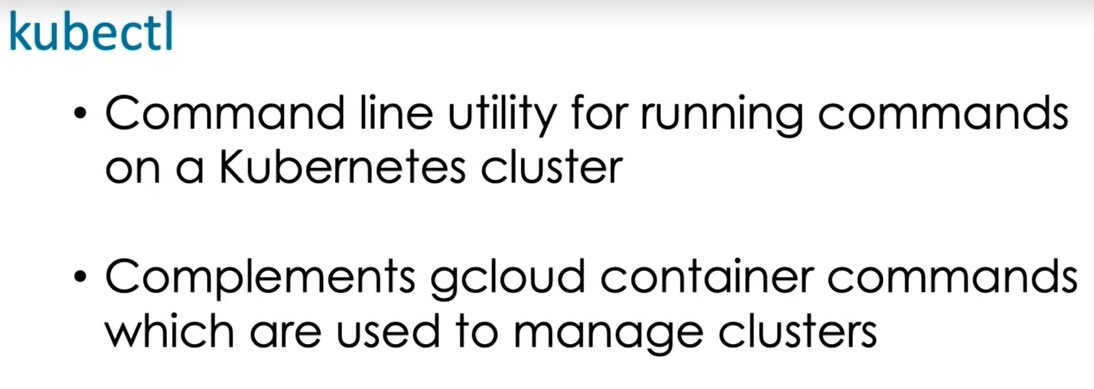
gcloud container clusters list

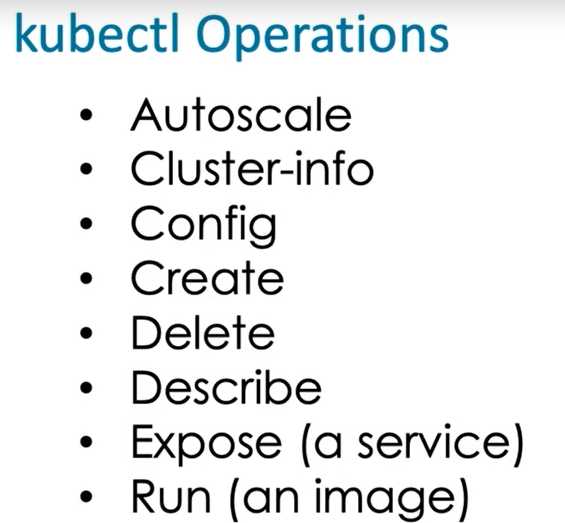
**Deploying application in kubernetes engine :**

Kubernetes engine > Workload > deploy .

**Managing cluster in kubernetes clusters**

Kubectl





Monitoring Kubernetes :

Stack driver monitoring .

Enable - stackdriver kubernetes engine monitoring .

Disable - legacy stackdriver logging

Monitoring > stack driver > kubernetes engine > clusters listed . - that's how we monitor the kubernetes cluster in the stack driver .

From the cluster's view, click on the name of the cluster you're interested in. We have our node pool that lists the status, we see the number of nodes running and the machine type that's being used.

Viewing status of kubernetes cluster :

Now, we can click on the name of the node pool to get even more details on the node pool itself. Now remember, a node pool is implemented as a managed instance group. So that's a group of virtual machine instances that are all the same configuration. By clicking on the name of the node pool, we see this level of detail. So the information includes things like the sides of the nodes, the machine type boot discs, security information, and detailed information about the nodes themselves, such as details about the CPU configuration and memory and storage as well. So, if you're interested in getting metrics, for example, how much CPU is being used on average over the last hour, that's the kind of information you get from stack driver monitoring. If you just wanna understand at a high level, the status of the infrastructure, like, is the status ready and things like that, then you can quickly get that from the Kubernetes engine console just by drilling into the name of the cluster and then getting node pool details.

When containers are tightly coupled and have similar life cycles you might run them in a single pod. It is common practice to run one container per pod. The common use of persistent storage or a database is not a reason to place two or more containers in the same pod.

A service is used to provide a stable endpoint for clients to reach an application. The service is responsible for maintaining information about healthy pods that can accept traffic and process requests.

A Persistent volume stores data independent of specific pods so if a pod fails, another pod can replace it and access the data on the persistent volume by using a persistent volume claim. The other options are not actually Kubernetes components.

Kubernetes Engine is GCP’s managed container orchestration service and meets the requirements.

Node pools are used to segment nodes in a cluster and are implemented using managed instance groups. The other options are not actually components of Kubernetes.

PersistentVolumeClaims is an abstraction used to manage persistent storage in Kubernetes.

gloud containers. Kubernetes Engine used to be called Container Engine. Gsutil is only used for working with Cloud Storage.

GCP uses managed instance groups to implement node pools. Unmanaged instance groups do not have the features needed to implement node pools, such as auto-healing. Neither App Engine Flexible nor Cloud Functions are used to implement node pools.

Kubectl is used to manage the internal state Kubernetes clusters. Gcloud containers are used to work with Kubernetes Engine resources.

gcloud container images list **to view metadata about existing container images**

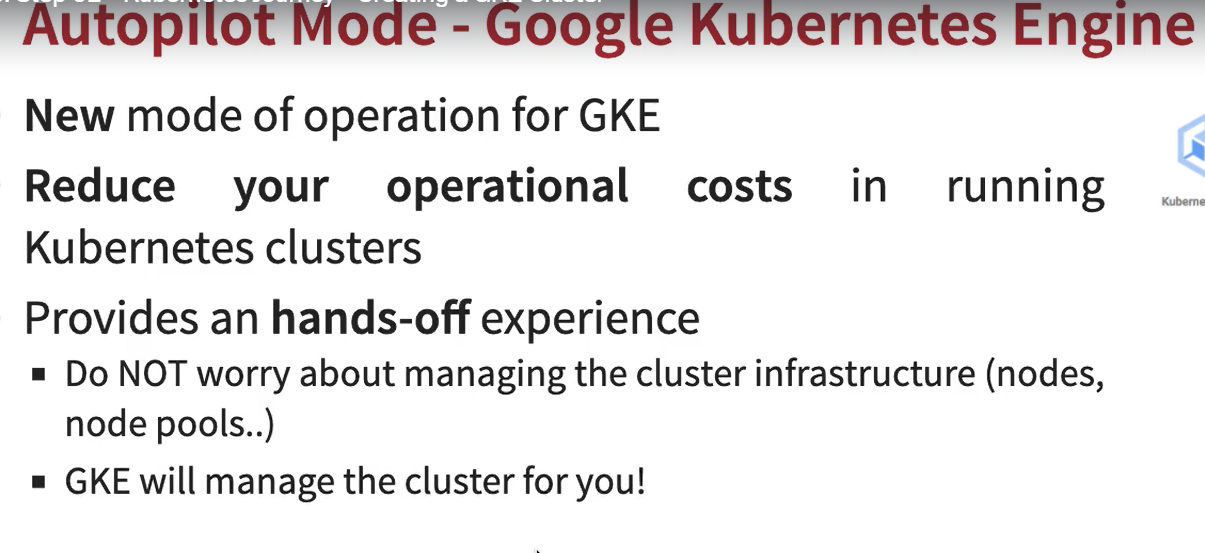
The correct answer is to use the "kubectl apply -f" with the name of the deployment file. Deployments are Kubernetes abstractions and are managed using kubectl, not gcloud.

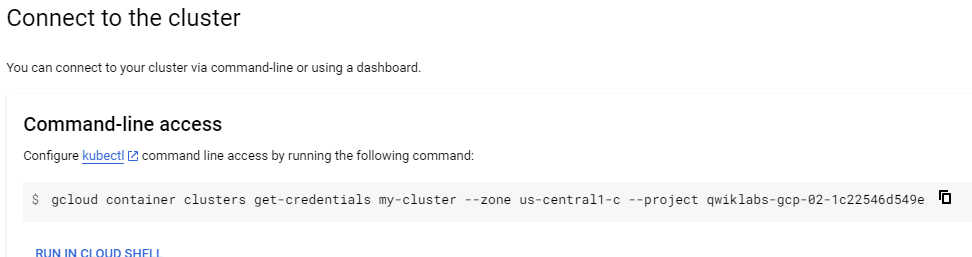
Regional clusters have replicas of the control plane while single zone and multi-zonal clusters have only one control plane. There is no such thing as a multi-regional cluster.

The correct command is to use kubectl autoscale specifying the appropriate min, max, and cpu percent.

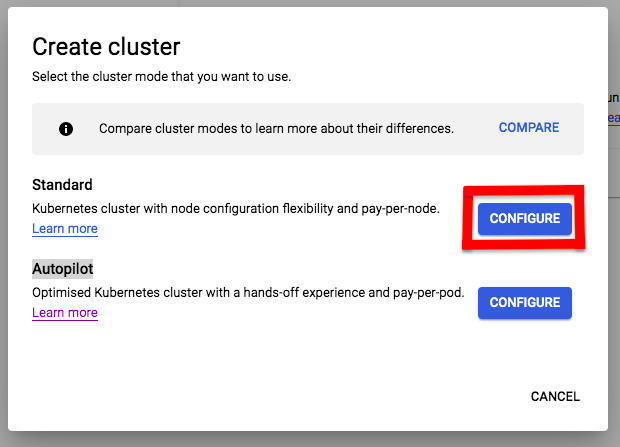
HANDS-ON :

**1:** Recently a new GKE mode was introduced - Autopilot (simplifies cluster management). However, for the demo, we want to use Standard. ***Choose Standard in the next step!***





Once a cluster gets created we need to connect to cluster using above command and kubeconfig entry will be generated for our cluster .

******

**2:** **Here are some of the commands we will run in the next few steps (**Refer back to this if you have any problems!)

* gcloud container clusters get-credentials my-cluster --zone us-central1-c --project my-kubernetes-project-304910
* kubectl create deployment hello-world-rest-api --image=in28min/hello-world-rest-api:0.0.1.RELEASE
* kubectl get deployment
* kubectl expose deployment hello-world-rest-api --type=LoadBalancer --port=8080
* kubectl get services
* kubectl get services --watch
* curl 35.184.204.214:8080/hello-world
* kubectl scale deployment hello-world-rest-api --replicas=3
* gcloud container clusters resize my-cluster --node-pool default-pool --num-nodes=2 --zone=us-central1-c
* kubectl autoscale deployment hello-world-rest-api --max=4 --cpu-percent=70
* kubectl get hpa
* kubectl create configmap hello-world-config --from-literal=RDS\_DB\_NAME=todos
* kubectl get configmap
* kubectl describe configmap hello-world-config
* kubectl create secret generic hello-world-secrets-1 --from-literal=RDS\_PASSWORD=dummytodos
* kubectl get secret
* kubectl describe secret hello-world-secrets-1
* kubectl apply -f deployment.yaml
* gcloud container node-pools list --zone=us-central1-c --cluster=my-cluster
* kubectl get pods -o wide
* kubectl set image deployment hello-world-rest-api hello-world-rest-api=in28min/hello-world-rest-api:0.0.2.RELEASE
* kubectl get services
* kubectl get replicasets
* kubectl get pods
* kubectl delete pod hello-world-rest-api-58dc9d7fcc-8pv7r
* kubectl scale deployment hello-world-rest-api --replicas=1
* kubectl get replicasets
* gcloud projects list
* kubectl delete service hello-world-rest-api
* kubectl delete deployment hello-world-rest-api
* gcloud container clusters delete my-cluster --zone us-central1-c