Creating Kubernetes cluster with GKE .

The best way I found to describe Kubernetes is that it's the best resource manager ever. What does a resource manager manage? It manages resources. So what are the most important resources that Kubernetes manages? It's your servers.

And these servers are in the cloud so they are virtual servers. They are not real servers, but they are actually servers which are in the cloud which are typically called virtual servers.

The fact is that different cloud providers have different names for these virtual servers. Amazon calls them EC2, Elastic Compute Cloud. Azure calls them virtual machines, and Google Cloud calls them Compute Engines. And what does Kubernetes call them? Kubernetes uses a very generic terminology and calls them nodes. Kubernetes can actually manage thousands of such nodes. Now, what do we do when you have thousands of things to manage? You introduce managers. While you might not really like your manager, managers do really have a role to play. To manage thousands of Kubernetes nodes we have a few master nodes. Typically, you'll have one master node, but when you need high availability you go for multiple master nodes. So what is a cluster? A cluster is nothing, but a combination of nodes and the master note. The nodes that do the work are called worker nodes or simply nodes. The nodes that do the management work are called master nodes. These master nodes ensure that the node are really motivated and charged up to do the work just like our manager does.

Master nodes ensure that the nodes are available and are doing some useful work. So at a high level, a cluster contains nodes which are managed by a master node. Now that we understood this, let's get down to creating a cluster.

Enable Kubernetes engine API .

We don't run a number of applications in Kubernetes. Where do we run them? We will run them in Kubernetes clusters.

The workloads are the applications that we would run in Kubernetes. The application that I might deploy to Kubernetes might be a REST API or a web application, which I would want to expose to the outside world. That's where concepts like Services and Ingress will be useful. Each application might have its own configuration. There might also be secret values or passwords that need to be stored. That's where you can make use of secrets and config maps. Certain applications might need shared persistent storage. That's where you can go for storage. As you can see, there are a lot of features which are present in here. We'll discuss them as we go further in the course. For now, let's focus on creating a cluster. So I'll go back to clusters and I would say, I would want to create a new cluster. There are two modes that you can create a cluster in. One is called the standard mode and the other one is called the autopilot mode. In the standard mode, we configure and manage all our nodes. However, in the autopilot mode you need to do very little configuration. GKE would do almost everything for you. For us to get started, I would choose the standard mode. We would want to choose how many nodes we want, what is the configuration for them and all that kind of stuff. So I would go ahead and choose GKE Standard and say Configure. This would bring us to the create cluster page where I can name my cluster.

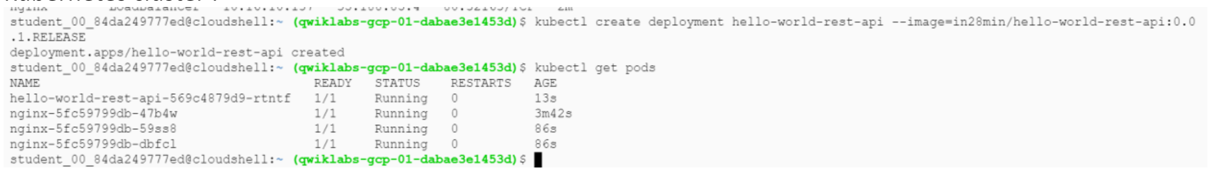
We can define zone and region for our cluster .

DEPLOY FIRST CONTAINER TO KUBERNETES :

- Connect to Kubernetes cluster : gcloud container clusters get-credentials cluster_name --zone Zone_of_cluster --project project_name
- to run commands against this cluster. How do we do that? That's where something called kubectl comes into picture. Kubectl is K U B E C T L. Kubectl is short form for kube controller.
- Kubectl interacts with the cluster
- Kubectl works with all Kubernetes cluster hosted in on prem , cloud .

Lets deploy helloworld rest api using kubectl : kubectl create deployment hello-world-rest-api --image=in28min/hello-world-rest-api:0.0.1.RELEASE

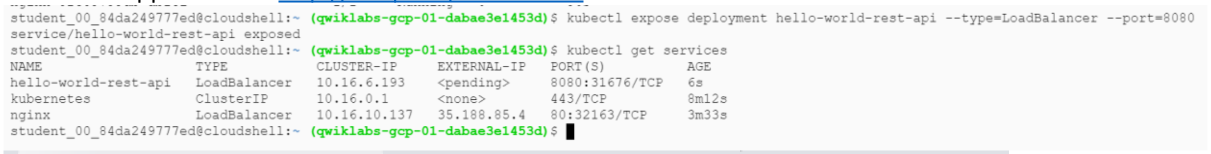
Image which we are using is pushed to dockerhub . The above command will deploy an application to Kubernetes cluster .



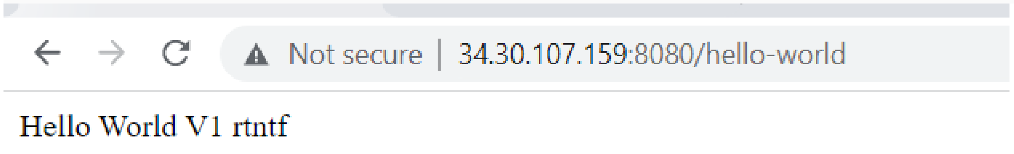
```
student_00_84da249777ed@cloudshell:~ (qwiklabs-gcp-01-dabae3e1453d)$ kubectl create deployment hello-world-rest-api --image=in28min/hello-world-rest-api:0.0
.1.RELEASE
deployment.apps/hello-world-rest-api created
student_00_84da249777ed@cloudshell:~ (qwiklabs-gcp-01-dabae3e1453d)$ kubectl get pods
NAME                                    READY   STATUS    RESTARTS   AGE
hello-world-rest-api-569c4879d9-rtntf   1/1     Running   0          13s
nginx-5fc59799db-47b4w                  1/1     Running   0          3m42s
nginx-5fc59799db-59ss8                  1/1     Running   0          86s
nginx-5fc59799db-dbfcl                  1/1     Running   0          86s
student_00_84da249777ed@cloudshell:~ (qwiklabs-gcp-01-dabae3e1453d)$
```

To expose deployment to outside world using load balancer : kubectl expose deployment hello-world-rest-api --type=LoadBalancer --port=8080

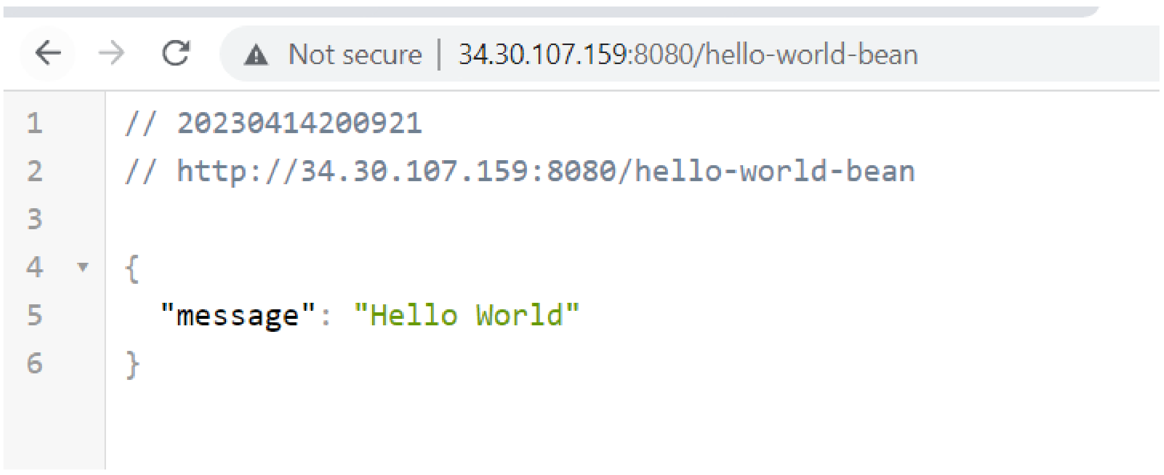
It will create a load balancer in GCP .
Access the application as http://endpoint/hello-world or hello-world-bean



```
student_00_84da249777ed@cloudshell:~ (qwiklabs-gcp-01-dabae3e1453d)$ kubectl expose deployment hello-world-rest-api --type=LoadBalancer --port=8080
service/hello-world-rest-api exposed
student_00_84da249777ed@cloudshell:~ (qwiklabs-gcp-01-dabae3e1453d)$ kubectl get services
NAME                   TYPE           CLUSTER-IP     EXTERNAL-IP   PORT(S)          AGE
hello-world-rest-api   LoadBalancer   10.16.6.193    <pending>     8080:31676/TCP   6s
kubernetes             ClusterIP      10.16.0.1      <none>        443/TCP          8m12s
nginx                  LoadBalancer   10.16.10.137   35.188.85.4   80:32163/TCP     3m33s
student_00_84da249777ed@cloudshell:~ (qwiklabs-gcp-01-dabae3e1453d)$
```



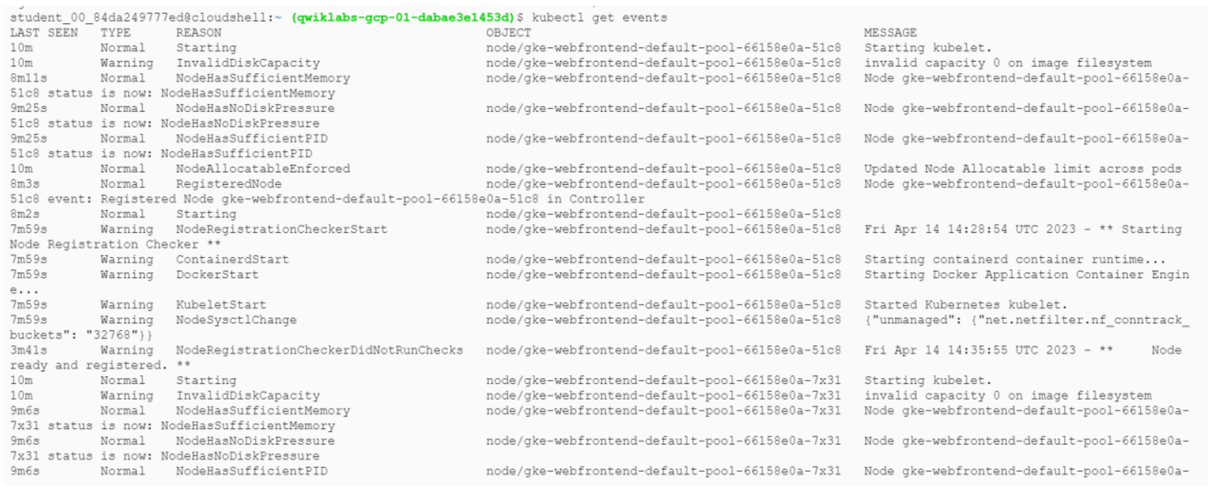
Hello World V1 rtntf



```
// 20230414200921
// http://34.30.107.159:8080/hello-world-bean

{
  "message": "Hello World"
}
```

To get list of all events in Kubernetes cluster : kubectl get events

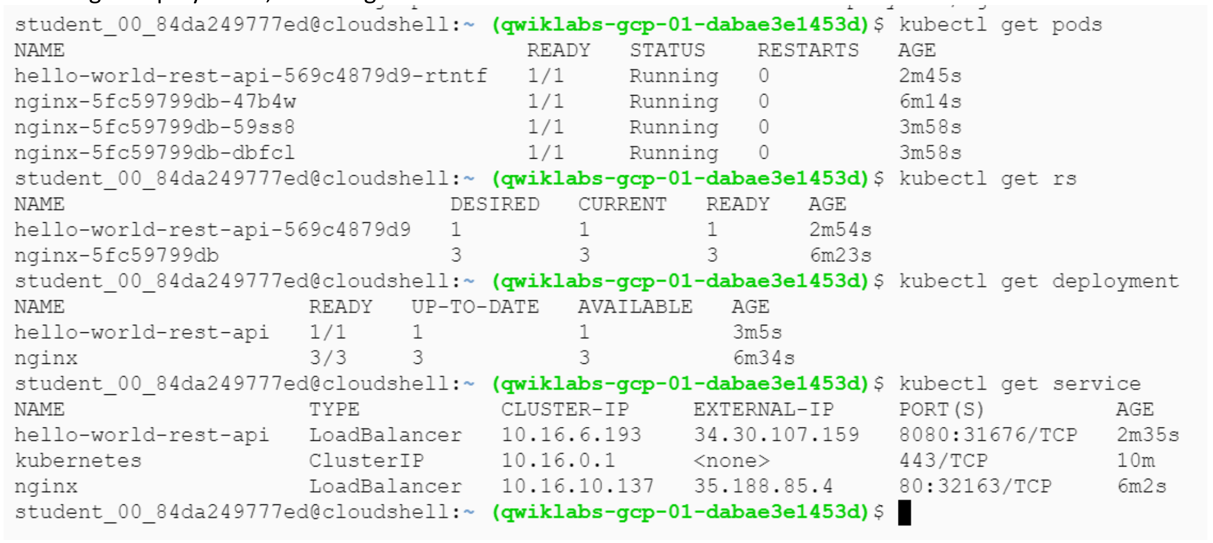


```
student_00_84da249777ed@cloudshell:~ (qwiklabs-gcp-01-dabae3e1453d)$ kubectl get events
LAST SEEN   TYPE      REASON                                  OBJECT                                              MESSAGE
10m         Normal    Starting                                node/gke-webfrontend-default-pool-66158e0a-51c8    Starting kubelet.
10m         Warning   InvalidDiskCapacity                     node/gke-webfrontend-default-pool-66158e0a-51c8    invalid capacity 0 on image filesystem
8m11s       Normal    NodeHasSufficientMemory                 node/gke-webfrontend-default-pool-66158e0a-51c8    Node gke-webfrontend-default-pool-66158e0a-
51c8 status is now: NodeHasSufficientMemory
9m25s       Normal    NodeHasNoDiskPressure                   node/gke-webfrontend-default-pool-66158e0a-51c8    Node gke-webfrontend-default-pool-66158e0a-
51c8 status is now: NodeHasNoDiskPressure
9m25s       Normal    NodeHasSufficientPID                    node/gke-webfrontend-default-pool-66158e0a-51c8    Node gke-webfrontend-default-pool-66158e0a-
51c8 status is now: NodeHasSufficientPID
10m         Normal    NodeAllocatableEnforced                 node/gke-webfrontend-default-pool-66158e0a-51c8    Updated Node Allocatable limit across pods
8m3s        Normal    RegisteredNode                          node/gke-webfrontend-default-pool-66158e0a-51c8    Node gke-webfrontend-default-pool-66158e0a-
51c8 event: Registered Node gke-webfrontend-default-pool-66158e0a-51c8 in Controller
8m2s        Normal    Starting                                node/gke-webfrontend-default-pool-66158e0a-51c8
7m59s       Warning   NodeRegistrationCheckerStart            node/gke-webfrontend-default-pool-66158e0a-51c8    Fri Apr 14 14:28:54 UTC 2023 - ** Starting
Node Registration Checker **
7m59s       Warning   ContainerdStart                         node/gke-webfrontend-default-pool-66158e0a-51c8    Starting containerd container runtime...
7m59s       Warning   DockerStart                             node/gke-webfrontend-default-pool-66158e0a-51c8    Starting Docker Application Container Engin
e...
7m59s       Warning   KubeletStart                            node/gke-webfrontend-default-pool-66158e0a-51c8    Started Kubernetes kubelet.
7m59s       Warning   NodeSysctlChange                        node/gke-webfrontend-default-pool-66158e0a-51c8    {"unmanaged": {"net.netfilter.nf_conntrack_
buckets": "32768"}}
3m41s       Warning   NodeRegistrationCheckerDidNotRunChecks  node/gke-webfrontend-default-pool-66158e0a-51c8    Fri Apr 14 14:35:55 UTC 2023 - **     Node
ready and registered. **
10m         Normal    Starting                                node/gke-webfrontend-default-pool-66158e0a-7x31    Starting kubelet.
10m         Warning   InvalidDiskCapacity                     node/gke-webfrontend-default-pool-66158e0a-7x31    invalid capacity 0 on image filesystem
9m6s        Normal    NodeHasSufficientMemory                 node/gke-webfrontend-default-pool-66158e0a-7x31    Node gke-webfrontend-default-pool-66158e0a-
7x31 status is now: NodeHasSufficientMemory
9m6s        Normal    NodeHasNoDiskPressure                   node/gke-webfrontend-default-pool-66158e0a-7x31    Node gke-webfrontend-default-pool-66158e0a-
7x31 status is now: NodeHasNoDiskPressure
9m6s        Normal    NodeHasSufficientPID                    node/gke-webfrontend-default-pool-66158e0a-7x31    Node gke-webfrontend-default-pool-66158e0a-
```

Get list of pods in Kubernetes cluster : kubectl get pods

To get all replicaset : kubectl get replicaset

kubectl get deployment , kubectl get service

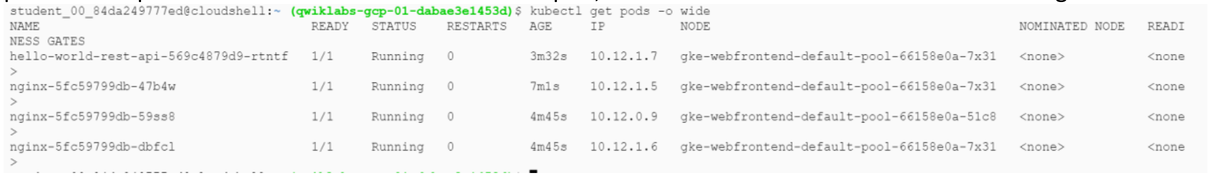


```
student_00_84da249777ed@cloudshell:~ (qwiklabs-gcp-01-dabae3e1453d)$ kubectl get pods
NAME                                    READY   STATUS    RESTARTS   AGE
hello-world-rest-api-569c4879d9-rtntf   1/1     Running   0          2m45s
nginx-5fc59799db-47b4w                  1/1     Running   0          6m14s
nginx-5fc59799db-59ss8                  1/1     Running   0          3m58s
nginx-5fc59799db-dbfcl                  1/1     Running   0          3m58s
student_00_84da249777ed@cloudshell:~ (qwiklabs-gcp-01-dabae3e1453d)$ kubectl get rs
NAME                              DESIRED   CURRENT   READY   AGE
hello-world-rest-api-569c4879d9   1         1         1       2m54s
nginx-5fc59799db                  3         3         3       6m23s
student_00_84da249777ed@cloudshell:~ (qwiklabs-gcp-01-dabae3e1453d)$ kubectl get deployment
NAME                   READY   UP-TO-DATE   AVAILABLE   AGE
hello-world-rest-api   1/1     1            1           3m5s
nginx                  3/3     3            3           6m34s
student_00_84da249777ed@cloudshell:~ (qwiklabs-gcp-01-dabae3e1453d)$ kubectl get service
NAME                   TYPE           CLUSTER-IP     EXTERNAL-IP     PORT(S)          AGE
hello-world-rest-api   LoadBalancer   10.16.6.193    34.30.107.159   8080:31676/TCP   2m35s
kubernetes             ClusterIP      10.16.0.1      <none>          443/TCP          10m
nginx                  LoadBalancer   10.16.10.137   35.188.85.4     80:32163/TCP     6m2s
student_00_84da249777ed@cloudshell:~ (qwiklabs-gcp-01-dabae3e1453d)$
```

When we executed the command kubectl create deployment, Kubernetes created a deployment, a replica set, and a port. When we executed the command kubectl exposed deployment Kubernetes created a service for us.

Pods in kuberneets :
Pod is smallest deployable unit in Kubernetes .
You cannot have a container in Kubernetes without a pod. Your container lives inside a pod.

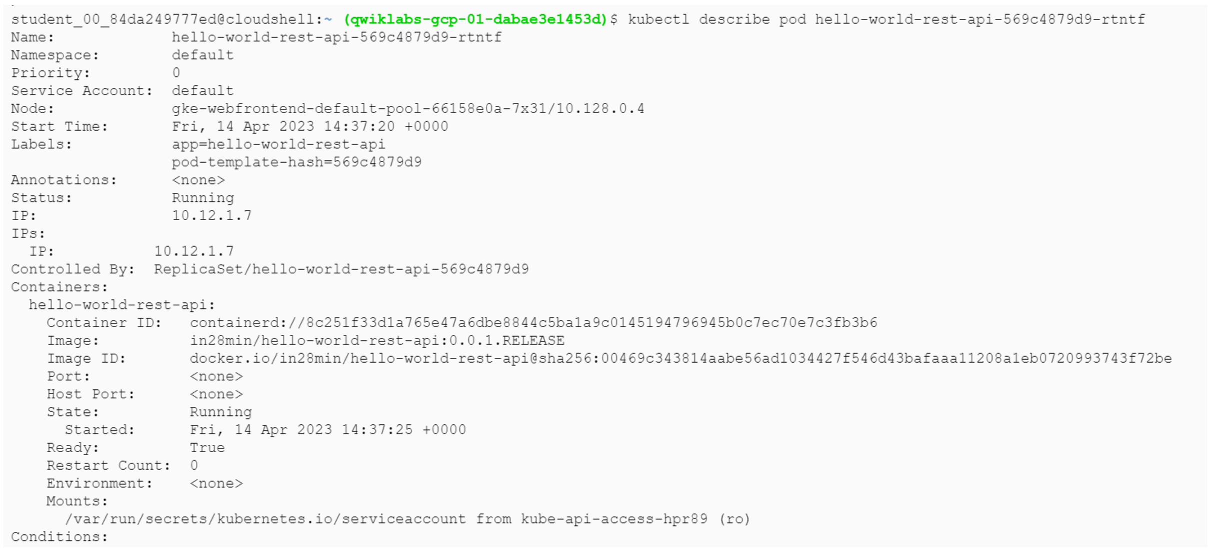
Get detailed information of pod : kubectl get pods -o wide
Each pod has unique ip address . A pod can actually contain multiple containers. All the containers which are present in a pod share resources. Within the same pod, the containers can talk to each other using local host.



```
student_00_84da249777ed@cloudshell:~ (qwiklabs-gcp-01-dabae3e1453d)$ kubectl get pods -o wide
NAME                                    READY   STATUS    RESTARTS   AGE     IP           NODE                                          NOMINATED NODE   READI
NESS GATES
hello-world-rest-api-569c4879d9-rtntf   1/1     Running   0          3m32s   10.12.1.7    gke-webfrontend-default-pool-66158e0a-7x31   <none>           <none
>
nginx-5fc59799db-47b4w                  1/1     Running   0          7m1s    10.12.1.5    gke-webfrontend-default-pool-66158e0a-7x31   <none>           <none
>
nginx-5fc59799db-59ss8                  1/1     Running   0          4m45s   10.12.0.9    gke-webfrontend-default-pool-66158e0a-51c8   <none>           <none
>
nginx-5fc59799db-dbfcl                  1/1     Running   0          4m45s   10.12.1.6    gke-webfrontend-default-pool-66158e0a-7x31   <none>           <none
>
```

To get explanation about pod component : kubectl explain pods Kubernetes node can contain multiple pods, and each of these pods can contain multiple containers.

kubectl describe pod pod_name : to get description about pod like namespace , node on which pod running etc.



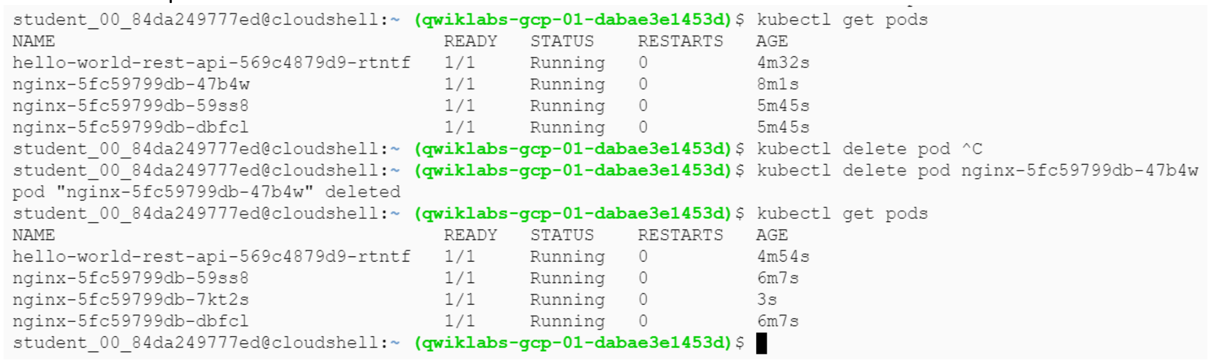
```
student_00_84da249777ed@cloudshell:~ (qwiklabs-gcp-01-dabae3e1453d)$ kubectl describe pod hello-world-rest-api-569c4879d9-rtntf
Name:             hello-world-rest-api-569c4879d9-rtntf
Namespace:        default
Priority:         0
Service Account:  default
Node:             gke-webfrontend-default-pool-66158e0a-7x31/10.128.0.4
Start Time:       Fri, 14 Apr 2023 14:37:20 +0000
Labels:           app=hello-world-rest-api
                  pod-template-hash=569c4879d9
Annotations:      <none>
Status:           Running
IP:               10.12.1.7
IPs:
  IP:           10.12.1.7
Controlled By:  ReplicaSet/hello-world-rest-api-569c4879d9
Containers:
  hello-world-rest-api:
    Container ID:   containerd://8c251f33d1a765e47a6dbe8844c5ba1a9c0145194796945b0c7ec70e7c3fb3b6
    Image:          in28min/hello-world-rest-api:0.0.1.RELEASE
    Image ID:       docker.io/in28min/hello-world-rest-api@sha256:00469c343814aabe56ad1034427f546d43bafaaa11208a1eb0720993743f72be
    Port:           <none>
    Host Port:      <none>
    State:          Running
      Started:      Fri, 14 Apr 2023 14:37:25 +0000
    Ready:          True
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-hpr89 (ro)
Conditions:
```

Namespace is a very important concept. It provides isolations for parts of the cluster from other parts of the cluster. What do I mean? Let's say, you have your development and QA environments running inside the same cluster. How do you separate the resources, of Dev from the resources of QA? One of the options, is to create separate namespaces for QA and Dev, and associate each of the resources with that specific namespace.
For now, let's keep things simple and stick to the default namespace. Now, what else does a pod have?
You can see that the pod has labels. We are attaching a label called app, is equal to Hello world REST API with this specific pod. Earlier, we talked about a number of concepts, a pod, a replica set, a deployment, a service. In Kubernetes, the way you link all these together is by using something called selectors and labels.
So labels are really, really important. When we get to tying up pod with a replica set or a service .

Annotations are typically meta information about that specific pod. What was the release ID? What was the build ID? What was the author name? And all that kind of stuff, are what are put into annotations.
And the last thing that you see in here is something called a status. This pod is currently in a running status.
The important thing that you need to remember about pod, is a pod provides a way to put your containers together. It gives them an IP address, and also, it provides a categorization for all these containers by associating them with labels. At high level, this is how you can picturize how pods would be running on nodes. On any Kubernetes nodes there can be multiple pods and each of these pods can contain multiple containers that are running as part of them.

Replicasets in Kubernetes :
Get details of replicaset : kubectl get replicaset / kubectl get rs

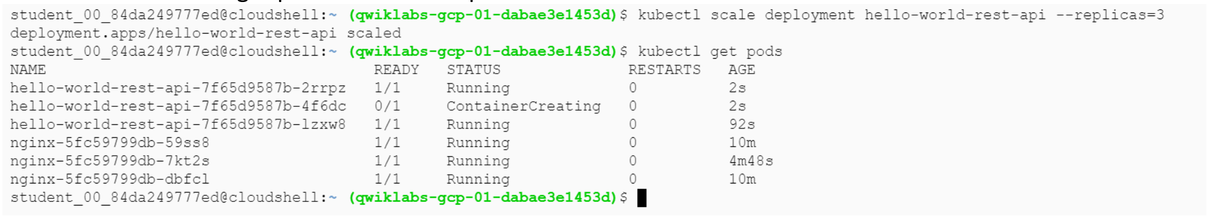
Replica set ensure specific number of pods are running at all times .
Delete pod : kubectl delete pods pod_name
If we delete the pod then replicaset will create new pod again .
The replica set always keeps monitoring the pods, and if there are lesser number of pods than what is needed, then it creates the pods. We said we would want one pod running all time and when it goes down to zero immediately the replica set looks at it and says, "Okay, there is one pod missing, I'll need to start it up" and it kicks off a new pod.



```
student_00_84da249777ed@cloudshell:~ (qwiklabs-gcp-01-dabae3e1453d)$ kubectl get pods
NAME                                    READY   STATUS    RESTARTS   AGE
hello-world-rest-api-569c4879d9-rtntf   1/1     Running   0          4m32s
nginx-5fc59799db-47b4w                  1/1     Running   0          8m1s
nginx-5fc59799db-59ss8                  1/1     Running   0          5m45s
nginx-5fc59799db-dbfcl                  1/1     Running   0          5m45s
student_00_84da249777ed@cloudshell:~ (qwiklabs-gcp-01-dabae3e1453d)$ kubectl delete pod ^C
student_00_84da249777ed@cloudshell:~ (qwiklabs-gcp-01-dabae3e1453d)$ kubectl delete pod nginx-5fc59799db-47b4w
pod "nginx-5fc59799db-47b4w" deleted
student_00_84da249777ed@cloudshell:~ (qwiklabs-gcp-01-dabae3e1453d)$ kubectl get pods
NAME                                    READY   STATUS    RESTARTS   AGE
hello-world-rest-api-569c4879d9-rtntf   1/1     Running   0          4m54s
nginx-5fc59799db-59ss8                  1/1     Running   0          6m7s
nginx-5fc59799db-7kt2s                  1/1     Running   0          3s
nginx-5fc59799db-dbfcl                  1/1     Running   0          6m7s
student_00_84da249777ed@cloudshell:~ (qwiklabs-gcp-01-dabae3e1453d)$
```

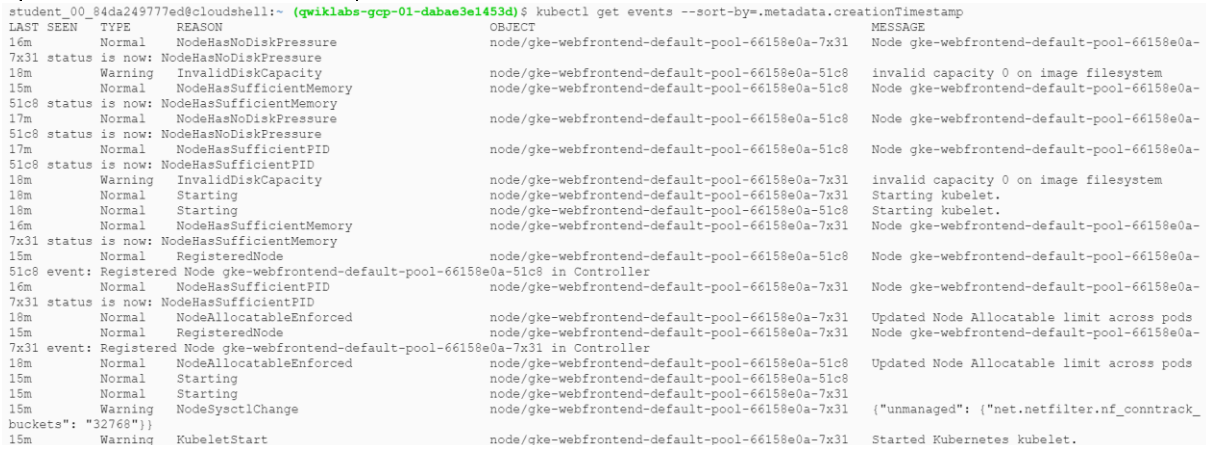
In the replica set I would want to have three pods or two pods. How can I do that, how can I tell the replica set to maintain a higher number of pods? kubectl scale deployment deployment_name --replicas=3

Now if we do kubectl get pods : number of pods will be increased .



```
student_00_84da249777ed@cloudshell:~ (qwiklabs-gcp-01-dabae3e1453d)$ kubectl scale deployment hello-world-rest-api --replicas=3
deployment.apps/hello-world-rest-api scaled
student_00_84da249777ed@cloudshell:~ (qwiklabs-gcp-01-dabae3e1453d)$ kubectl get pods
NAME                                    READY   STATUS              RESTARTS   AGE
hello-world-rest-api-7f65d9587b-2rrpz   1/1     Running             0          2s
hello-world-rest-api-7f65d9587b-4f6dc   0/1     ContainerCreating   0          2s
hello-world-rest-api-7f65d9587b-lzxw8   1/1     Running             0          92s
nginx-5fc59799db-59ss8                  1/1     Running             0          10m
nginx-5fc59799db-7kt2s                  1/1     Running             0          4m48s
nginx-5fc59799db-dbfcl                  1/1     Running             0          10m
student_00_84da249777ed@cloudshell:~ (qwiklabs-gcp-01-dabae3e1453d)$
```

In kubectl get rs : desired number of pods will be increased to 3 .
To get details of all events in Kubernetes in sorted order of time : kubectl get events --sort-by=.metadata.creationTimestamp

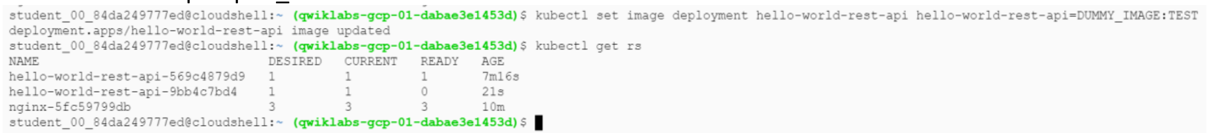


```
student_00_84da249777ed@cloudshell:~ (qwiklabs-gcp-01-dabae3e1453d)$ kubectl get events --sort-by=.metadata.creationTimestamp
LAST SEEN   TYPE      REASON                     OBJECT                                          MESSAGE
16m         Normal    NodeHasNoDiskPressure      node/gke-webfrontend-default-pool-66158e0a-7x31   Node gke-webfrontend-default-pool-66158e0a-
7x31 status is now: NodeHasNoDiskPressure
18m         Warning   InvalidDiskCapacity        node/gke-webfrontend-default-pool-66158e0a-51c8   invalid capacity 0 on image filesystem
15m         Normal    NodeHasSufficientMemory    node/gke-webfrontend-default-pool-66158e0a-51c8   Node gke-webfrontend-default-pool-66158e0a-
51c8 status is now: NodeHasSufficientMemory
17m         Normal    NodeHasNoDiskPressure      node/gke-webfrontend-default-pool-66158e0a-51c8   Node gke-webfrontend-default-pool-66158e0a-
51c8 status is now: NodeHasNoDiskPressure
17m         Normal    NodeHasSufficientPID       node/gke-webfrontend-default-pool-66158e0a-51c8   Node gke-webfrontend-default-pool-66158e0a-
51c8 status is now: NodeHasSufficientPID
18m         Warning   InvalidDiskCapacity        node/gke-webfrontend-default-pool-66158e0a-7x31   invalid capacity 0 on image filesystem
18m         Normal    Starting                   node/gke-webfrontend-default-pool-66158e0a-7x31   Starting kubelet.
18m         Normal    Starting                   node/gke-webfrontend-default-pool-66158e0a-51c8   Starting kubelet.
16m         Normal    NodeHasSufficientMemory    node/gke-webfrontend-default-pool-66158e0a-7x31   Node gke-webfrontend-default-pool-66158e0a-
7x31 status is now: NodeHasSufficientMemory
15m         Normal    RegisteredNode             node/gke-webfrontend-default-pool-66158e0a-51c8   Node gke-webfrontend-default-pool-66158e0a-
51c8 event: Registered Node gke-webfrontend-default-pool-66158e0a-51c8 in Controller
16m         Normal    NodeHasSufficientPID       node/gke-webfrontend-default-pool-66158e0a-7x31   Node gke-webfrontend-default-pool-66158e0a-
7x31 status is now: NodeHasSufficientPID
18m         Normal    NodeAllocatableEnforced    node/gke-webfrontend-default-pool-66158e0a-7x31   Updated Node Allocatable limit across pods
15m         Normal    RegisteredNode             node/gke-webfrontend-default-pool-66158e0a-7x31   Node gke-webfrontend-default-pool-66158e0a-
7x31 event: Registered Node gke-webfrontend-default-pool-66158e0a-7x31 in Controller
18m         Normal    NodeAllocatableEnforced    node/gke-webfrontend-default-pool-66158e0a-51c8   Updated Node Allocatable limit across pods
15m         Normal    Starting                   node/gke-webfrontend-default-pool-66158e0a-51c8
15m         Normal    Starting                   node/gke-webfrontend-default-pool-66158e0a-7x31
15m         Warning   NodeSysctlChange           node/gke-webfrontend-default-pool-66158e0a-7x31   {"unmanaged": {"net.netfilter.nf_conntrack_
buckets": "32768"}}
15m         Warning   KubeletStart               node/gke-webfrontend-default-pool-66158e0a-7x31   Started Kubernetes kubelet.
```

Deployments in Kubernetes :
Lets say we have version v1 and we want to update to v2 . We would want zore downtime deployment .
Kubectl get rs : check replicastet and deployment deployed .

Lets deploy new version of image : kubectl set image deployment deployment_name container_name=DUMMY_IMAGE:TEST

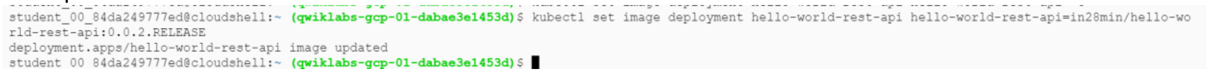
So we are updating the image of container in the deployment and test whether our deployment will go down or not . When we update the image new replicaset will be created with new image .
When we check with kubectl get pods we will get status of new pods with new image .
kubectl describe pod pod_name



```
student_00_84da249777ed@cloudshell:~ (qwiklabs-gcp-01-dabae3e1453d)$ kubectl set image deployment hello-world-rest-api hello-world-rest-api=DUMMY_IMAGE:TEST
deployment.apps/hello-world-rest-api image updated
student_00_84da249777ed@cloudshell:~ (qwiklabs-gcp-01-dabae3e1453d)$ kubectl get rs
NAME                              DESIRED   CURRENT   READY   AGE
hello-world-rest-api-569c4879d9   1         1         1       7m16s
hello-world-rest-api-9bb4c7bd4    1         1         0       21s
nginx-5fc59799db                  3         3         3       10m
student_00_84da249777ed@cloudshell:~ (qwiklabs-gcp-01-dabae3e1453d)$
```

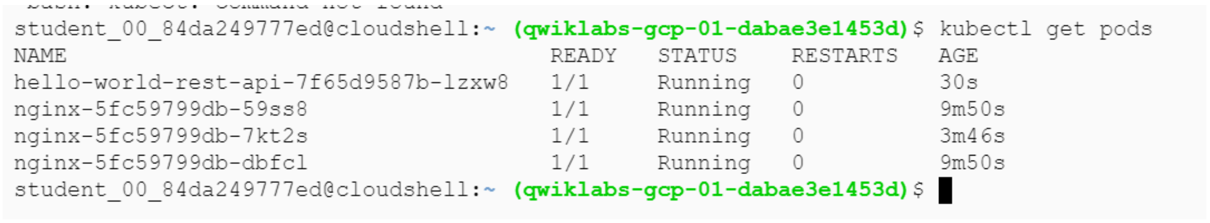
since image we specified is incorrect it will give error in pod .

Lets update image: kubectl set image deployment deployment_name container_name=in28min/hello-world-rest-api:0.0.2>RELEASE



```
student_00_84da249777ed@cloudshell:~ (qwiklabs-gcp-01-dabae3e1453d)$ kubectl set image deployment hello-world-rest-api hello-world-rest-api=in28min/hello-wo
rld-rest-api:0.0.2.RELEASE
deployment.apps/hello-world-rest-api image updated
student 00 84da249777ed@cloudshell:~ (qwiklabs-gcp-01-dabae3e1453d)$
```

Now if we check kubectl get pods – 3 pods will be running with new image and older pods will be terminated with older version .



```
student_00_84da249777ed@cloudshell:~ (qwiklabs-gcp-01-dabae3e1453d)$ kubectl get pods
NAME                                    READY   STATUS    RESTARTS   AGE
hello-world-rest-api-7f65d9587b-lzxw8   1/1     Running   0          30s
nginx-5fc59799db-59ss8                  1/1     Running   0          9m50s
nginx-5fc59799db-7kt2s                  1/1     Running   0          3m46s
nginx-5fc59799db-dbfcl                  1/1     Running   0          9m50s
student_00_84da249777ed@cloudshell:~ (qwiklabs-gcp-01-dabae3e1453d)$
```

Now when we check replicaset : kubectl get rs – older version of rs will be with 0 pods and new image version rs will be running with 3 pods . so as the new replicaset creates new pods the older pods go on terminating one by one .
So in summary, a deployment is very very important to make sure that you are able to update new releases of applications without downtime. The strategy which the deployment is using by default is something called rolling updates. What it does is, let's say I have five insensitive of V1 and I want to update to V2, the rolling update strategy It updates one port at a time. So it launches up a new pod for V2. Once it's up and running it reduces the number of pods for V1. Next, it increases the number of pods for V2 and so on and so forth until the number of ports for V1 becomes zero. And the all the traffic goes then to V2. There are other deployment strategies

Pod – wrapper for set of container . pod has ip , labels , annotations . Replicaset ensures specific number of pod are always running . Replica set is always tied to specific version . Deployment ensure that release upgrade from v1 to v2 happens smoothly .

Services in Kubernetes :

kubectl get pods -o wide : we can see that each pod has its own ip address . And the service that we create to expose our application will load balance traffic to all pods behind it . Ip of pods keep on changing and we want our user to access 1 single ip for application . The role of a service is to provide an always available external interface to the applications which are running inside the ports. A service basically allows your application to receive traffic through a permanent lifetime IP address. Now, when was this service created? This service was created when we did an exposed deployment.

we executed kubectl expose deployment. type is equal to load balancer This is when the service was created with an IP address and that's the IP address that we are using to access the application. Now let's go to the UI and let's go to the service and ingress. This is the service that we have created. Hello World rest API status is okay, type is load balancer and it's load balancing between whatever pods are active at that particular point in time. You'd see the end points in here. If you click this and go in and further scroll down you can see all the labels associated with it. You can see the deployments which are associated with it. and the ports which are running right now. You can see that these are all the ports which are serving the requests for that specific service. Now, how is this specific service implemented in Google Cloud there is a Google Cloud load balancer, which is created. So if you search for load balancing and go to the network services load balancing, you'd see that there is a specific load balancer which was created for this service. So let's click that and you can see what are the things that are in the front end.

So that's the front end and in the back end we have a number of things. So this is the load balancer, which is getting updated. As we create new ports, which delete ports everything gets updated as the backend. However, the front end remains the same, the same URL same port will be able to access the application and respect you of how many ports are running in the background. In summary, a service allows your application to receive traffic through a permanent lifetime address. A service remains up and running. It provides a constant front end interface irrespective you of whatever changes are happening to the back end which are all the pods where your applications are running. The great thing about Kubernetes is the fact that it provides excellent integration with different cloud provider specific load balancers. We were using a specific type of service called load balancer, and we saw that a Google Cloud platform load balancer was created for us. That shows how well Kubernetes integrated with Google Cloud platform to create a load balancer. If you were working with AWS or if you were working with Azure, then that cloud provider specific load balancers would have been created for you as a service. Let's go to our cloud shell do a clear and run
kubectl get services
This would list all the services that are running right now. You see that we have a service load balancer type of service.
A load balancer can load balance between multiple pods. Now the other type of service is a cluster IP service or here you can see that the Kubernetes service is actually running as a cluster IP service.
Now what does the cluster IP service mean? A cluster IP service can only be accessed from inside the cluster. You'll not be able to access this service from outside the cluster.

You can see that there is no external IP for this specific service. So if you have any services which are directly consumed inside your cluster, you can have them use a cluster IP.

Kubernetes architecture :
What are the important components that are running as part of your master node? The most important component that is running as part of your master node is something called etcd. That's the distributed database, etcd. All the configuration changes that we are making, all the deployments that we are creating, all the scaling operations that we are performing, all the details of those are stored in a distributed database. What we are doing when we are executing those commands is setting the desired state for Kubernetes. We are telling Kubernetes, "I would want five instances of application A, I would want 10 instances of application B." That's what is called desired state. And the desired state is stored in the distributed database, etcd. so all the Kubernetes resources like deployment, services, all the configuration that we make is stored finally into this distributed database. The great thing about this database is that it is distributed. Typically, we would recommend you to have about three to five replicas of this database so that the Kubernetes cluster state is not lost.

The second important component inside the master node is something called the API server, kube-apiserver. Earlier, we were executing commands from kubectl. We were making changes from the Google Cloud console from the interface provided by Google Cloud. How does kubectl talk to Kubernetes cluster? How does the Google Cloud interface or the Google Cloud console talk to the Kubernetes cluster? The way they make their changes is through the API server. If I try to make a change through kubectl or if I try to make a change to the Google Cloud console, the change is submitted to this API server and processed from here. The other two important components which are present in here are the scheduler and the controller manager. The scheduler is responsible for scheduling the Pods onto the nodes. In a Kubernetes cluster, you'll have several nodes. And when we are creating a new Pod, you need to decide which node the Pod has to be scheduled onto. The decision might be based on how much memory is available, and how much CPU is available. Are there any Pod conflicts? And a lot of such factors. So scheduler considers all those factors and schedules the Pods onto the appropriate nodes. The controller manager, manages the overall health of the cluster. Whenever we are executing kubectl commands, we are updating the desired state. The kubectl manager makes sure that whatever desired state that we have we would want 10 instances of application A, we would want 10 instances of application B, we would want five instances of release too. All those changes needs to be executed into the cluster and the controller manager is responsible for that. It makes sure that the actual state of the Kubernetes cluster matches the desired state.

The important thing about a master node is typically the user applications like our Hello World REST API will not be scheduled onto the master node. All the user applications would be running typically in pods inside the worker nodes or just the node. So one of the important components of the node is the applications that we would want to run, the Hello World REST API of web application or things like that. Where would they all be running? They'll be running inside Pods. On a single node, you might have several Pods running. Now, what are the other components which are present on the node? The other components which are present on the node are number one is a node agent. It's called a kubelet, K-U-B-E-L-E-T. What is a job of a kubelet? The job of a kubelet is to make sure that it monitors what's happening on the node and communicates it back to the master node. So if a Pod goes down, what does the node agent do? It reports it to the controller manager. The other component which is present in here is a networking component called kube-proxy. Earlier, we created a deployment and we exposed the deployment as a service. How was that possible? That is possible through the networking component. It helps you in exposing services around your nodes and your Pods. The other important component of the worker node is the container runtime. We'd want to run containers inside our Pods and these need the container runtime. The most frequently used container runtime is Docker. Actually, you can use Kubernetes with any OCI, Open Container Interface, runtime spec implementations. So, we talked a lot about what is present on the master node and what is present on the node or the worker node. Before we end this specific section let's discuss a few important questions. The first question we already talked about, does a master node run any of the application related containers? Does it run Hello World REST API and things like that? We already discussed this, right? The answer to this is no. The master node is typically having only the stuff which is related to what is needed

to control your worker nodes or the nodes. Now, the second question is, can you only run Docker containers in Kubernetes? The answer to that is no. We already talked about this as well, right? If your container is compatible with OCI, Open Container Interface, that is fine. You can run those containers in Kubernetes.
Let's get to the third question which is a very, very interesting one. What happens if the master node goes down or what happens if a specific service on a master node goes down? Will the applications go down?
The answer to that is no. The applications can continue to run working even with the master node down.
When I'm executing a URL to access an application, the master node does not get involved at all.
The only thing that would be involved in those kinds of situations are just the worker nodes,
the nodes which are doing the work. So, even if the master node goes down or the API server goes down,
our applications would continue to be working. You not be able to make changes to them,
but the existing applications would continue to run.

kubectl get componentstatuses :
You can see all the componentstatuses for all the things that we have talked about. You can see that we have two etcd databases, etcd0 and etcd1 These are both healthy. You can see the controller manager, which is healthy. You can see the scheduler, which is healthy as well. So, these are some of the components which are running as part of your Kubernetes master node. In this step, we discussed the different components
which are present in the master node and the worker nodes or just the nodes.

Kubernetes rollouts:
kubectl rollout history deployment deployment_name – it will give history of deployment revisions . Change cause for revisions will be none . So you have no idea why this specific release, what was the command which was executed to do that specific release. Now how do we actually get something into the change cause?
What do we need to do is, when we actually do a set image on the deployment, we will need to add
in a simple option called, record is equal to true. How do we do that?
Let's get started right now.
kubectl set image deployment deployment_name container_name=image --record=true

Now when we update the image by setting record true , then in rollout history change cause will be recorded to above command .

Roll back deployment to previous version : kubectl rollout undo deployment deployment_name --to-revision=revisison_no

One of the interesting things about the rollout is you can also do a pause deployment, or you can do an un-pause deployment. Let's say you are deploying 100 instances or 200 instances. You're scaling to a huge number of instances. And you find that something is a problem. Then, you can also pause the deployment.
So you can say kubectl rollout, pause deployment, hello world, rest, API. And then the deployment would be paused at that specific state. Until now, we have been playing with a lot of deployments. But until now we did not look at anything behind to look at the logs. How do I look at the logs?kubectl logs pod_name

Kubernetes is declarative we can define our desired state in yaml manifests .
To get deployment configuration in yaml format : kubectl get deployment deployment_name -o yaml

Let's save yaml configuration : kubectl get deployment deployment_name -o yaml > deployment.yaml
Let's get service.yaml also in similar way : kubectl get service service_name -o yaml > service.yaml

Let's update replicas to 2 in deployment.yaml file :
Save the changes of deployment file : kubectl apply -f deployment.yaml – deployment will be updated .

Lets create resources using yaml :
Create deployment.yaml and service.yaml for hello-world-rest-api .
Kubectl apply -f deployment.yaml

Kubectl get svc --watch

Lets update deployment.yaml resplicas to 3.

kubectl diff -f deployment.yaml : it will give difference between our existing deployment and configuration mentioned in deployment file .

kubectl get pods

Understand yaml configuration :
There are two things which are defined in this deployment file. One is the deployment, the other one is the service. And if you look at both of them at a high level, they have very, very similar structure. You have a API version, Kind, Metadata, and Spec. Over here in the service as well, you have very very similar thing, API version, Kind, Metadata, and Spec. Now the API version for the deployment is extensions/v1beta1 that specifies the version of the API we are making use of and over here for the service, it's V1. And the Kind for deployment is deployment obviously. And first service is service.

The next thing is the Metadata. Metadata defines information about this specific kind. So over here, this metadata defines the metadata about deployment. So what is the metadata that we are defining in here?
We are defining what is the name of the deployment, what is the name space, which it needs to make use of and what are the labels that are attached for that specific deployment? And if you scroll down, you'd see that similar configuration is present for service as well. We are attaching a name to the service. We are attaching it with a name space and we are giving a few labels for this service. Now in the configuration, in the deployment configuration, what do we need to do? Is we need to match the Pods with the deployment. So we would want to create a few Pods and we would need to match it with the deployment. And that definition is done in the spec. So inside the spec is where we are defining the Pod and we are defining how it needs to be mapped to the deployment.
The most important part in the deployment definition Is this definition which is present in here, Which is the definition of a Pod.A Pod can contain multiple containers. Here, right now we have just one container.So this is the container which we are defining. This is the image for that specific container. We are defining an imagepullpolicy.What would happen if a Docker image Is already locally existing?What we are specifying in here is if not present. If the image is not locally present, only then pull it.The other option would be to say, Always. So Always is always pull the image from the image registry. What i'll do is I'll undo this changeAnd go back to the default, if not present. We are also configuring a name for this specific container.So earlier when we did a deployment, Earlier when we were doing a deployment, We were setting the Hello World REST API is equal to this. We said this is the container name And we are setting it to the new image. And this container name is what is mapping from here.So we are updating the container inside the Pod. So we are updating the container Inside the deployment to the new image. So this is the image we were trying to update When we were executing this specific command. And over here, you can have multiple containers. So similar to this,I can actually define a number of containers over here.That's the reason why you have a hyphen in here which says, 'This is an array.'So a Pod can contain multiple containers. You are defining all the containers down here, You're specifying a restart policy of Always, Which means if there is a failure in starting it up, Try it again and again. So we are telling it to,We are telling the deployment To keep trying to start the Pod again and again. And we are saying termination grace period of 30 seconds. So we are saying, give it a chance to terminate gracefully. So if you want to terminate the Pod, Give it 30 seconds to terminate. So this is the spec where the actual details of the Pod are defined and inside the metadata, we are defining the labels of the Pod. So we are saying, the label on the Pod is app Hello World REST API.So the template over here is completely the definition of the Pod labels and the Pod spec.So we are defining a label of app Hello World REST API for the Pod and we are defining what should be inside the Pod. What kind of containers, what is the imagePullPolicy,what should happen if the Pod does not start up fine? So that's the template.If you scroll up a little bit, we have something called a selector. The selector is how a Pod is mapped to a deployment. So over here, we are specifying a selector of match labels. We are saying, 'Use this label. 'So this deployment would map to any Pod which contains this specific label. So over here, this Pod has this label and we are using that label to map the deployment with the Pods.The other things which are defined in here is how many Pods should be present as part of the deployment? We are saying three pods as part of this deployment and we are specifying the update strategy. So if there is a new deployment, how should the update happen? We are saying, rolling update.Rolling update is basically when you have hundred instances, you would want to update 10 instances at a time,20 instances at a time. You don't want to update all of them at the same time. You'll want to go one step at a time. And over here, you're specifying the max surge and the max unavailable.How many new instances should you create at each point in time?And how many of them can be unavailable? How many of them can be in a status of not ready? Right now, one of the interesting things that you see in here is, we just have one label, which is present

in here.So each one of these have one label, which is the same. What we'll do at a later point in time is we'll assign a couple of labels. So we'll assign two labels and we'll play around with it and try and understand labels in much more depth. Now if you go down to the service over here in the service, what we are defining is a selector to the Pod. So we are defining this label as the way this service maps to the Pod. The important thing to understand is that the service does not map to a deployment. The service directly maps to a Pod. So here, whatever we are placing in here is the selector of the Pod, is the label of the Pod. We are specifying a type of load balancer and we are specifying that the session affinity is none. Let's say you have something like a web application. In that case, you'd want all the requests from one user to go to the same Pod because the session is maintained in that specific Pod. And in that case,you can actually set session affinity to true. But over here we are talking about REST Apis's we'll not use session affinity at all.So we set this session affinity to none.Other than this,the other configuration which is present in hereare the Pod on which the service is made available.So the Pod is 80, 80 and the target Pod is 80, 80and the protocol which is used is also TCP.