**Kubernetes Series | Part 1 - Basic Components.**

Gobalakrishnan Viswanathan

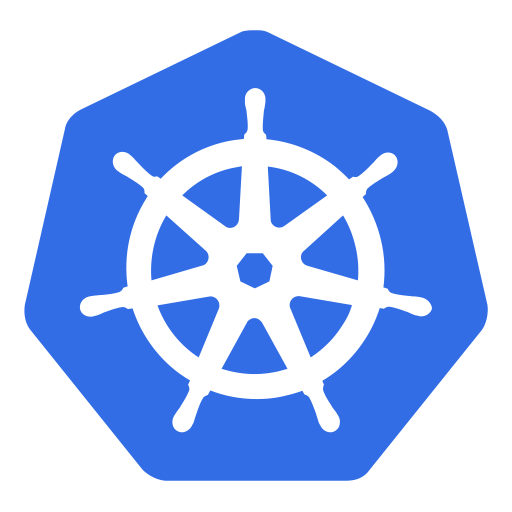
[Gobalakrishnan Viswanathan](https://gobiviswa.medium.com/?source=post_page-----462abfac3ee9--------------------------------)

[Mar 19·5 min read](https://gobiviswa.medium.com/kubernetes-series-part-1-basic-components-462abfac3ee9?source=post_page-----462abfac3ee9--------------------------------)

*Part 2 of the series can be found*[*here*](https://gobiviswa.medium.com/677664724901?source=friends_link&sk=89bac6ef7b4dbbd25978d1432ee3dcf6) *part 3 of the series can be found*[*here*](https://gobiviswa.medium.com/kubernetes-series-part-3-hands-on-4574c41de734)

From its [official site](https://kubernetes.io/)

**Kubernetes, also known as K8s, is an open-source system for automating deployment, scaling, and management of containerized applications**



It **groups containers** **that make up an application into logical units** for easy management and discovery. *K8s is developed by Google*.

* K8 is a **Opensource, Container Orchestration Framework/tool**.
* Means, K8s used to manage the applications made up by hundreds/thousands of containers in different deployment environments like physical machines, cloud machines etc.

**Need of Container Orchestration tool:**

* Usage of microservices increased the possibility of container technologies. Because containerization is the best way to host the small, independent services known as micro-services.
* Now the raise of containers usage due to the micro-service architecture resulted that each application is depends on hundreds/thousands of containers.
* **Managing these high amount of containers in different environments using script/self made tools is not easy anymore**. This actually cause the real need for the orchestration tool that manages the containers of the application. Managing here means giving **High Availability, Scalability and Disaster recovery** to the applications.

Here comes Kubernetes. Now, lets learn the basic components of K8s.

Kubernetes Components:

**Pods:**

* Pods are the smallest/basic unit you can create and manage in K8s. Pod is group of one or more containers with shared storage and network resources, have a specification on how to run the containers.
* Pods are something like a abstraction layer on top of the containers, creates the running environment for them.**The abstraction is enforced by K8s because we only need to interact with Its own layers but not with the actual containers**. Means that, Existing container images in the Kubernetes can be replaced by any other technology containers easily.
* Usually one container per Pod is the default scenario unless when the main application needs some small services(tightly coupled) which can be run in the same Pods.

**Nodes:**

* Pods runs on a Node. Node is a worker machine either virtual/physical servers. A node can have multiple Pods. **K8s master** automatically handles scheduling the **pods** across the **Nodes** in the cluster.

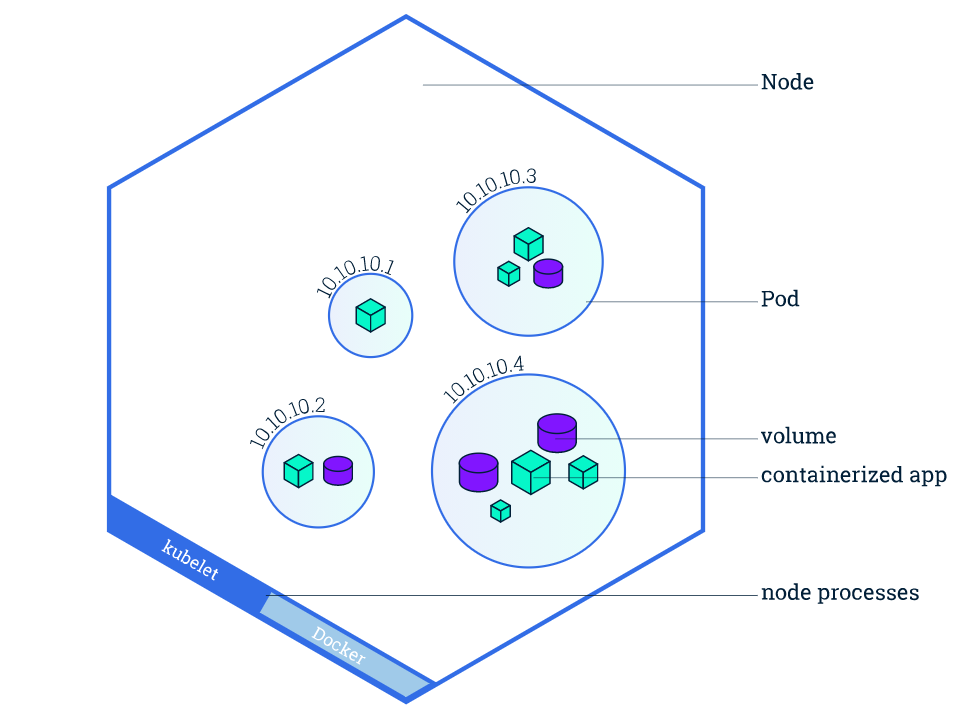


Image from official website

As seen in the above image, Node consists many Pods. Pods are having its own internal IP address which will be used to communicate between each other. Every Pods having their own storage and container images to be run. **These IP address will be static/permanent**. When specific Pod is not running, A new pod will be created with the same IP so that the connection configuration no need to be changed.

* **Kubelet, responsible for communication between the Kubernetes Master and the Node**; it manages the Pods and the containers running on a machine.
* There is a components named **Service & Ingress** plays important role. ***Service****used to give static IPs to the Pods*. All the requests from the outside world to your application domain will be coming to **Ingest**first. Then this Component will forward the the requests to the specific requested Pod.

Till now we have seen very basic components of K8s. We yet to see the real cool features of it.

**Config Map & Secret:**Pods can connect to each other using **Service** feature. To make this connection, logically there should be somewhere the configurations needs to be stored right ? The man is named as configMap.

* Config Map is an Object used to store non-confidential data as a key-value pair.
* Pods use this Map to access between each other. This config map can be used as environmental variables, command-line arguments or a configuration files.
* This config Map allows you to decouple the configuration from your containers to make sure the containers are portable. For example, in your application, you need to replace your Sqlite3 database with Mongo-db database. For this change, you just need to change in the ConfigMap and place your container in the Node and nothing else.
* Placing secret information like database connection details in ConfigMap is not advisable because this is just a plain text, no encryption applied to the data. For storing this, we can use **Secret**component.

**Volume:**The data our application generating should be persistent, means when container gets restarted, data should not lost. **Volumes** is the component to achieve that. This can store the data in local machine means the node where the Pod running or to the remote storage outside the K8s cluster (both cloud storage or own storage cluster).

**What if my application Pod dies?**Lets take my application Pod is dead and my user accessing it, it shows site cant reachable which is not good thing. To make sure this is not happening, everything in K8s is replicated. Means that, the application pod will be replicated to the another Node based on the configuration we provide. All these replicated Pods connected to the same service(remember IP is permanent). So replicated Pods also can be accessed with the same IP. **Service component also does the Load balancing with these replicas**.

**Deployments:**Deployments are the blue-prints for the Pods. We create deployments to give updates regarding Pods and replicas. We know that, Pods are the abstraction layer for the container images. Deployments are kind of abstraction layers for the Pods. So in practice, most of the time we will work with Deployments not with Pods. With this Deployment, we can replicate the application Pods but not the Database Pods. Because database has state means data. if we replicate the DB Pod, inconsistency between the replicas may occur. **StatefulSet**is another component is K8s for this purpose.

So all the database applications should be created using **StatefulSet** but not with **Deployments**. Statefulset will take care the database is synchronized with the service and its replicas. But, doing this is not so easy in K8s. Common practice is running the database application outside the K8 cluster.

Summarize:

* Pod component is an abstraction layer for the container image(s) which run in the Node(server). One or more Pods can run in the server.
* Service component is used to have the communication between the Pods.
* Ingress component is route the traffic to the K8 cluster.
* Configuration for the Pods can be stored in the ConfigMap and Secrets.
* Data storage is happening using Volumes component.
* Deployments & Statefulsets are for Pods replica configurations.

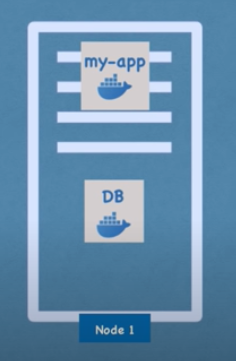
Yes, there are lot of components available in K8s. But these are very important to start with. With these, we can built very powerful k8s cluster. That’s it for now on the basic components. Will meet you with the Architecture of K8s soon.

**Kubernetes series| Part 2 -Architecture**

Part 1 of the series can be found [*here*](https://gobiviswa.medium.com/kubernetes-series-part-1-basic-components-462abfac3ee9?source=friends_link&sk=f261c43819097d977d062ccc40dafd21)*.*Part 3 of the series can be found [*here*](https://gobiviswa.medium.com/kubernetes-series-part-3-hands-on-4574c41de734).

In the first part, we have seen some important components in the K8s cluster. In this part lets talk about the components in the Architecture.

**Node processes:**Let’s have a basic setup of one name node with two application Pods like below.



* One of the main components of K8s architecture is worker servers or nodes.
* Each node can have one or more application pods where containers will be placed. Nodes are the servers which actually does the work.
* Three important process must be installed in every nodes to manage Pods. Lets discuss these.

1. **Container-Runtime**is a software that runs and manages the components required to run **containers**. This tool make it easier to securely execute and efficiently deploy **containers**, a key component of **container** management.
2. **Kubelet** is the primary “node agent” that runs on each node. It is actually registering the node server to the K8s. Kubelets working with the configuration called PodSpec. PodsSpec is a Yaml or Json object which actually describes the Pod with all the details. Kubelet ensures that all the containers described in the PodSpec is running fine and healthy by interacting with nodes and containers, also takes care of assigning resources to the containers.
3. The communication between the Pods handled by the Service component which is actually load balances and route the request to the specific Pods. The actual **process responsible for forwarding requests from Service component to Pods is Kube-Proxy**which should be installed in all the nodes. It also make sure that communication between Service and Pods is performant with low overhead.

**Master Processes:**Master nodes are completely different processes running inside, which controls and manages a set of worker nodes. 4 process should run in every master node to control the cluster state and worker nodes.

1. **API Server:**When we deploy new application in the K8s Cluster, We interact with the API Server through UI, Command Line tool or K8s API. This is like a gateway to the cluster to do all the changes in the cluster configurations.  
   Also acts as a Authentication gate keeper to allow only autheticated requests to the cluster. This is the only entry point to the K8s Cluster.
2. **Scheduler:**When we give request to schedule new Pod, the API Server will validate the request and sends the request to the Scheduler. Then Scheduler will start the Pod in one of the worker node. Scheduler will see the resources requested for the application Pod, have the intelligence to assign it to the best worker node based on the resources available. Scheduler will decide which node the application should go to, then **Kubelet is the process will start the container Pod with the specifications given by the Scheduler**.
3. **Controller Manager**Controller manager is responsible for re-schedule of the applications when crash happens. It detects the Cluster state changes. For example, if pod dies, controller manager will find this as soon as possible and sends the request to the Scheduler to re-schedule the specific Pod. Then the Scheduler will do the request to Kubelet to schedule the Pod.
4. **etcd**It is a **Key-Value store** to the cluster state. All the changes happening in the cluster like new pod creation, Pod crash, Pod re-scheduling will be updated in this store. All the above three process API Server, Scheduler and Controller-Manager will use this key value store data to know the current state of the Cluster. So this etcd component can be called as Cluster Brain which as all the data about the K8s cluster. This store is only having cluster state data but not the application’s data which is running in the Pods.

Now, we seen important master processes and how crucial these especially etcd data. This data must be reliably stored means replicated. In practice, K8s cluster is made up of more than one master nodes. All the master nodes run its own processes where etcd data store is distributed across the master nodes.

**Kubernetes series | part 3 - hands on**

Gobalakrishnan Viswanathan

[Gobalakrishnan Viswanathan](https://gobiviswa.medium.com/?source=post_page-----4574c41de734--------------------------------)

[Mar 24·7 min read](https://gobiviswa.medium.com/kubernetes-series-part-3-hands-on-4574c41de734?source=post_page-----4574c41de734--------------------------------)

*Previous parts of the series can be found here*[***part-1***](https://gobiviswa.medium.com/kubernetes-series-part-1-basic-components-462abfac3ee9?source=friends_link&sk=f261c43819097d977d062ccc40dafd21)***,***[***part-2***](https://gobiviswa.medium.com/kubernetes-series-part-2-architecture-677664724901?source=friends_link&sk=89bac6ef7b4dbbd25978d1432ee3dcf6)***.***



In this post, Lets have our fingers dirty with Kubernetes. There are few ways available to have the Kubernetes setup.

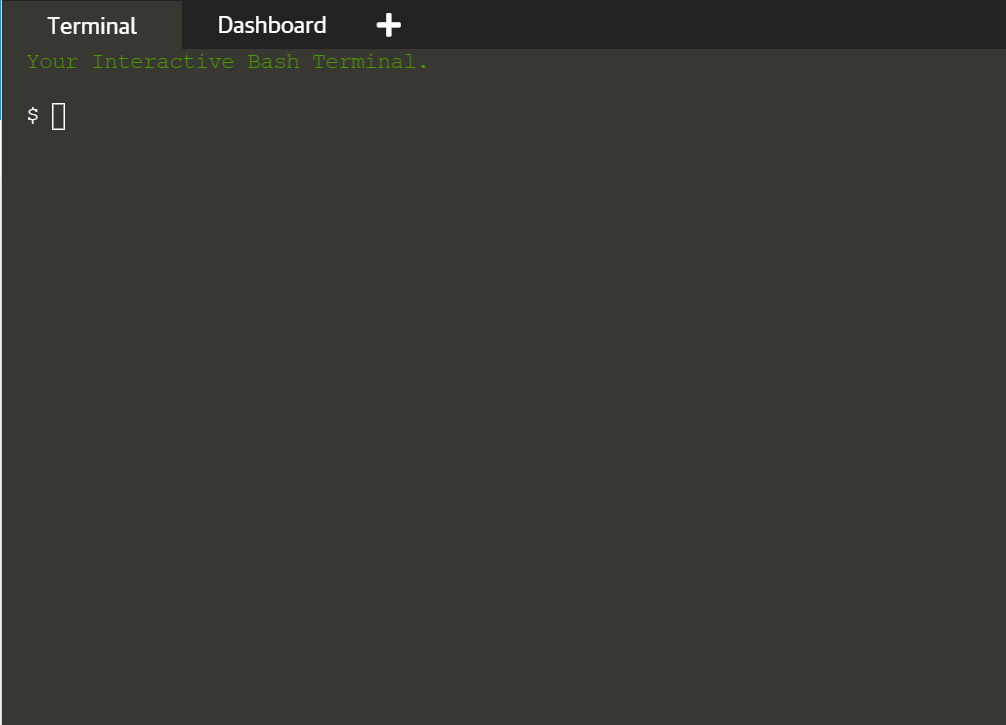
1. Cloud based K8s cluster setup like GCP, Azure or AWS
2. Installing K8s tools like Minikube & Kubectl
3. Using online playgrounds to have hands-on

Cloud based cluster setup is suitable for multi node cluster which is obviously not necessary to get initial hands-on. Installing Minikube & Kubectl is a good option so that we can have a single node cluster where both master & worker nodes will be placed. But it will take some time to get started. So my choice here is to go with ready-to-use setup. Here, I am gonna try [Katacoda](https://www.katacoda.com/courses/kubernetes/) .My heartful thanks to the Katacoda team for this free tool .

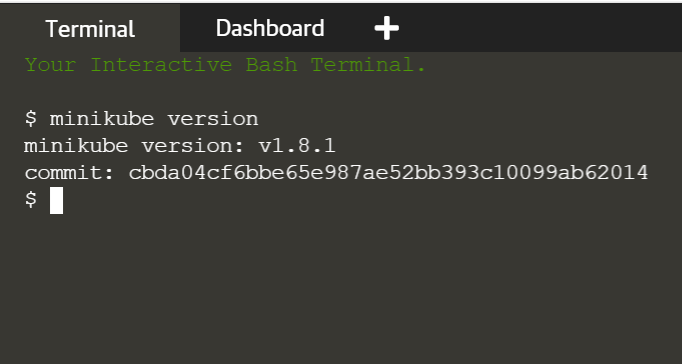
Let’s start…

Go to the [**Katacoda**](https://www.katacoda.com/courses/kubernetes/)home page for Kubernetes. There we can see lot of scenarios given by the team to explore the K8s. For now, Lets click Start scenario button of **Launch A Single Node Cluster**block. This Single Node Cluster given by Katacoda itself using **Minikube .**Click on the Start scenario to get into the terminal world of K8s.

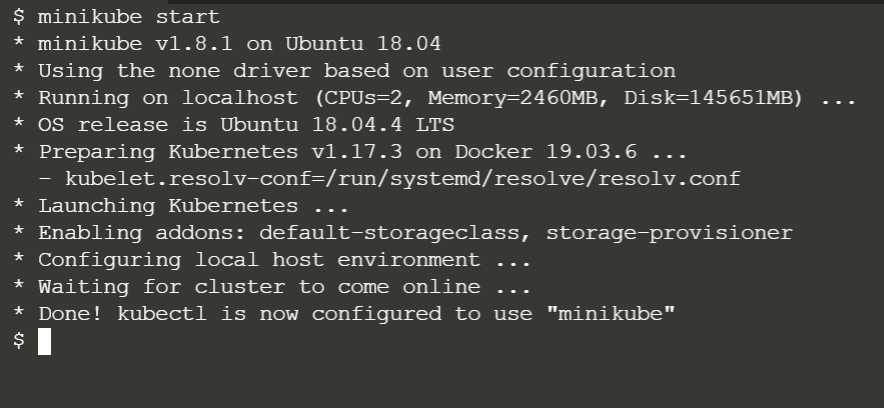
***Minikube****is a tool that makes it easy****to run Kubernetes locally****. Minikube runs a single-node Kubernetes cluster inside a VM on your laptop for users looking to try out Kubernetes or develop with it day-to-day.*



We can see the empty terminal once we click the Start button, this is nothing but a ubuntu terminal. All the required tools like Minikube are already installed in the server. We can validate required tools installed properly with below commands.

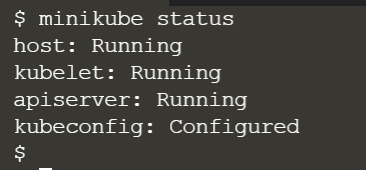


Use below command to start our single node cluster using Minikube tool.



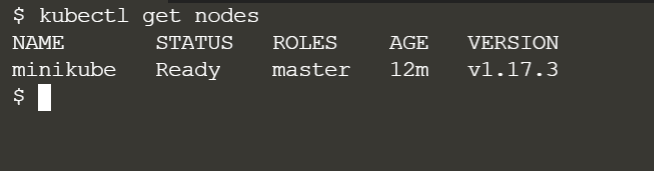
Here go, we have started out single node cluster using Minikube tool. Last line of the log says, kubectl is configured to use with Minikube means that **Kubectl is a command line tool to interact with Minikube cluster**. Minikube is pre-installed with Docker, even we don’t have Docker installed, it is going to work.

To get the status of the cluster:

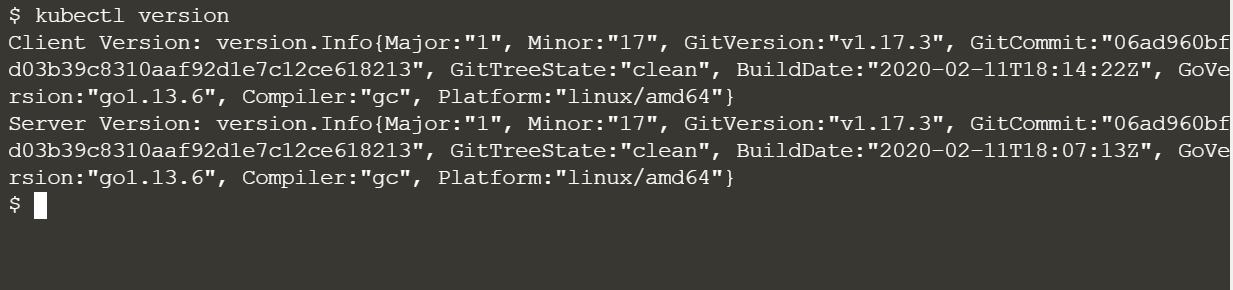


**minikube status**to get the status of the cluster. It is giving status of some of the processes running in the cluster.

From the above log,  
**\* apiserver**is a entry point to the K8s cluster. To do all the



**kubectl get nodes**to get the nodes information of the cluster. For now there is single node running as a master.

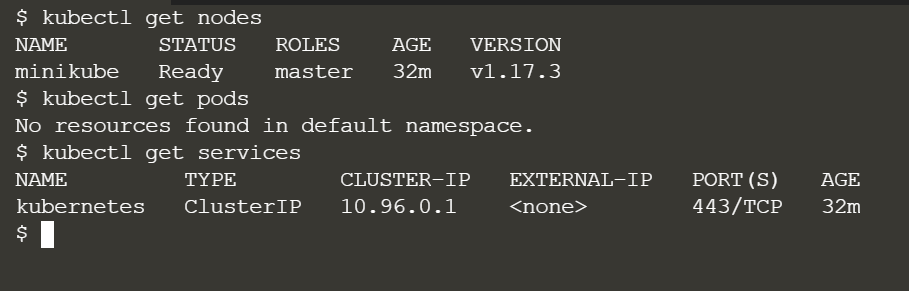


**kubectl version**to see the client & server version of the Kubernetes. Here, v1.17.3 is the kubernetes verison we are currently have in katacoda tool.

we have seen two command line tools as of now. Those are Minikube and Kubectl. Minikube is just to start and stop the cluster. For all other operations with the cluster, we will use kubectl.

**Basic kubectl commands:  
kubectl get**command is to get status and configuration details of the components. for example,

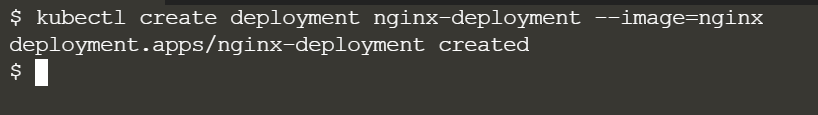
**kubectl get nodes :**to get the node details of the cluster  
**kubectl get pods :**to get the pod running in the server  
**kubectl get services :**to get list of all services in the namespace. There is many get commands available.



Since our katacode cluster is new, we don’t have any pod running. Lets create a Pod now.

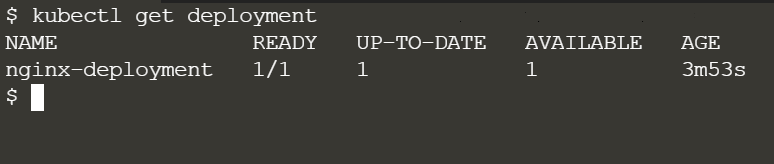
**Creating Pod / Deployment:**We know that Pod is the smallest unit of the cluster. We will not usually work with Pods. We will work with the component called **Deployment**which is like a abstraction layer to the Pods.

we can create Deployment using **kubectl create**command. Once created, the Deployment ensures that the desired number of Pods are running and available at all times.

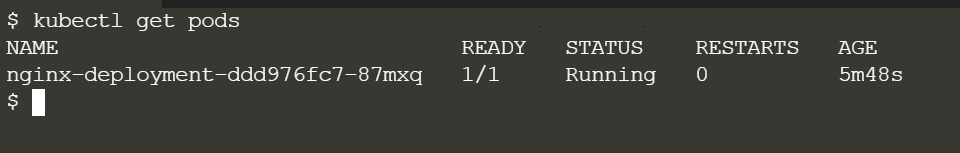


* nginx-deployment is the deployment name, — image specify which image to use.
* Docker is pre-installed in Minikube. So it can get the nginx docker image from the dockerhub. Using that, the deployment will be created.
* Deployment component will have blueprint on how to create the deployment including default configurations. Here, deployment name and image name are the two required parameters. Then default patameters are used here.

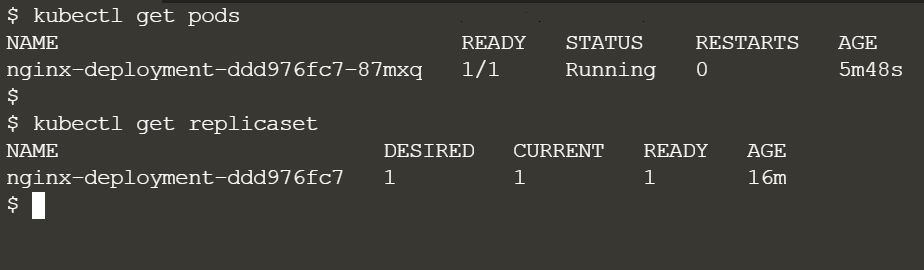
Now lets see the status of the deployment. The below image says, deployment is ready.



So obviously now we should see the output for **kubectl get pods** command. The output below says, there is a Pod running with a name **nginx-deployment-ddd976fc7–87mxq.**



* Here one concept we need to discuss is replicaset. Replicaset is managing the replicas of the Pod. This will be automatically created by the deployment command. We can see the replicaset using below command.



If you see the replicaset name and Pod name, we can understand how this works. So Replicaset is a copy of a Pod, we can use this copy to replicate the crashed/dead Pod. If I understand this correctly, the actual Pod itself created by using this Replicaset only.

For summary,

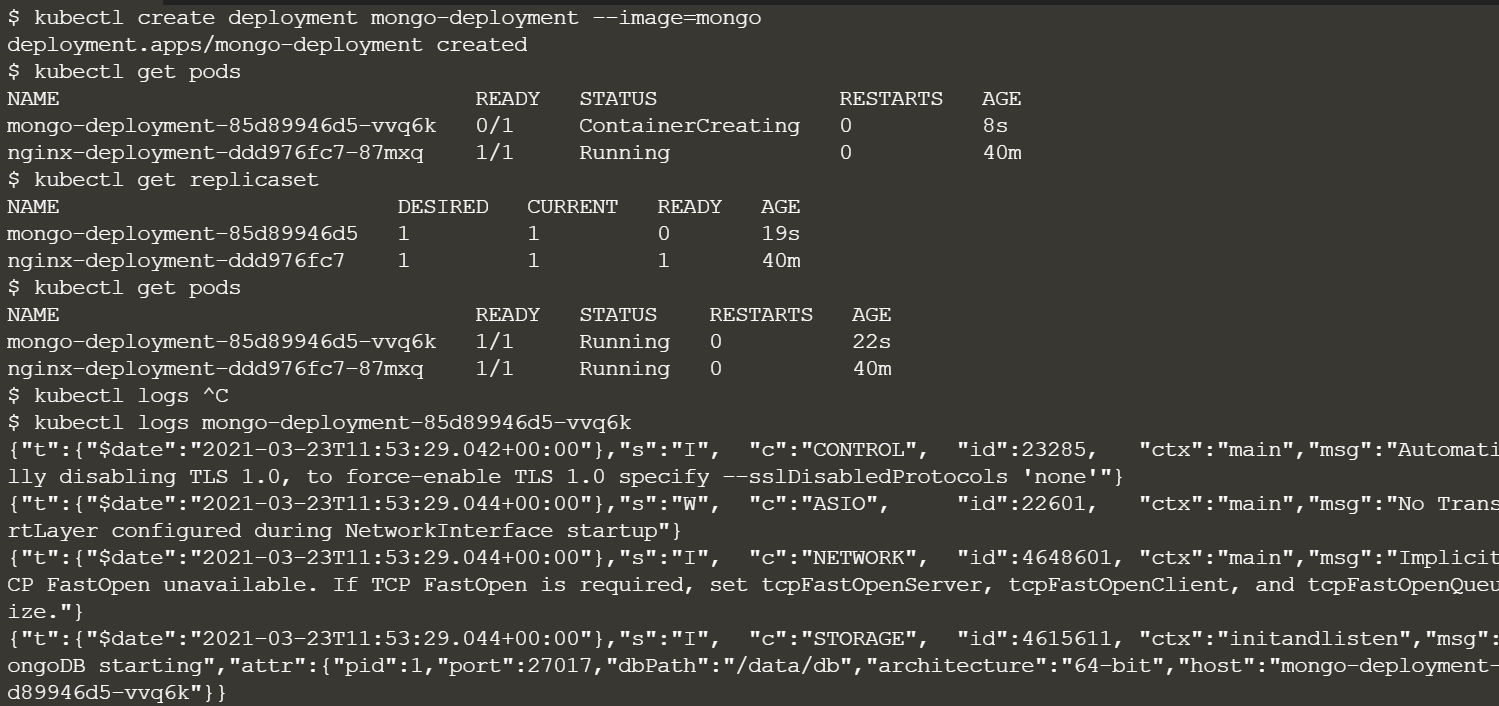
* **Deployment** is managing the Replicaset
* **Replicaset** manages all the replicas of the Pod
* **Pod** is an abstraction of the container images
* And finally **Container** contains our actual applications.

From all above 4, we will work only with Deployments most of the time. other three components will be automatically managed by the K8s.

**Edit the Deployment:  
kubectl edit**command is use to edit the available deployment. This command will open up the Yaml file of the Deployment. This is auto generated config file created by the **kubectl** **create deployment** command with the default values.

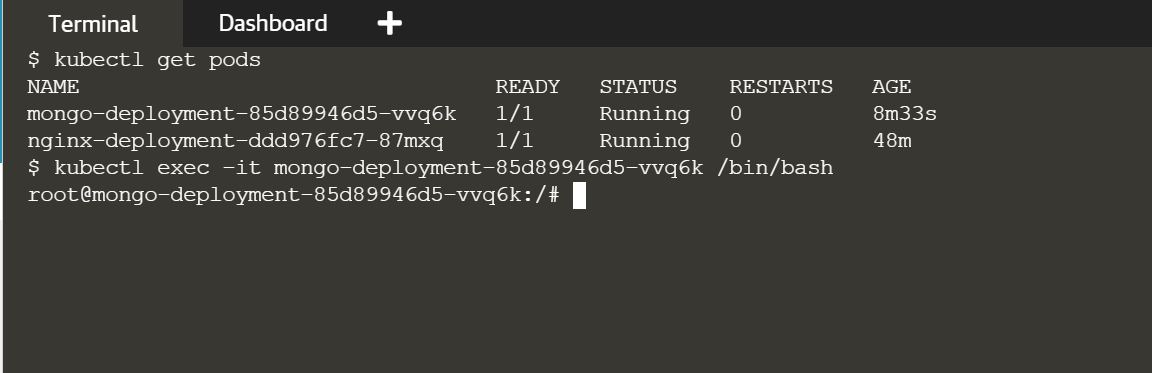
**kubectl edit deployment nginx-deployment**

**How to debug the Pods:**using **kubectl logs <image\_name> ,**we can get the actual log that is being printed by the container image which is inside the pod. I didn’t get as much logs with nginx Pod. So I created new mongodb Pod to see logs command works. Below image shows the output.



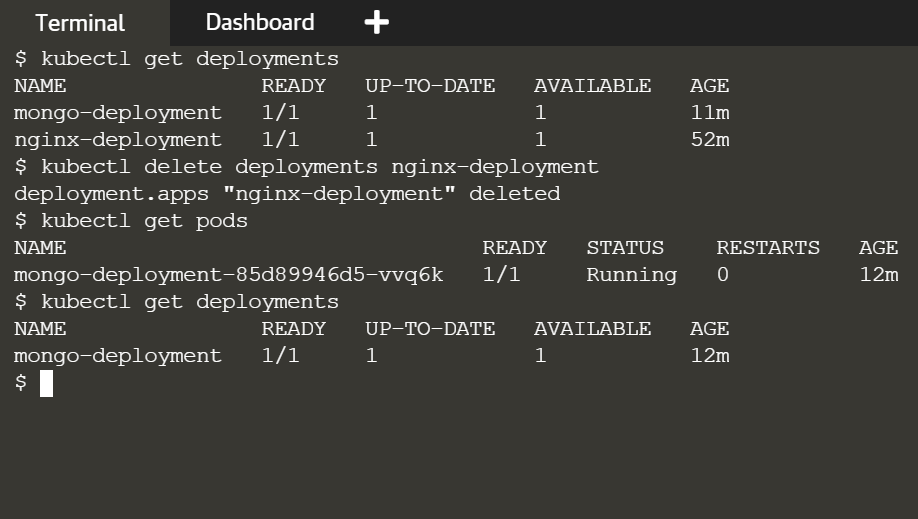
Once I ran the create command for mongodb, deployment created, after some time Pod came to running state. Then I ran the logs command & it printed all the logs from the MongoDB container. This will be very usefult to debug the containers inside the Pod.

Another most useful debugging command is exec. it will open the terminal of the application running in the Pod.



As seen above, by using the exec command I entered the Mongodb terminal where I can run commands to debug the application.

**Deleting the deployment:  
kubectl delete**command used to delete the deployment.



**Deployment creation using config file:**

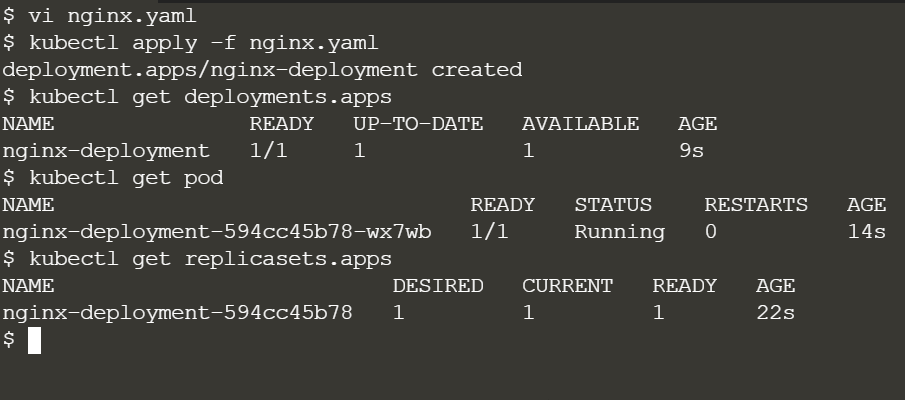
we are creating the deployments using the command line by giving the required parameters name and image. But in production, we may need to give lot of options for deployment creation. So writing all the options in the command line is not a feasible way. For that, we can add all the configuration options in the file and pass it to the create command.

lets create a file named nginx-deployment.yaml and add the following content.

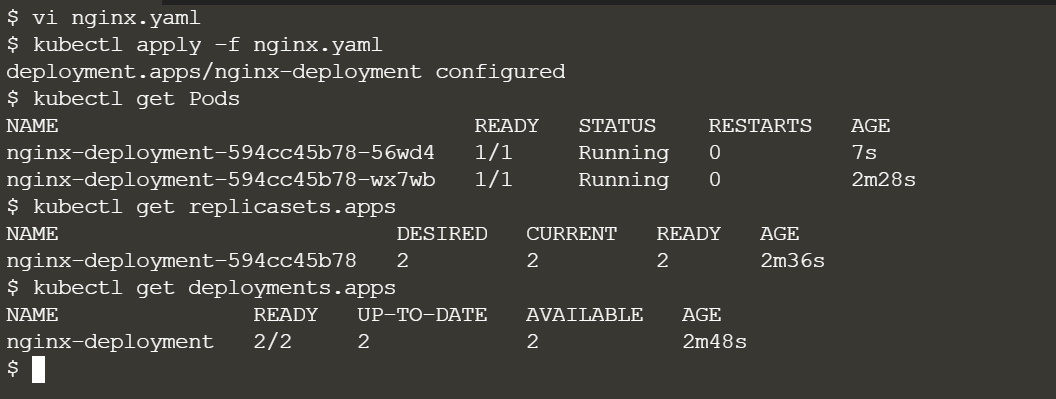
**apiVersion: apps/v1  
kind: Deployment  
metadata:  
 name: nginx-deployment  
 labels:  
 app: nginx  
spec:  
 replicas: 1  
 selector:  
 matchlabels:  
 app: nginx  
 template:  
 metadata:  
 labels:  
 app: nginx  
 spec:  
 containers:  
 - name: nginx  
 image: nginx:1.16  
 ports:  
 - containerPort: 80**

This is a basic content needed to create the nginx Pod using the file. Now deployment can be created using this file with apply command.

**kubectl apply -f mongo.yaml**command will read the deployment config file and created the deployment like below.



Now, whatever changes we want to do to the deployment, we can go and change the nginx.yaml file. Then the change will be automatically updated to the deployment. for example, I am going to change the replicas to 2 in the file.



Here we can see two Pods running as per the configuration given.

Summary

We got hands-on with the following concepts as of now,

* CRUD deployment commands  
  1. Create: **kubectl create deployment <name> image**2. Edit: **kubectl edit deployment <name>**3. Delete: **kubectl delete deployment <name>**
* Status of diffetent Pods  
  **kubectl get nodes | pod | services | replicaset | deployment**
* Debugging Pods:  
  Get logs of the Pod: **kubectl logs pod\_name**More info of pod: **kubectl describe pod pod\_name**
* Get interactive terminal  
  **kubectl exec -it pod\_name — /bin/bash**
* Deployment with config file  
  1. create/update:**kubectl apply -f config\_file\_name**2. Del ete: **kubectl delete -f config\_file\_name**