Kubernetes Tutorial – A Comprehensive Guide For Kubernetes

Last updated on Nov 27,2019 *23.7K Views*

Sahiti Kappagantula

Kubernetes is a platform that eliminates the manual processes involved in deploying containerized applications. In this blog on Kubernetes Tutorial, you will go through all the concepts related to this multi-container management solution.

The following topics will be covered in this tutorial:

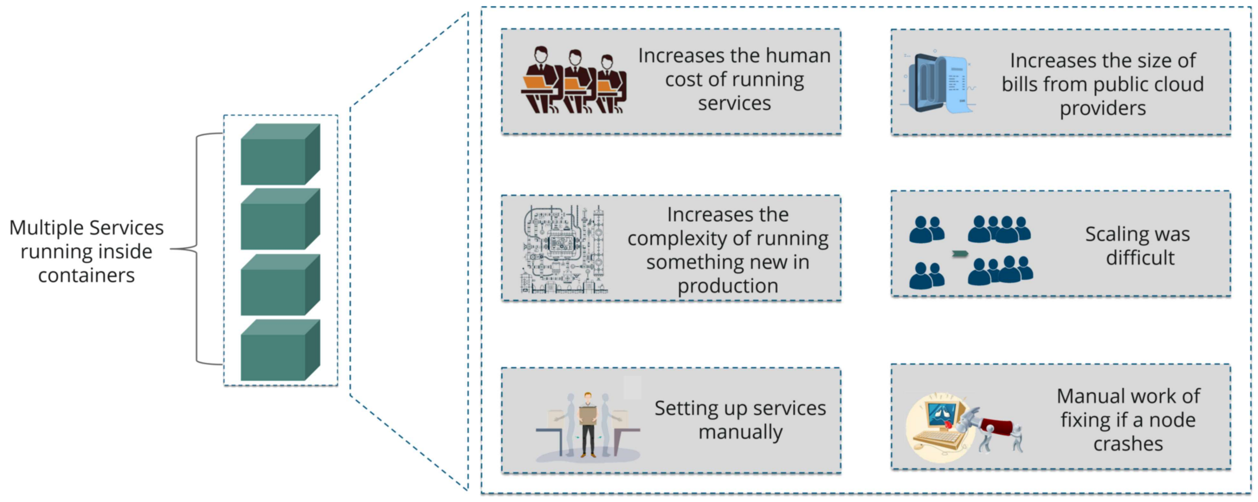
Now, before moving forward in this blog, let me just quickly brief you about containerization.

So, before containers came into existence, the developers and the testers always had a tiff between them. This usually, happened because what worked on the dev side, would not work on the testing side. Both of them existed in different environments. Now, to avoid such scenarios containers were introduced so that both the Developers and Testers were on the same page.

Handling a large number of containers all together was also a problem. Sometimes while running containers, on the product side, few issues were raised, which were not present at the development stage. This kind of scenarios introduced the Container Orchestration System.

Before I deep dive into the orchestration system, let me just quickly list down the challenges faced without this system.

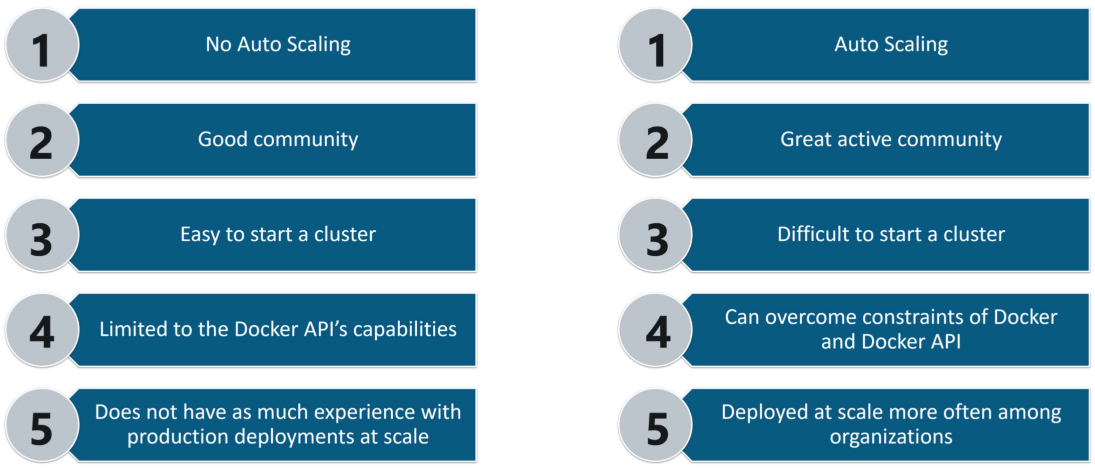
Kubernetes Tutorial: Challenges Without Container Orchestration


Multiple Services running inside containers
Increases the human cost of running services
Increases the size of bills from public cloud providers
Increases the complexity of running something new in production
Scaling was difficult
Setting up services manually
Manual work of fixing if a node crashes

As you can see in the above diagram when multiple services run inside containers, you may want to scale these containers. In large scale industries, this is really tough to do. That's because it would increase the cost to maintain services, and the complexity to run them side by side.

Now, to avoid setting up services manually & overcome the challenges, something big was needed. This is where Container Orchestration Engine comes into the picture.

This engine, lets us organize multiple containers, in such a way that all the underlying machines are launched, containers are healthy and distributed in a clustered environment. In today's world, there are mainly two such engines: **Kubernetes** & **Docker**

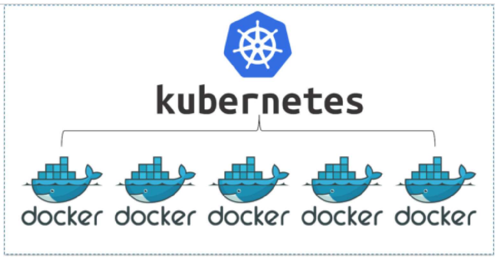


1	No Auto Scaling	1	Auto Scaling
2	Good community	2	Great active community
3	Easy to start a cluster	3	Difficult to start a cluster
4	Limited to the Docker API's capabilities	4	Can overcome constraints of Docker and Docker API
5	Does not have as much experience with production deployments at scale	5	Deployed at scale more often among organizations

As you can refer to the above image, Kubernetes, when compared with Docker Swarm owns a great active community and empowers auto-scaling in many organizations. Similarly, Docker Swarm has an easy to start cluster when compared to Kubernetes, but it is limited to the Docker API's capabilities.

Well, folks, these are not the only differences between these top tools. If you wish to know the detailed differences between both these container orchestration tools, you can click here.

If I could choose my pick between the two, then it would have to be Kubernetes since, containers need to be managed and connected to the outside world for tasks such as scheduling, load balancing, and distribution.



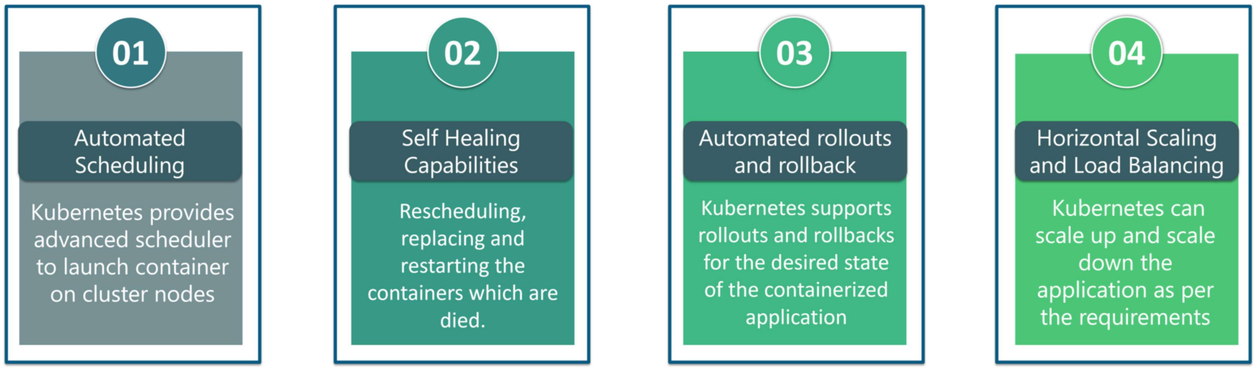
But, if you think logically, Docker Swarm would make a better option, as it runs on top of Docker right? If I were you, I would have definitely got confused about which tool to use. But hey, Kubernetes being an undisputed leader in the market and also does run on top of Docker containers with better functionalities.

Now, that you have understood the need for Kubernetes, it's a good time, that I tell you **What is Kubernetes?**

Kubernetes is an open-source system that handles the work of scheduling containers onto a compute cluster and manages the workloads to ensure they run as the user intends. Being the Google's brainchild, it offers excellent community and works brilliantly with all the cloud providers to become a *multi-container management solution.*

Kubernetes Tutorial: Kubernetes Features

The features of Kubernetes, are as follows:


01
Automated Scheduling
Kubernetes provides advanced scheduler to launch container on cluster nodes
02
Self Healing Capabilities
Rescheduling, replacing and restarting the containers which are died.
03
Automated rollouts and rollback
Kubernetes supports rollouts and rollbacks for the desired state of the containerized application
04
Horizontal Scaling and Load Balancing
Kubernetes can scale up and scale down the application as per the requirements

- **Automated Scheduling:** Kubernetes provides advanced scheduler to launch container on cluster nodes based on their resource requirements and other constraints, while not sacrificing availability.
- **Self Healing Capabilities:** Kubernetes allows to replaces and reschedules containers when nodes die. It also kills containers that don't respond to user-defined health check and doesn't advertise them to clients until they are ready to serve.
- **Automated rollouts & rollback:** Kubernetes rolls out changes to the application or its configuration while monitoring application health to ensure it doesn't kill all your instances at the same time. If something goes wrong, with Kubernetes you can rollback the change.
- **Horizontal Scaling & Load Balancing:** Kubernetes can scale up and scale down the application as per the requirements with a simple command, using a UI, or automatically based on CPU usage.

Kubernetes Tutorial: Kubernetes Architecture

Kubernetes Architecture has the following main components:

- Master nodes
- Worker/Slave nodes

I am going to discuss each one of them one by one. So, initially let's start by understanding the **Master Node**.

Master Node

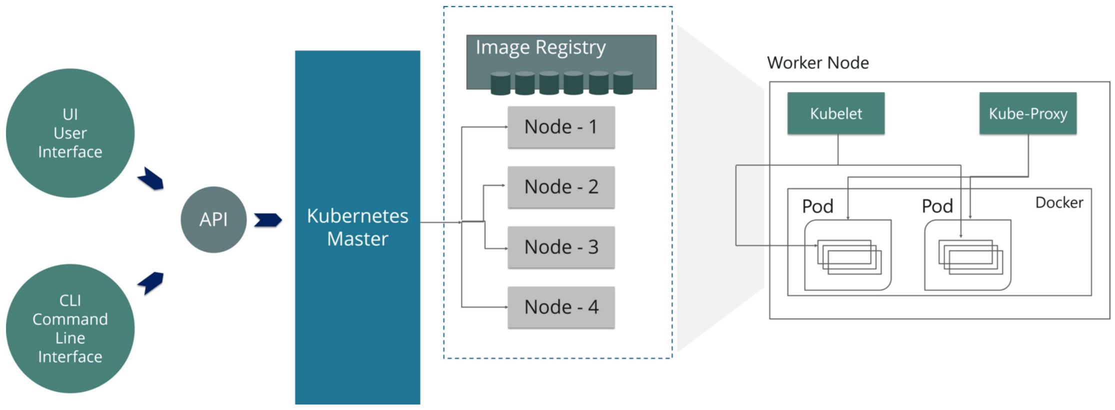
The master node is responsible for the management of Kubernetes cluster. It is mainly the entry point for all administrative tasks. There can be more than one master node in the cluster to check for fault tolerance.

As you can see in the above diagram, the master node has various components like API Server, Controller Manager, Scheduler and ETCD.

- **API Server:** The API server is the entry point for all the REST commands used to control the cluster.
- **Controller Manager:** Is a daemon that regulates the Kubernetes cluster, and manages different non-terminating control loops.
- **Scheduler:** The scheduler schedules the tasks to slave nodes. It stores the resource usage information for each slave node.
- **ETCD:** ETCD is a simple, distributed, consistent key-value store. It's mainly used for shared configuration and service discovery.

Worker/Slave nodes

Worker nodes contain all the necessary services to manage the networking between the containers, communicate with the master node, and assign resources to the scheduled containers.



As you can see in the above diagram, the worker node has various components like Docker Container, Kubelet, Kube-proxy, and Pods.

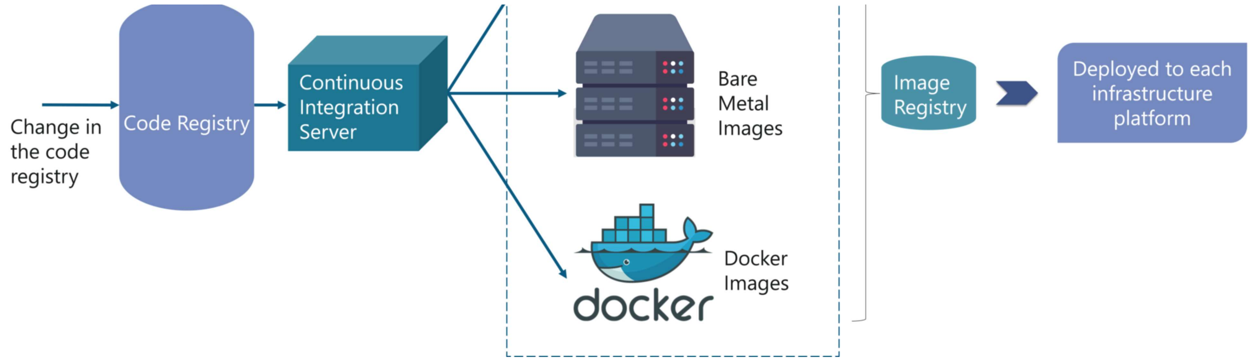
- **Docker Container:** Docker runs on each of the worker nodes, and runs the configured pods
- **Kubelet:** Kubelet gets the configuration of a Pod from the API server and ensures that the described containers are up and running.
- **Kube-proxy:** Kube-proxy acts as a network proxy and a load balancer for a service on a single worker node
- **Pods:** A pod is one or more containers that logically run together on nodes.

If you want a detailed explanation of all the components of Kubernetes Architecture, then you can refer to our blog on Kubernetes Architecture.

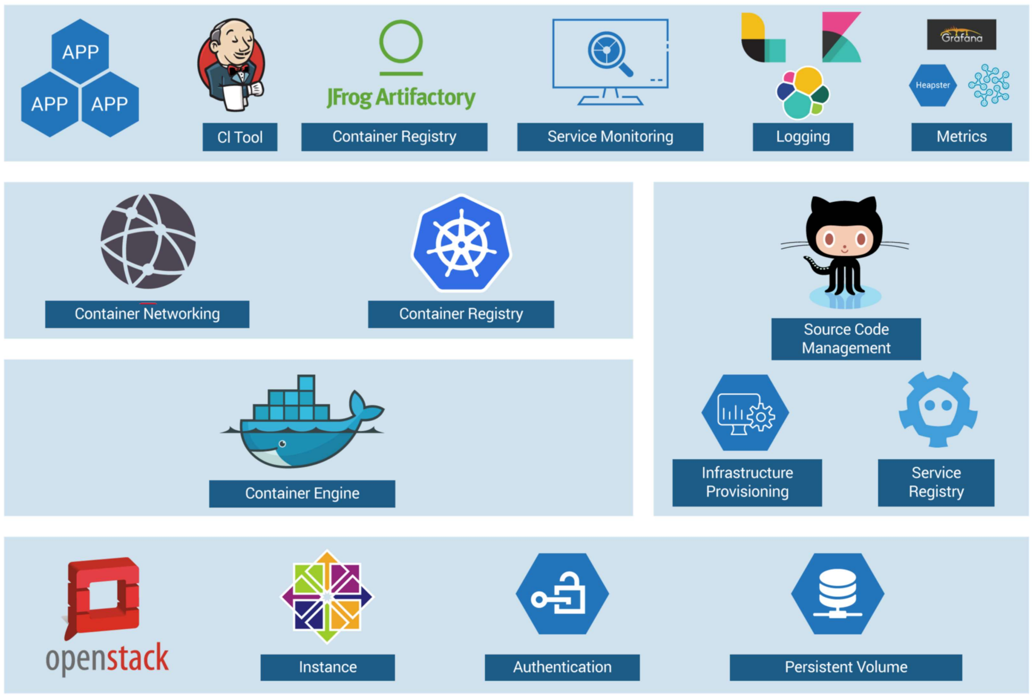
Kubernetes Tutorial: Kubernetes Case-Study

Yahoo! JAPAN is a web services provider headquartered in Sunnyvale, California. As the company aimed to virtualize the hardware, company started using **OpenStack** in 2012. Their internal environment changed very quickly. However, due to the progress of cloud and container technology, the company wanted the capability to launch services on various platforms.

Problem: How to create images for all required platforms from one application code, and deploy those images onto each platform?


Change in the code registry
Code Registry
Continuous Integration Server
Bare Metal Images
Docker Images
docker
Image Registry
Deployed to each infrastructure platform

Now, let us focus on container workflow to understand how they used Kubernetes as a deployment platform. Refer to the below image to sneak peek into platform architecture.



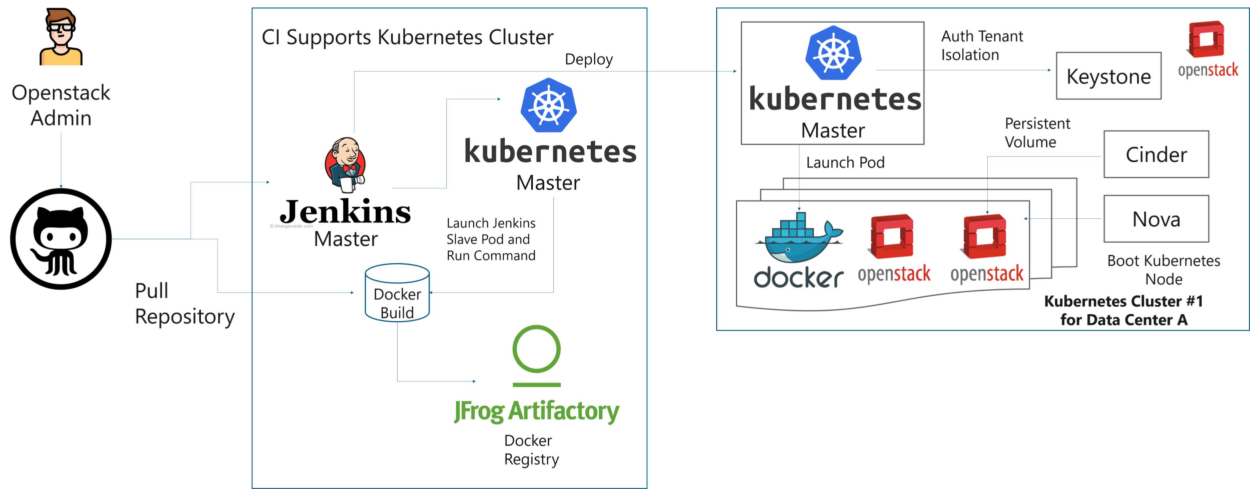
OpenStack instances are used, with Docker, Kubernetes, Calico, etcd on top of it to perform various operations like Container Networking, Container Registry, and so on.

vOps Training

When you have a number of clusters, then it becomes hard to manage them right?

So, they just wanted to create a simple, base OpenStack cluster to provide the basic functionality needed for Kubernetes and make the OpenStack environment easier to manage.

By the combination of Image creation workflow and Kubernetes, they built the below toolchain which makes it easy from code push to deployment.



This kind of toolchain made sure that all factors for production deployment such as multi-tenancy, authentication, storage, networking, service discovery were considered.

That's how folks, **Yahoo! JAPAN** built an automation toolchain for "one-click" code deployment to Kubernetes running on OpenStack, with help from **Google** and **Solinea**.

Kubernetes Tutorial: Hands-On

In this Hands-On, I will show you how to create a deployment and a service. I am using an Amazon EC2 instance, to use Kubernetes. Well, Amazon has come up with **Amazon Elastic Container Service** for **Kubernetes (Amazon EKS)**, which allows them to create Kubernetes clusters in the cloud very quickly and easily. If you wish to learn more about it, you can refer to the blog here.

Step 1: First **create a folder** inside which you will create your deployment and service. After that, use an editor and **open a Deployment file**.

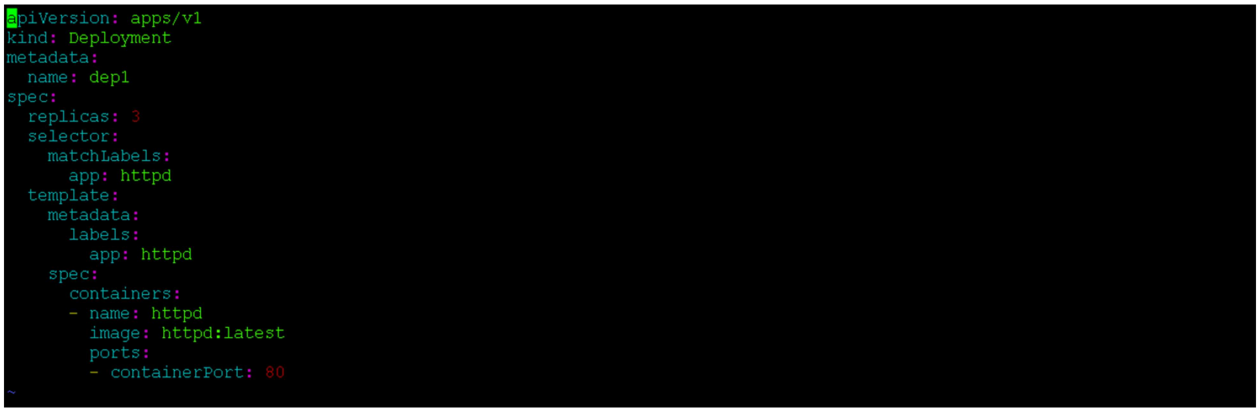
```
mkdir handsOn
cd handsOn
vi Deploy.yaml
```

```
ubuntu@kmaster:~/handsOn$ vi Deploy.yaml
```

Step 2: Once you open the deployment file, mention all the specifications for the application you want to deploy. Here I am trying to deploy an **httpd** application.

```
apiVersion: apps/v1  #Defines the API Version
kind: Deployment     #Kinds parameter defines which kind of file is it, over here it is Deployment
metadata:
```

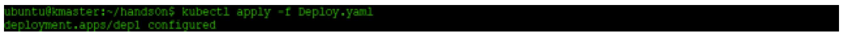
```
        - containerPort: 80 #The application would be exposed on port 80
```



```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: dep1
spec:
  replicas: 3
  selector:
    matchLabels:
      app: httpd
  template:
    metadata:
      labels:
        app: httpd
    spec:
      containers:
      - name: httpd
        image: httpd:latest
        ports:
        - containerPort: 80
~
```

Step 3: After you write your deployment file, apply the deployment using the following command.

```
kubectl apply -f Deploy.yaml
```

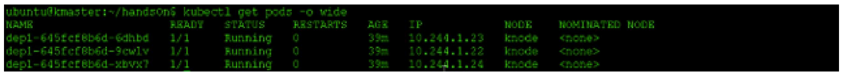


```
ubuntu@kmaster:~/handsOn$ kubectl apply -f Deploy.yaml
deployment.apps/dep1 configured
```

Here -f is a flag name used for the file name.

Step 4: Now, once the deployment is applied, get the list of pods running.

```
kubectl get pods -o wide
```



```
ubuntu@kmaster:~/handsOn$ kubectl get pods -o wide
NAME                    READY   STATUS    RESTARTS   AGE   IP            NODE    NOMINATED NODE
dep1-645fcf8b6d-6dhbd   1/1     Running   0          39m   10.244.1.23   knode   <none>
dep1-645fcf8b6d-9cwlv   1/1     Running   0          39m   10.244.1.22   knode   <none>
dep1-645fcf8b6d-xbvx7   1/1     Running   0          39m   10.244.1.24   knode   <none>
```

Here, -o wide are used to know on which node is the deployment running.

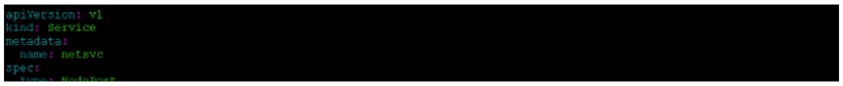
Step 5: After you have created a deployment, now you have to create a service. For that again use an editor and open a blank **service.yaml file**.

```
vi service.yaml
```

```
ubuntu@kmaster:~/handsOn$ vi service.yaml
```

Step 6: Once you open a service file, mention all the specifications for the service.

```
apiVersion: v1  #Defines the API Version
kind: Service   #Kinds parameter defines which kind of file is it, over here it is Service
metadata:
  name: netsvc   #Stores the name of the service
spec:            # Under Specifications, you mention all the specifications for the service
  type: NodePort
  selector:
    app: httpd
ports:
-protocol: TCP
 port: 80
 targetPort: 8084    #Target Port number is 8084
```



```
apiVersion: v1
kind: Service
metadata:
  name: netsvc
spec:
  type: NodePort
```

Subscribe to our Newsletter, and get personalized recommendations.

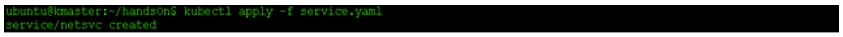
Sign up with Google

Signup with Facebook

Already have an account? Sign in.

See Batch Details

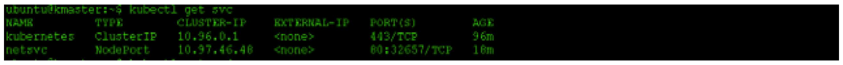
```
kubectl apply -f service.yaml
```



```
ubuntu@kmaster:~/handsOn$ kubectl apply -f service.yaml
service/netsvc created
```

Step 8: Now, once your service is applied to check whether the service is running or not use the following command.

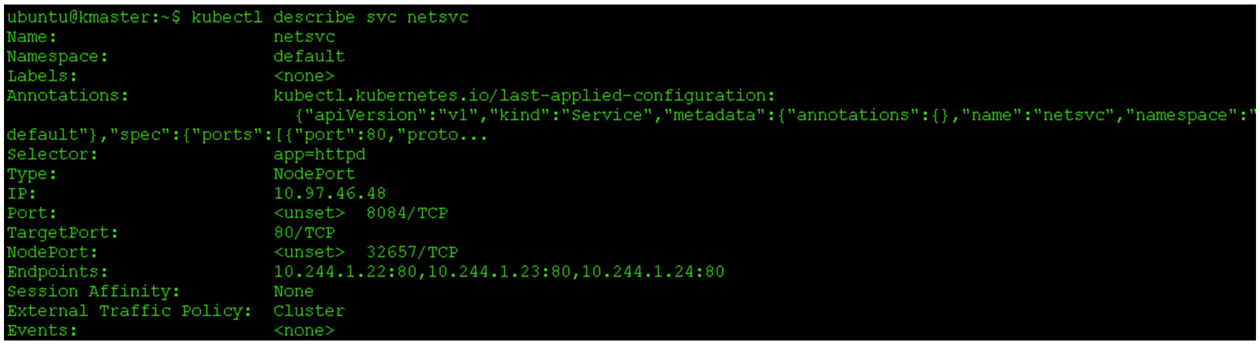
```
kubectl get svc
```



```
ubuntu@kmaster:~$ kubectl get svc
NAME         TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)          AGE
kubernetes   ClusterIP   10.96.0.1      <none>        443/TCP          96m
netsvc       NodePort    10.97.46.48    <none>        80:32657/TCP     18m
```

Step 9: Now, to see the specifications of service, and check which Endpoint it is binded to, use the following command.

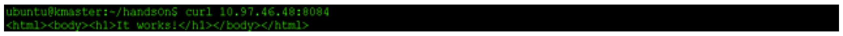
```
kubectl describe svc <name of the service>
```



```
ubuntu@kmaster:~$ kubectl describe svc netsvc
Name:                     netsvc
Namespace:                default
Labels:                   <none>
Annotations:              kubectl.kubernetes.io/last-applied-configuration:
                            {"apiVersion":"v1","kind":"Service","metadata":{"annotations":{},"name":"netsvc","namespace":"
default"},"spec":{"ports":[{"port":80,"proto...
Selector:                 app=httpd
Type:                     NodePort
IP:                       10.97.46.48
Port:                     <unset>  8084/TCP
TargetPort:               80/TCP
NodePort:                 <unset>  32657/TCP
Endpoints:                10.244.1.22:80,10.244.1.23:80,10.244.1.24:80
Session Affinity:         None
External Traffic Policy:  Cluster
Events:                   <none>
```

Step 10: Now since we are using amazon ec2 instance, to fetch the webpage and check the output, use the following command.

```
curl ip-address
```



```
ubuntu@kmaster:~/handsOn$ curl 10.97.46.48:8084
<html><body><h1>It works!</h1></body></html>
```

*If you found this Kubernetes Tutorial blog relevant, check out the **[Kubernetes Certification Training](#)** by Edureka, a trusted online learning company with a network of more than 250,000 satisfied learners spread across the globe.*

*Got a question for us? Please mention it in the comments section of "**Kubernetes Tutorial**" and I will get back to you.*