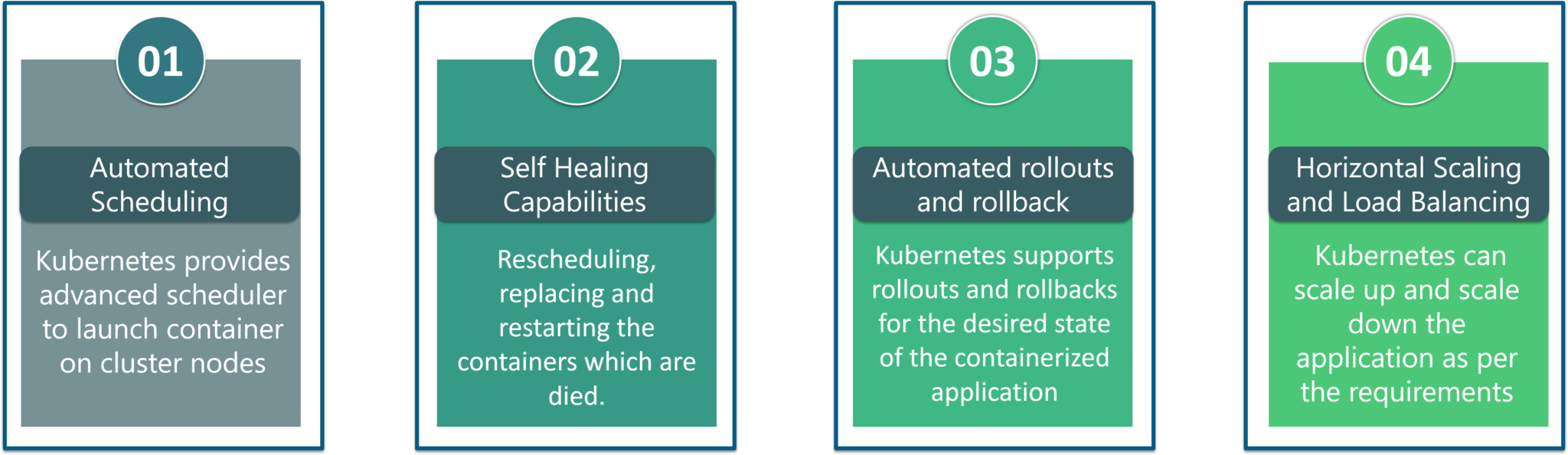
**Kubernetes: What is Kubernetes?**

[Kubernetes](https://www.edureka.co/blog/what-is-kubernetes-container-orchestration) 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: Kubernetes Features**

The features of Kubernetes, are as follows:



* **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: Kubernetes Architecture**

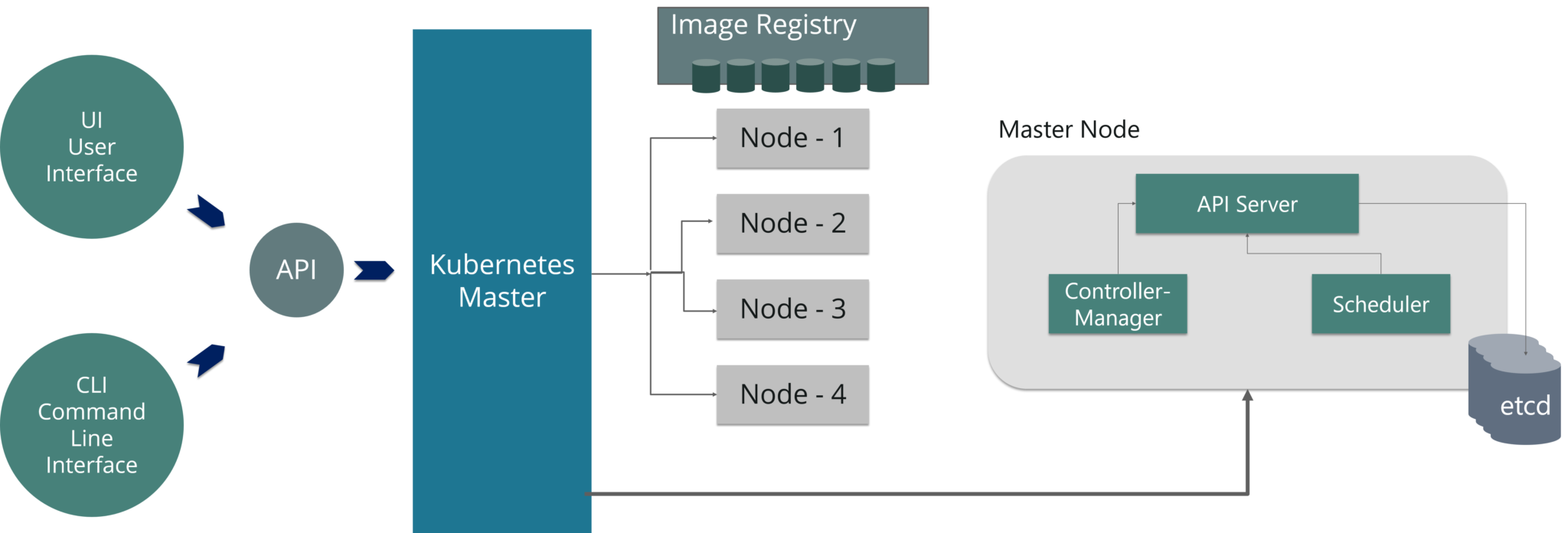
Kubernetes Architecture has the following main components:

* 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.

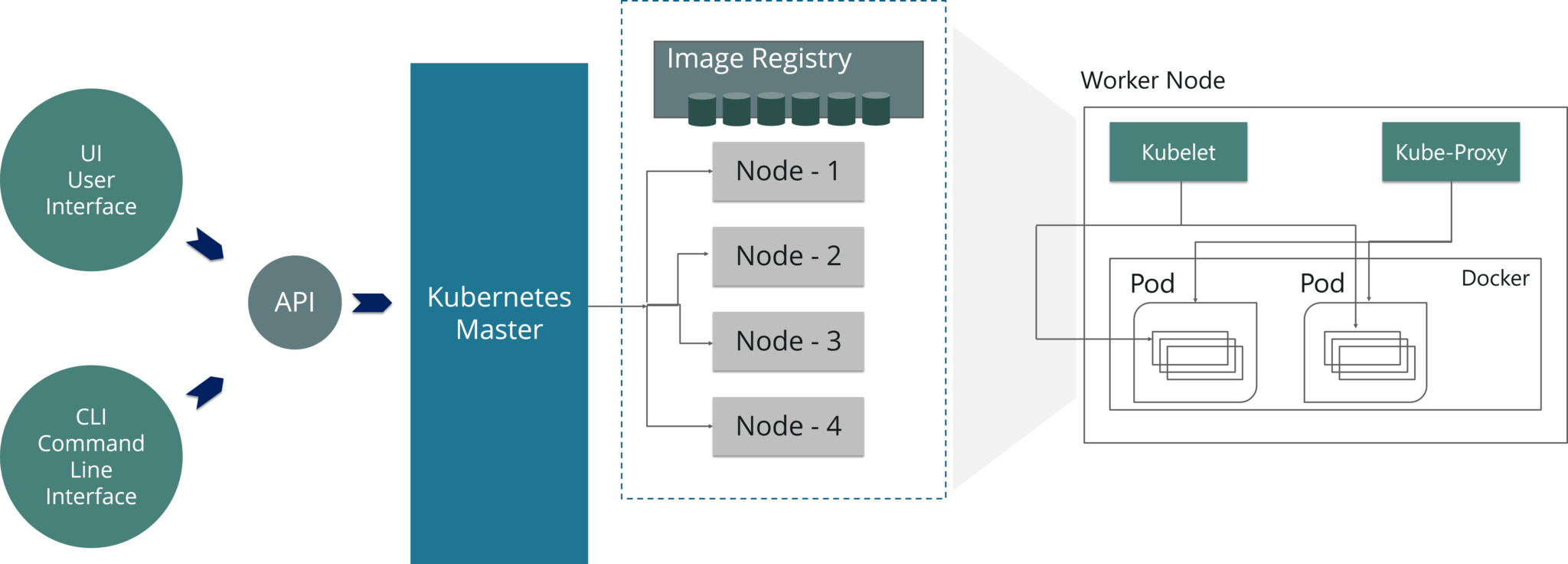


As you can see in the above diagram, the master node has various components like API Server, Controller Manager, Scheduler and ETCD.

* **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.](https://www.edureka.co/blog/kubernetes-architecture/)

**Kubernetes: 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.](https://www.edureka.co/blog/amazon-eks/)

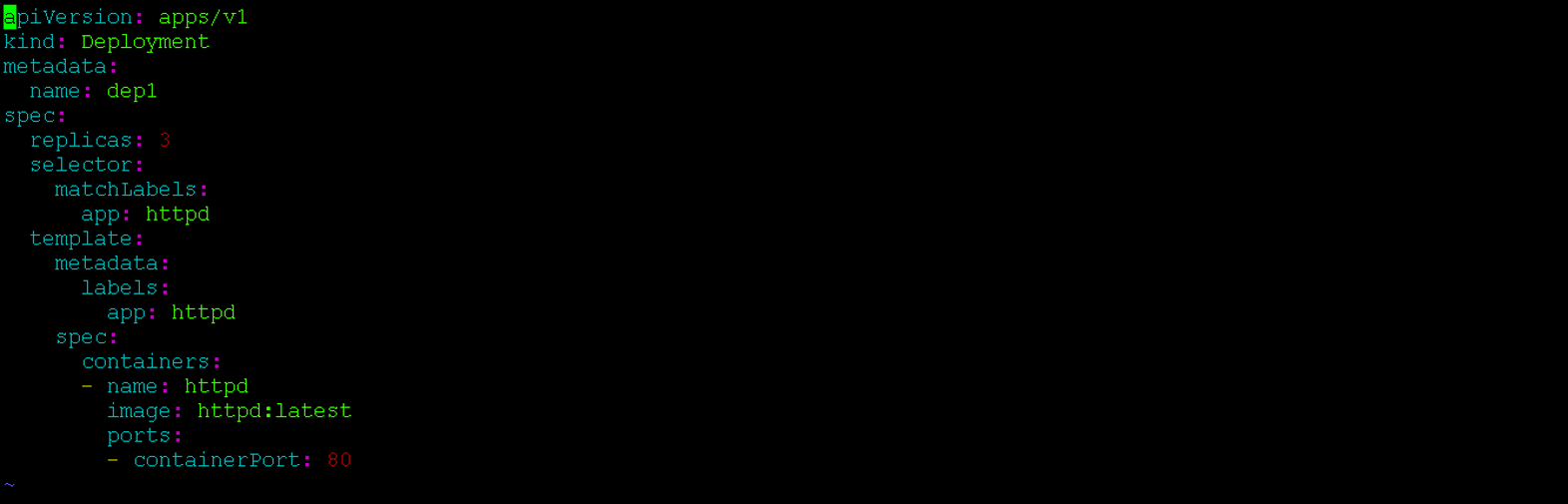
**Step 1:**First **create a folder** inside which you will create your deployment and service. After that, use an editor and **open a Deployment file**.

|  |  |
| --- | --- |
| 1  2  3 | mkdir handsOn  cd handsOn  vi Deploy.yaml |

**Snapshot of Demo - Kubernetes Tutorial - Edureka**

**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.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | apiVersion: apps/v1  #Defines the API Version  kind: Deployment     #Kinds parameter defines which kind of file is it, over here it is Deployment  metadata:    name: dep1        #Stores the name of the deployment  spec:               # Under Specifications, you mention all the specifications for the deployment    replicas: 3       # Number of replicas would be 3    selector:     matchLabels:       app: httpd     #Label name which would be searched is httpd    template:      metadata:      labels:        app: httpd   #Template name would be httpd    spec:            # Under Specifications, you mention all the specifications for the containers     containers:     - name: httpd   #Name of the containers would be httpd       image: httpd:latest  #The image which has to be downloaded is httpd:latest       ports:       - containerPort: 80 #The application would be exposed on port 80 |

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

|  |  |
| --- | --- |
| 1 | kubectl apply -f Deploy.yaml |

**Snapshot of Demo - Kubernetes Tutorial - Edureka**

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.

|  |  |
| --- | --- |
| 1 | kubectl get pods -o wide |

**Snapshot of Demo - Kubernetes Tutorial - Edureka**

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

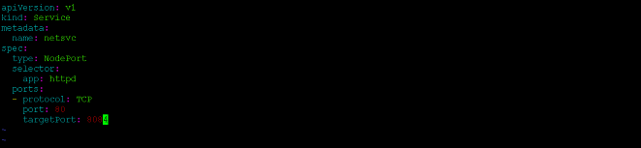
**Step 5:**After you have created a deployment, now you have to create a service. For that again use an editor and open a blank **service.yaml file**.

|  |  |
| --- | --- |
| 1 | vi service.yaml |

**Snapshot of Demo - Kubernetes Tutorial - Edureka**

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

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | 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 |

****

**Step 7:** After you write your service file, apply the service file using the following command.

**[Certified Kubernetes Administrator Exam Trainin](https://www.edureka.co/kubernetes-certification" \t "_blank)**

|  |  |
| --- | --- |
| 1 | kubectl apply -f service.yaml |

**https://d1jnx9ba8s6j9r.cloudfront.net/blog/wp-content/uploads/2018/11/d7-1.png**

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

|  |  |
| --- | --- |
| 1 | kubectl get svc |

**Snapshot of Demo - Kubernetes Tutorial - Edureka**

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

|  |  |
| --- | --- |
| 1 | kubectl describe svc <name of the service> |

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

|  |  |
| --- | --- |
| 1 | curl ip-address |

*Snapshot of Demo - Kubernetes Tutorial - Edureka*