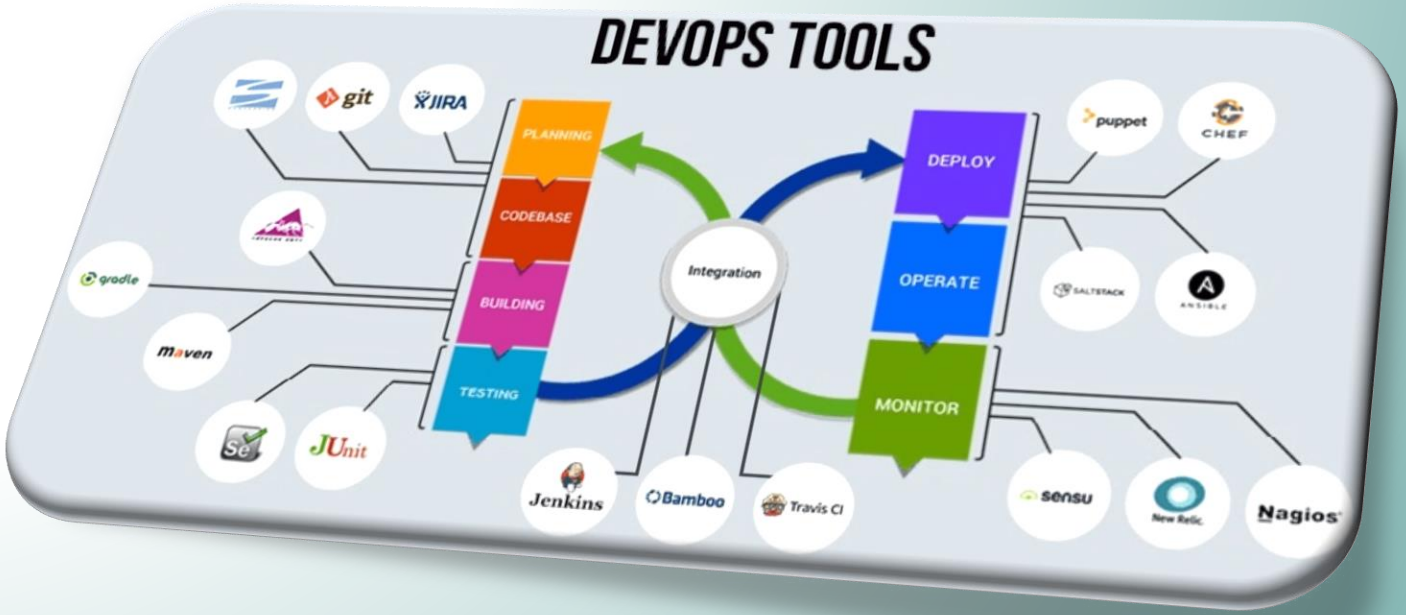
Container Orchestration [Kubernetes]



AGENDA

- Introduction to Kubernetes
- Kubernetes Architecture
- Kubernetes Installation
- Creating a Deployment in Kubernetes Using YAML
- Services in Kubernetes
- Ingress in Kubernetes

INTRODUCTION TO KUBERNETES

INTRODUCTION TO KUBERNETES

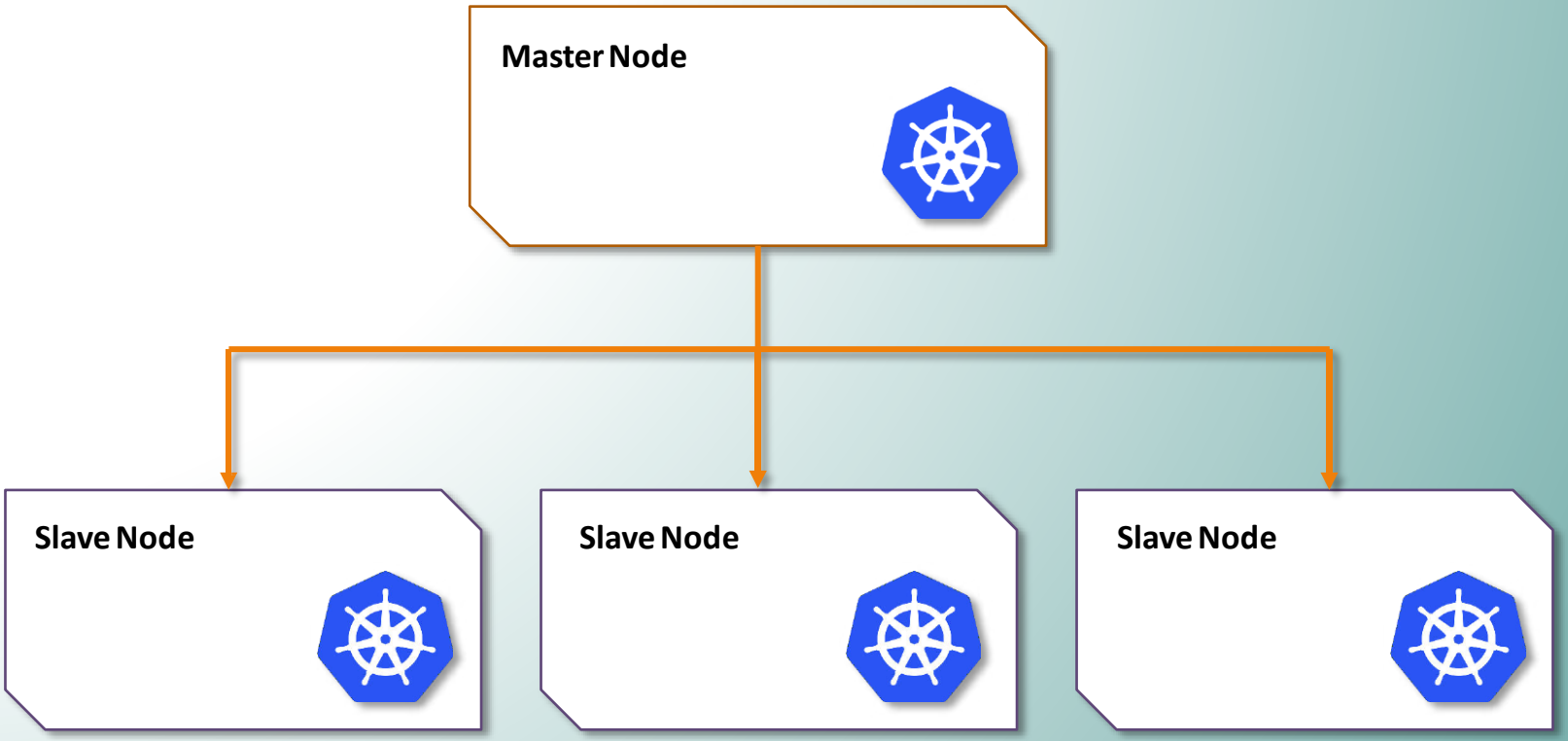
- Kubernetes is an open-source container orchestration software.
- It was originally developed by Google.
- It was first released on July 21, 2015.
- It is the ninth most active repository on GitHub in terms of number of commits.

FEATURES OF KUBERNETES

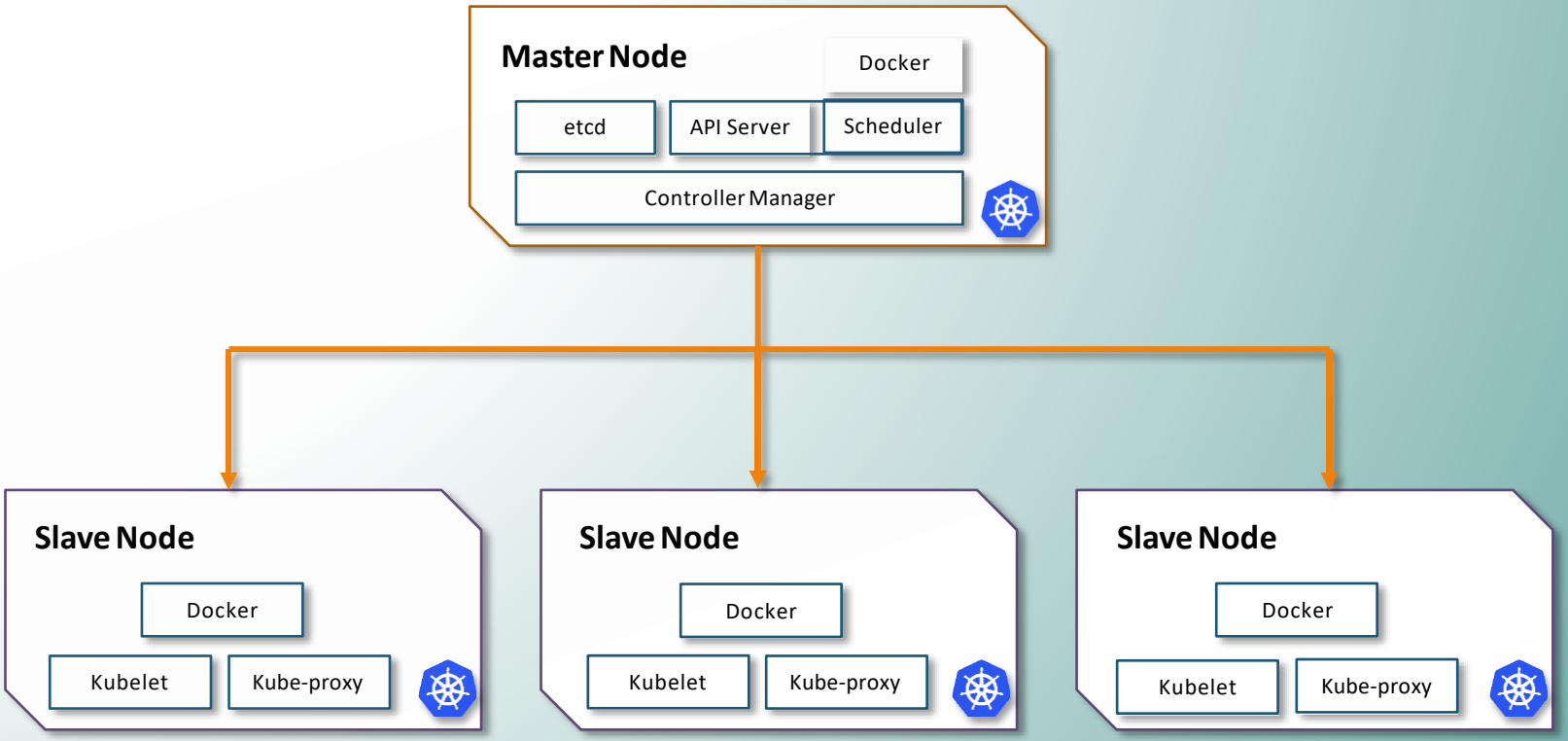
- Pods
- Replication Controller
- Storage Management
- Resource Monitoring
- Health Checks
- Service Discovery
- Networking
- Secret Management
- Rolling Updates

KUBERNETES ARCHITECTURE

KUBERNETES ARCHITECTURE



KUBERNETES ARCHITECTURE



KUBERNETES ARCHITECTURE: MASTER COMPONENTS

KUBERNETES ARCHITECTURE: MASTER COMPONENTS

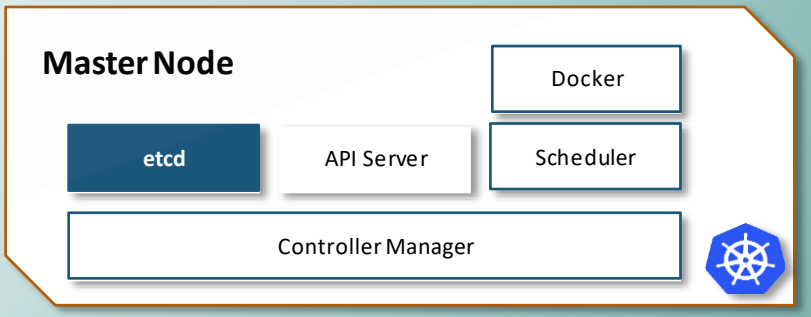
etcd

API Server

Scheduler

Controller Manager

It is a highly available distributed key–value store, which is used to store cluster wide secrets. It is only accessible by the Kubernetes API server, as it has sensitive information.



KUBERNETES ARCHITECTURE: MASTER COMPONENTS

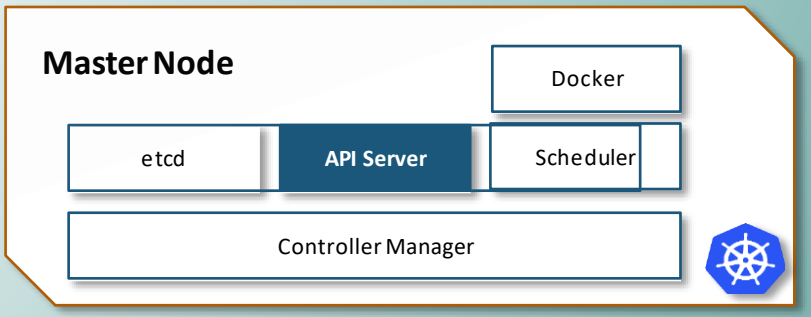
etcd

API Server

Scheduler

Controller Manager

It exposes Kubernetes API. Kubernetes API is the front-end for the Kubernetes Control Plane and is used to deploy and execute all operations in Kubernetes.



KUBERNETES ARCHITECTURE: MASTER COMPONENTS

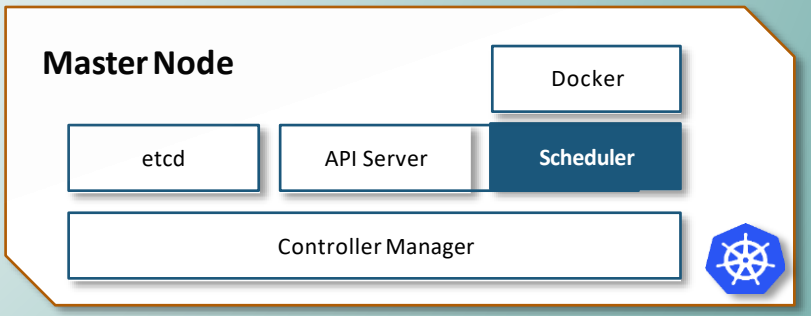
etcd

API Server

Scheduler

Controller Manager

The scheduler takes care of scheduling of all processes and the dynamic resource management and manages present and future events on the cluster.



KUBERNETES ARCHITECTURE: MASTER COMPONENTS

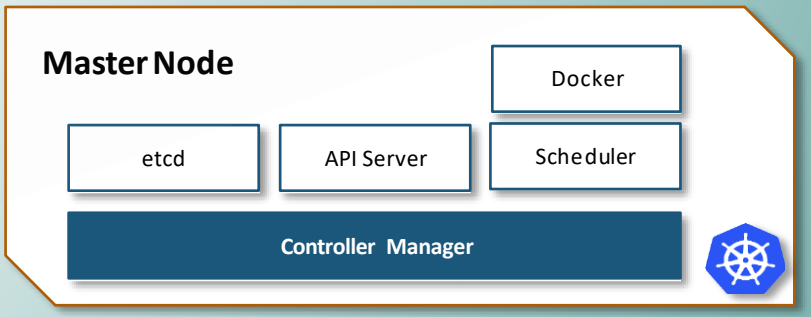
etcd

API Server

Scheduler

Controller Manager

The controller manager runs all controllers on the Kubernetes cluster. Although each controller is a separate process, to reduce complexity, all controllers are compiled into a single process. They are as follows: **Node Controller, Replication Controller, Endpoints Controller, Service Accounts and TokenControllers**.



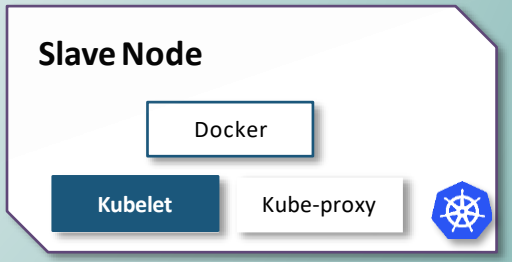
KUBERNETES ARCHITECTURE: SLAVE COMPONENTS

KUBERNETES ARCHITECTURE: SLAVE COMPONENTS

Kubelet

Kube-proxy

Kubelet takes the specification from the API server and ensures that the application is running according to the specifications which were mentioned. Each node has its own kubelet service.

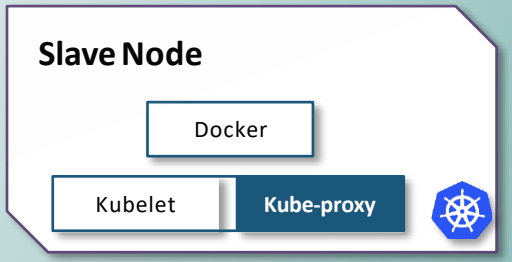


KUBERNETES ARCHITECTURE: SLAVE COMPONENTS

Kubelet

Kube-proxy

This proxy service runs on each node and helps in making services available to the external host. It helps in connection forwarding to the correct resources. It is also capable of doing primitive load balancing.

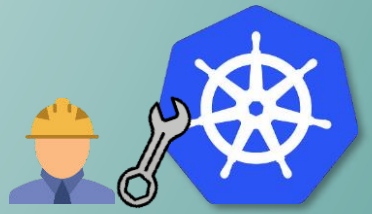


KUBERNETES INSTALLATION

KUBERNETES INSTALLATION

There are numerous ways to install Kubernetes. Following are some of the popular ways:

- **Kubeadm**: Bare Metal Installation
- **Minikube**: Virtualized Environment for Kubernetes
- **Kops**: Kubernetes on AWS
- **Kubernetes on GCP**: Kubernetes running on Google Cloud Platform



HANDS-ON: INSTALLING KUBERNETES USING KUBEADM

WORKING OF KUBERNETES

WORKING OF KUBERNETES

Pods can have one or more containers coupled together. They are the basic unit of Kubernetes. To increase high availability, we always prefer pods to be in replicas.

WORKING OF KUBERNETES

Services are used to load balance the traffic among the pods. It follows round-robin distribution among the healthy pods.

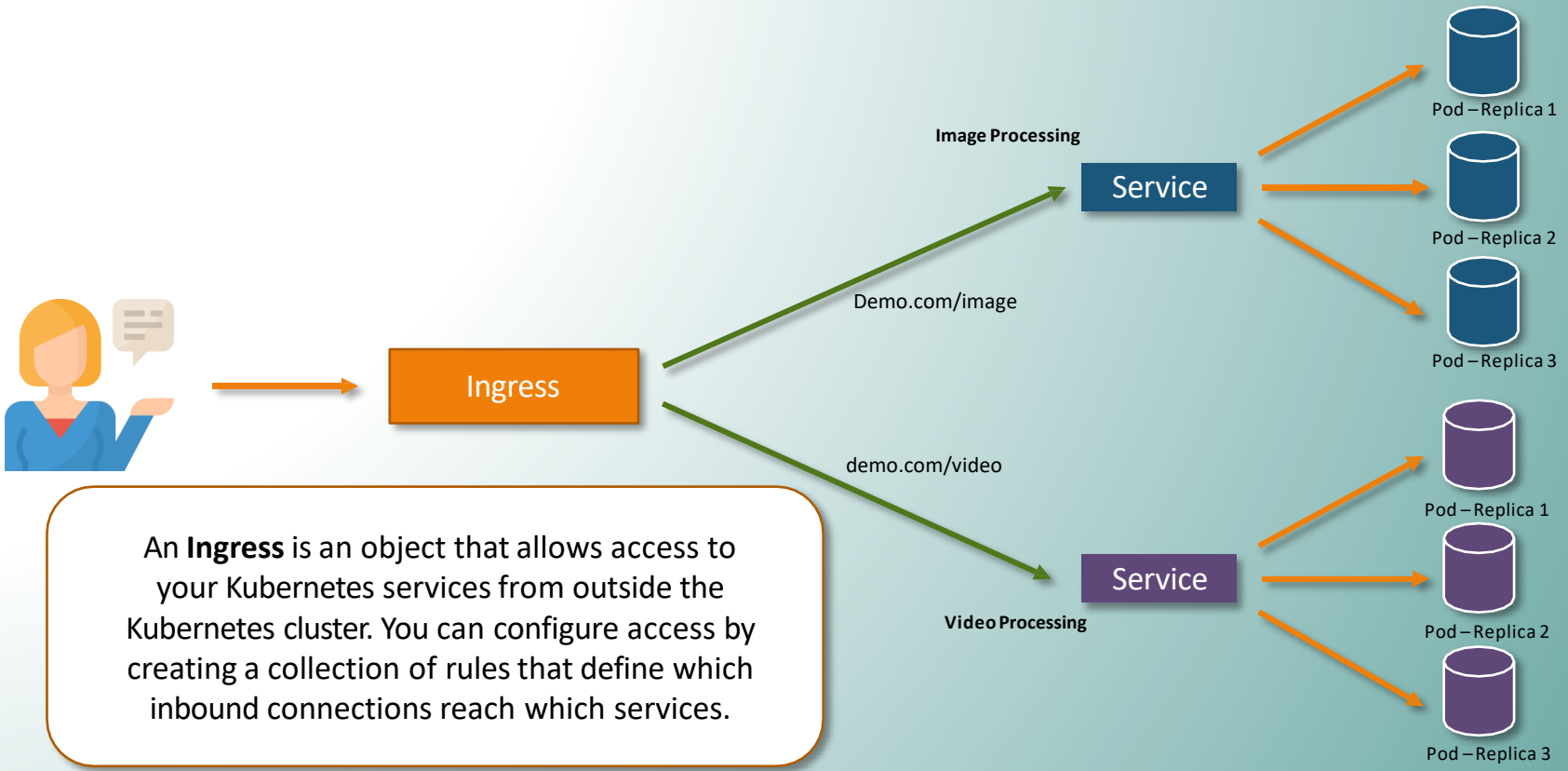
Service

Pod – Replica 1

Pod – Replica 2

Pod – Replica 3

WORKING OF KUBERNETES

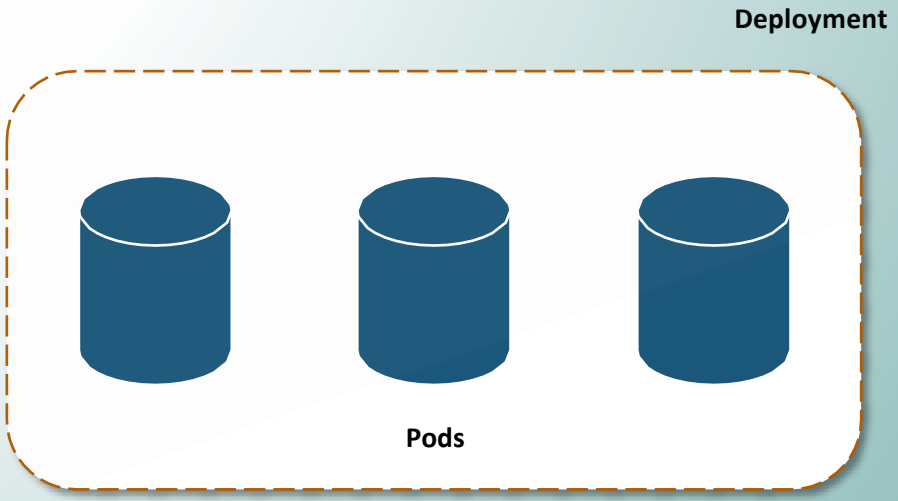


An **Ingress** is an object that allows access to your Kubernetes services from outside the Kubernetes cluster. You can configure access by creating a collection of rules that define which inbound connections reach which services.

DEPLOYMENTS IN KUBERNETES

DEPLOYMENTS IN KUBERNETES

Deployment in Kubernetes is a controller which helps your applications reach the desired state; the desired state is defined inside the deployment file.



YAML SYNTAX FOR DEPLOYMENTS

This YAML file will deploy 3 pods for nginx and will maintain the desired state, which is 3 pods, until this deployment is deleted.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
  matchLabels:
      app: nginx
  template:
  metadata:
      labels:
        app:nginx
    spec:
      containers:
      - name: nginx
        image:nginx:1.7.9
        ports:
        - containerPort:80
```

CREATING A DEPLOYMENT

Once the file is created, to deploy this deployment use the following syntax:

Syntax

```
kubectl create –f nginx.yaml
```

ubuntu@ip-172-31-39-244: ~

```
ubuntu@ip-172-31-39-244:~$ kubectl create -f nginx.yaml
deployment.apps/nginx-deployment created
ubuntu@ip-172-31-39-244:~$
```

LISTING THE PODS

To view the pods, type the following command:

Syntax

```
kubectl get po
```

ubuntu@ip-172-31-39-244: ~

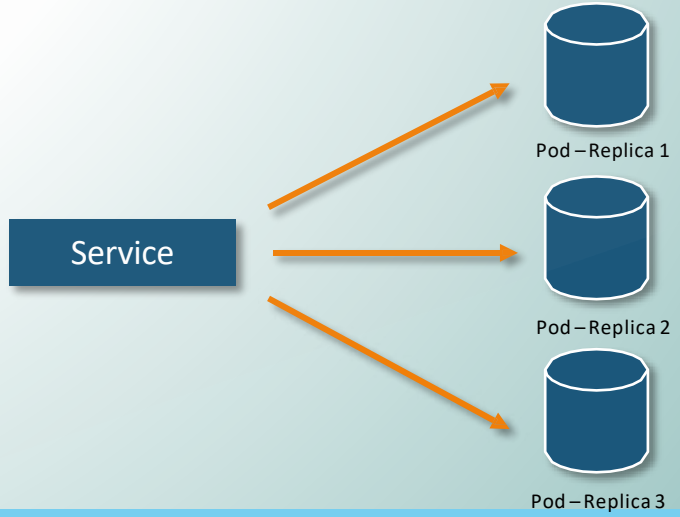
```
ubuntu@ip-172-31-39-244:~$ kubectl get po
NAME                                READY   STATUS    RESTARTS   AGE
nginx-deployment-76bf4969df-24vpl   1/1     Running   0          4m38s
nginx-deployment-76bf4969df-frz7j   1/1     Running   0          4m38s
nginx-deployment-76bf4969df-grnmc   1/1     Running   0          4m38s
ubuntu@ip-172-31-39-244:~$
```

As you can see, the number of pods are matching with the number of replicas specified in the deployment file.

CREATING A SERVICE

CREATING A SERVICE

A Service is basically a round-robin load balancer for all pods, which matches with its name or selector. It constantly monitors the pods; in case a pod gets unhealthy, the service will start deploying the traffic to other healthy pods.



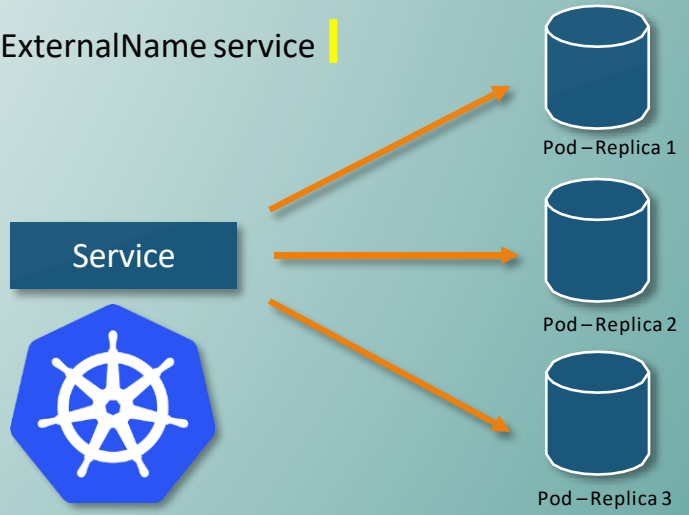
SERVICE TYPES

ClusterIP: Exposes the service on cluster-internal IP

NodePort: Exposes the service on each Node's IP at a static port

LoadBalancer: Exposes the service externally using a cloud provider's load balancer

ExternalName: Maps the service to the DNS Name mentioned with the ExternalName service



CREATING A NODEPORT SERVICE

We can create a NodePort service using the following syntax:

Syntax

```
kubectl create service nodeport <name-of-service> --tcp=<port-of-service>:<port-of-container>
```

ubuntu@ip-172-31-39-244: ~

```
ubuntu@ip-172-31-39-244:~$ kubectl create service nodeport nginx --tcp=80:80
service/nginx created
ubuntu@ip-172-31-39-244:~$
```

CREATING A NODEPORT SERVICE

To know the port, on which the service is being exposed, type the following command:

Syntax

```
kubectl get svc nginx
```

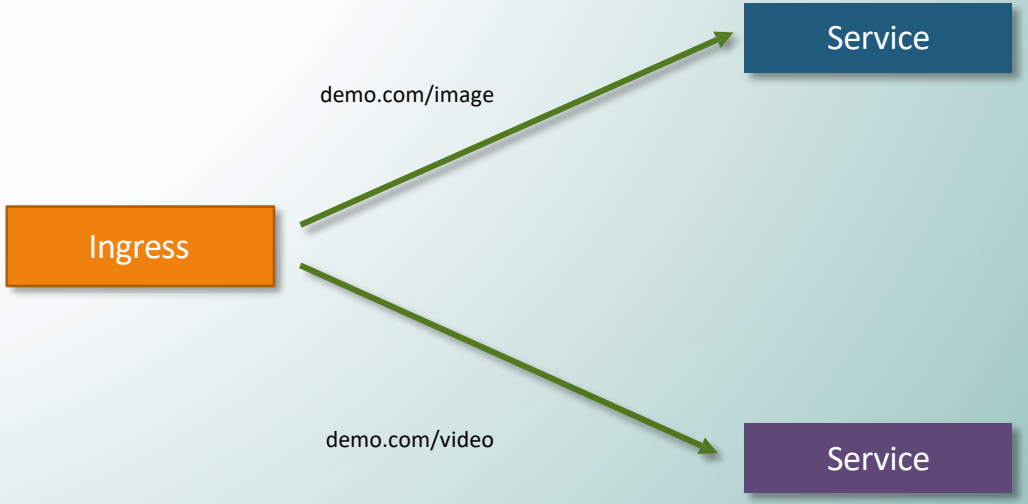
ubuntu@ip-172-31-39-244: ~

```
ubuntu@ip-172-31-39-244:~$ kubectl get svc nginx
NAME     TYPE       CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
nginx    NodePort   10.103.235.81   <none>        80:32043/TCP   114s
ubuntu@ip-172-31-39-244:~$
```

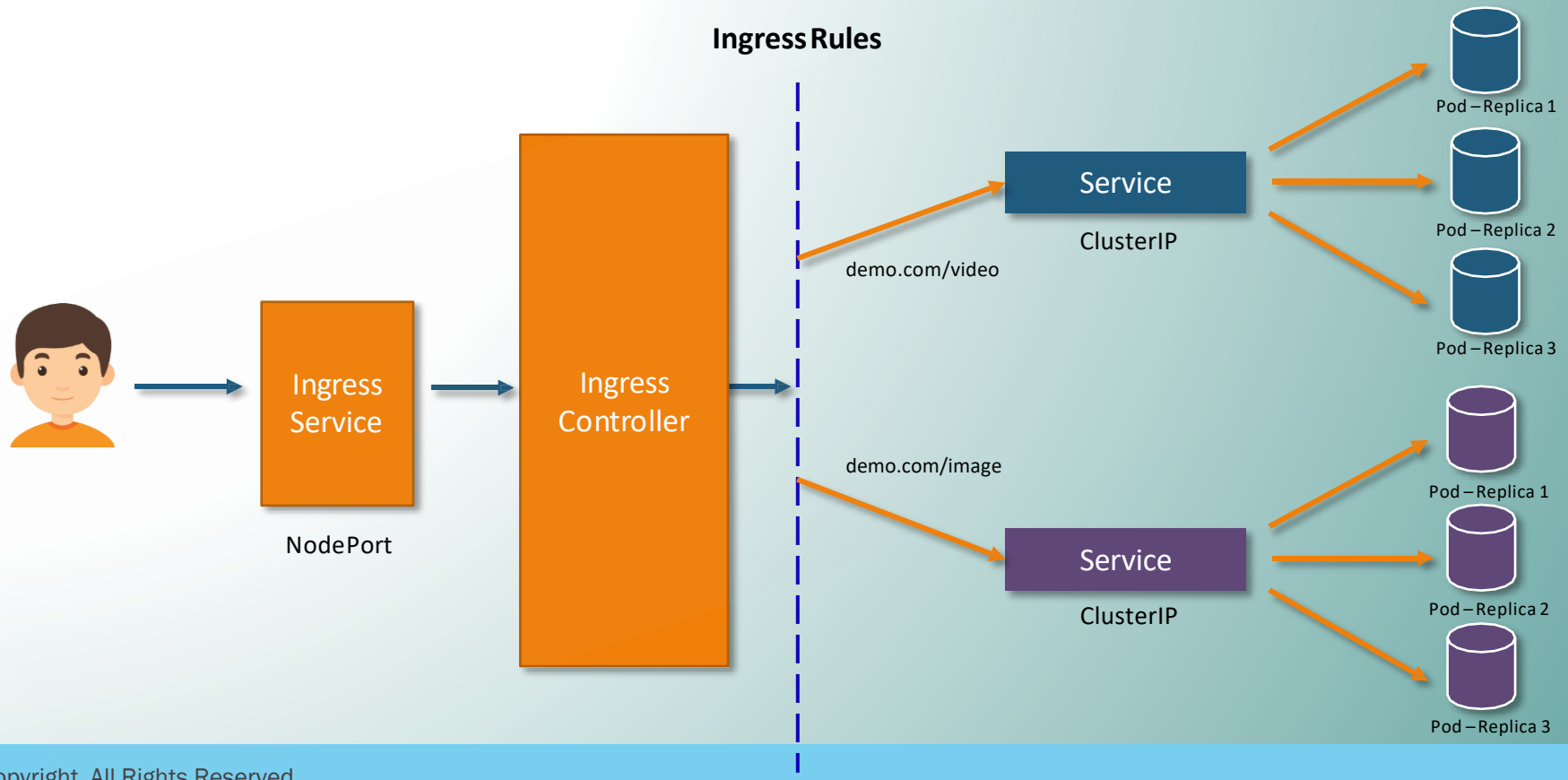
CREATING AN INGRESS

WHAT IS AN INGRESS?

Kubernetes ingress is a collection of routing rules that govern how external users access services running in a Kubernetes cluster.



WHAT IS AN INGRESS?



INSTALLING INGRESS CONTROLLER

We will be using the nginx ingress controller for our demo. We can download it from the following link:

Link

https://github.com/kubernetes/ingress-nginx/blob/master/docs/deploy/index.md

DEFINING INGRESS RULES

The following rule, will redirect traffic which asks for /foo to nginx service. All other requests will be redirected to ingress controller's default page.

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: simple-fanout-example
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
  - http:
      paths:
      - path: /foo
        backend:
          serviceName: nginx
          servicePort: 80
```

DEPLOYING INGRESS RULES

To deploy ingress rules, we use the following syntax:

Syntax

```
kubectl create –f ingress.yaml
```

ubuntu@ip-172-31-17-194: ~

```
ubuntu@ip-172-31-17-194:~$ kubectl create -f ingress.yaml
ingress.extensions/simple-fanout-example created
ubuntu@ip-172-31-17-194:~$
```

VIEWING INGRESS RULES

To list the ingress rules we use the following syntax:

Syntax

```
kubectl get ing
```

ubuntu@ip-172-31-17-194: ~

```
ubuntu@ip-172-31-17-194:~$ kubectl get ing
NAME                    HOSTS   ADDRESS   PORTS   AGE
simple-fanout-example   *                 80      2m5s
ubuntu@ip-172-31-17-194:~$
```

Thank you