Kubernetes

Orchestrating Containers with Ease

By Santhosh Kiran Jaligama

What is Kubernetes?

- **Definition:** Kubernetes is an open-source platform designed to automate deploying, scaling, and operating containerized applications.
- **History:** Developed by Google, now maintained by the Cloud Native Computing Foundation (CNCF).
- **Logo and Pronunciation:** "K8s" is a common abbreviation.

Key Features of Kubernetes

- Automatic Bin Packing
- Self-Healing
- Horizontal Scaling
- Service Discovery and Load Balancing
- Automated Rollouts and Rollbacks
- Secret and Configuration Management

Key Features of Kubernetes in Detail

Automatic Bin Packing

- **Description:** Kubernetes optimizes the placement of containers based on resource requirements and constraints. It schedules containers on nodes with sufficient resources, maximizing efficiency and utilization.
- **Example:** If a node has available CPU and memory resources, Kubernetes will place a new container there to balance the load across the cluster.

Self-Healing

- **Description:** Kubernetes automatically restarts containers that fail, replaces containers, kills containers that don’t respond to user-defined health checks, and doesn’t advertise them to clients until they’re ready to serve.
- **Example:** If a container running a web server crashes, Kubernetes will detect the failure and restart the container to ensure continuous availability.

Key Features of Kubernetes in Detail

Horizontal Scaling

- **Description:** Kubernetes allows you to scale applications up and down by adding or removing container instances (pods) in response to demand, either manually or automatically based on resource utilization.
- **Example:** During a peak traffic period, you can scale out a web application to handle more requests by increasing the number of replicas.

Service Discovery and Load Balancing

- **Description:** Kubernetes automatically assigns IP addresses to containers and a single DNS name for a set of containers. It can also load balance traffic across multiple containers.
- **Example:** Kubernetes services ensure that incoming requests are distributed evenly across all available instances of a web application.

Key Features of Kubernetes in Detail

Automated Rollouts and Rollbacks

- **Description:** Kubernetes automates the deployment of new versions of applications, as well as rolling back to previous versions if there are issues.
- **Example:** When updating an application, Kubernetes can gradually roll out the changes to minimize disruption and can quickly rollback if problems are detected.

Secret and Configuration Management

- **Description:** Kubernetes provides mechanisms to manage sensitive information, such as passwords, OAuth tokens, and SSH keys, as well as application configuration data.
- **Example:** Secrets can be mounted as files in the pod's filesystem or accessed as environment variables, ensuring sensitive data is securely handled.

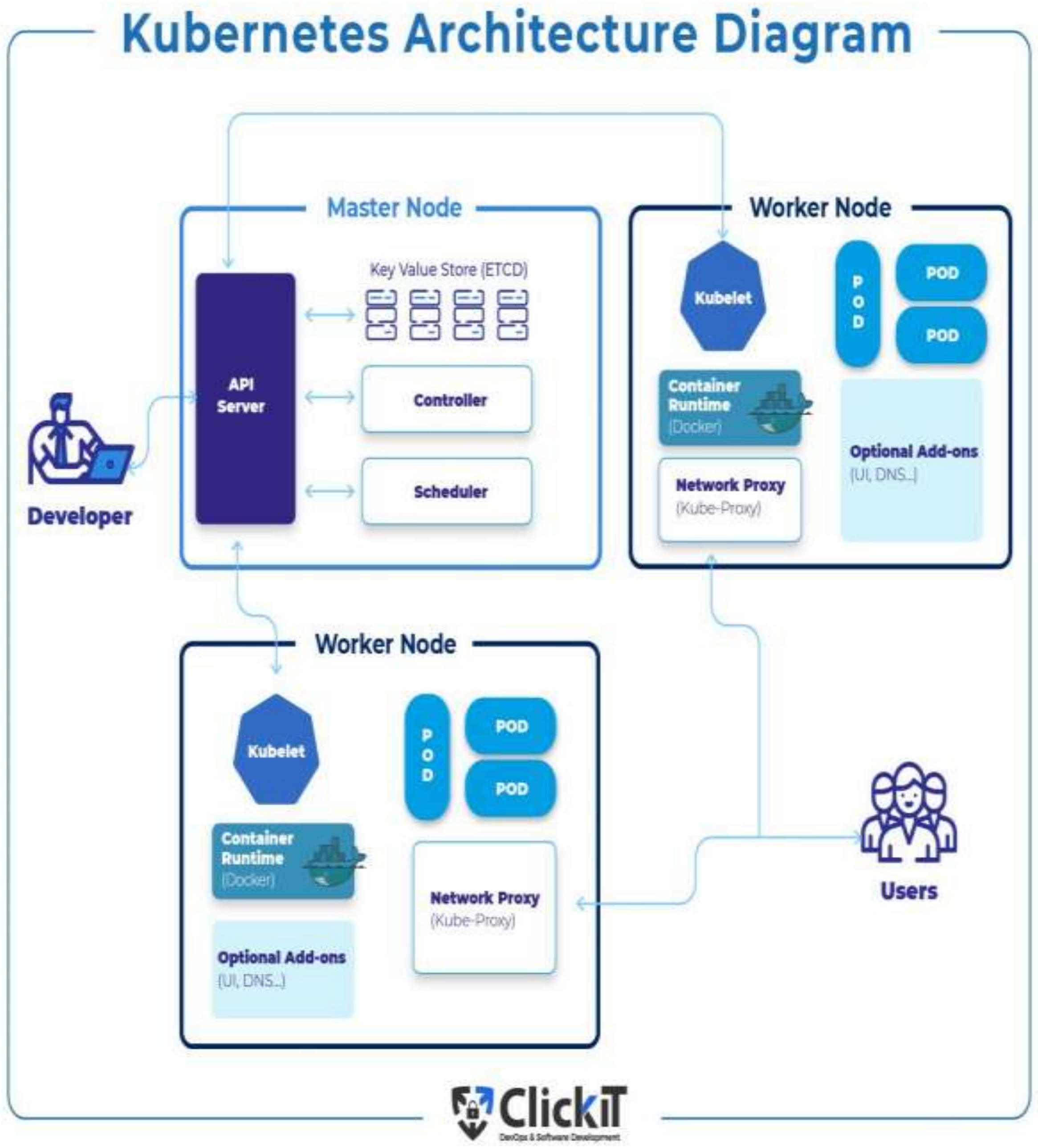
Kubernetes Architecture

Master Node:

- **API Server:** The front-end for the Kubernetes control plane and is used to deploy and execute all operations in Kubernetes.
- **Scheduler:** Assigns work to nodes based on resource availability and predefined policies.
- **Controller Manager:** Manages the state of the cluster, ensuring the desired number of pods are running.
- **etcd:** Distributed key-value store for all cluster data, only accessible by API Server. (secret keys, config Maps)

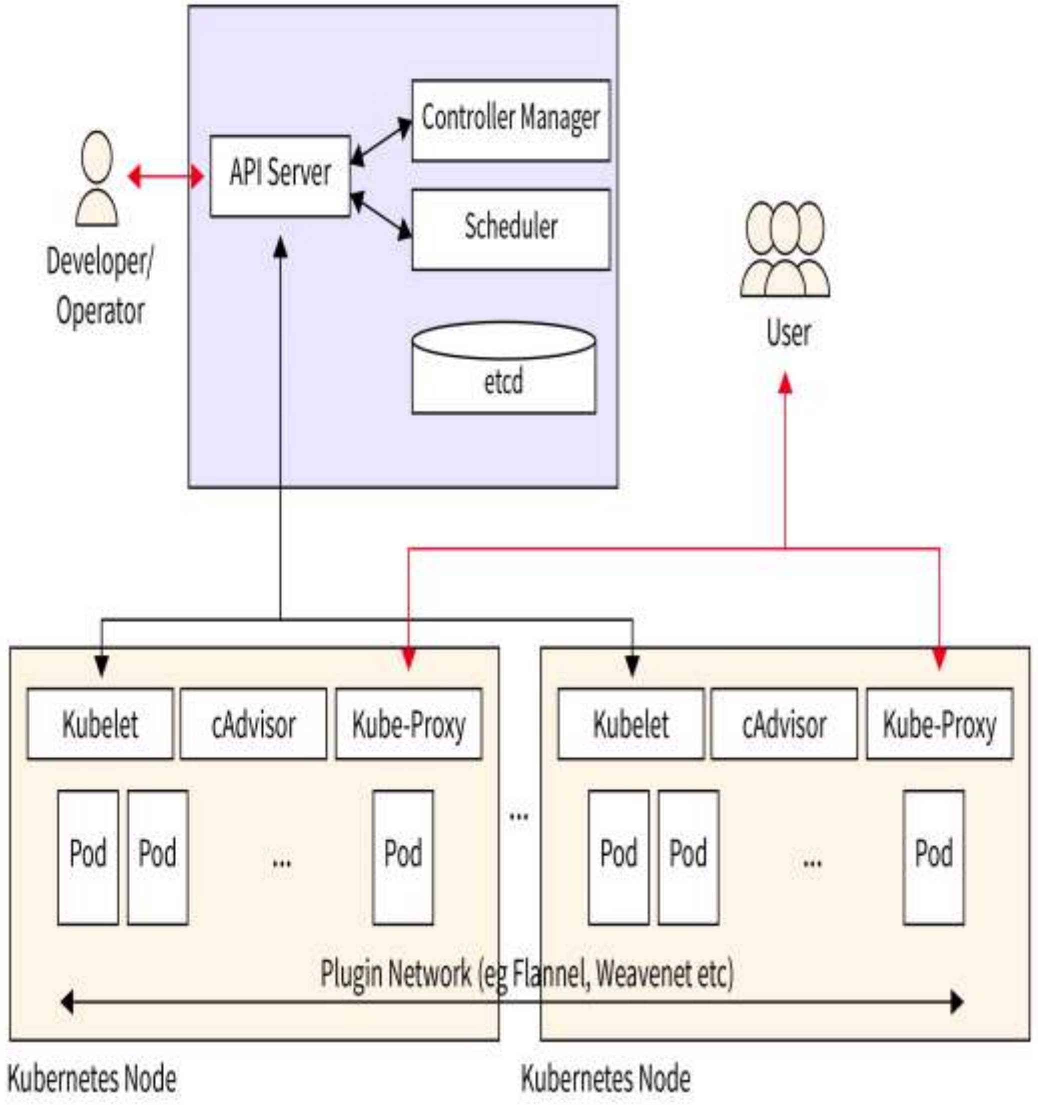
Worker Nodes:

- **Kubelet:** An agent that ensures containers are running in a pod.
- **Kube-proxy:** Manages network connectivity and load balancing.
- **Container Runtime:** Software responsible for running containers (e.g., Docker, containerd).


Kubernetes Architecture Diagram
Developer
Master Node
API Server
Key Value Store (ETCD)
Controller
Scheduler
Worker Node
Kubelet
POD
POD
POD
Container Runtime (Docker)
Network Proxy (Kube-Proxy)
Optional Add-ons (UI, DNS...)
Worker Node
Kubelet
POD
POD
POD
Container Runtime (Docker)
Optional Add-ons (UI, DNS...)
Network Proxy (Kube-Proxy)
Users
ClickIT
DevOps & Software Development

Workflow of Kubernetes

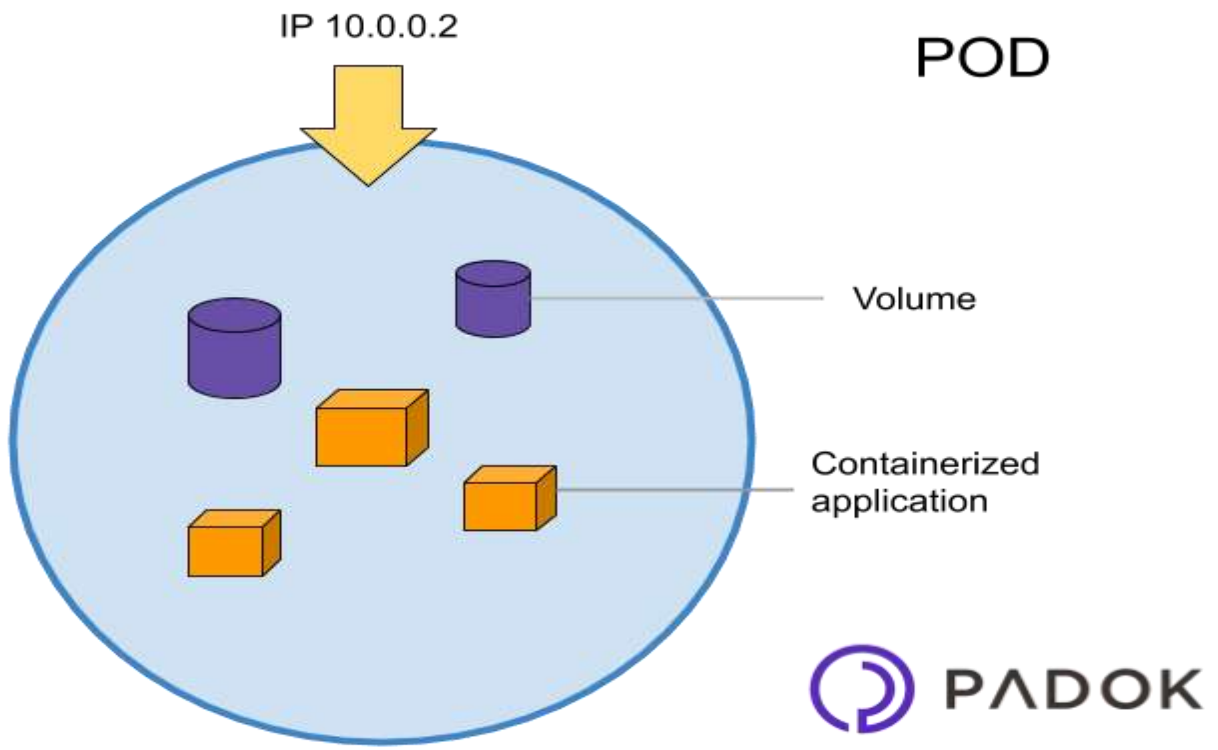
- **API Request Initiation:** User or external system initiates an API request.
- **API Server Validation:** API server validates the request and permissions.
- **Desired State Retrieval:** API server retrieves the current desired state from etcd.
- **Processing by Controllers:** Controllers in the control manager process the request.
- **Scheduling Decision:** Scheduler determines the best worker node for the pod.
- **Pod Definition Creation:** API server creates a Pod definition based on the request and scheduler's decision.
- **Pod Assignment to Nodes:** Pod definition is sent to the appropriate Kubelet on the chosen worker node.
- **Container Runtime Invocation:** Kubelet communicates with the container runtime to start the containers.
- **Networking and Load Balancing:** Kube-proxy sets up networking and load balancing for the Pod.
- **Reporting to Control Plane:** Kubelet reports the status of containers and pods back to the control plane.
- **Eventual Consistency:** The Control plane continuously monitors the cluster state to maintain the desired state over time.



Kubernetes Objects

Pod: A pod is the basic object of Kubernetes. It is in charge of encapsulating containers, storage resources, and internal IPs. One pod represents one instance of an application in Kubernetes. These pods are launched in a Kubernetes cluster which is composed of nodes. A node is a worker machine (VM or physical machine) that has a container runtime environment. This means that a pod runs on a node but can easily be instantiated on another node (for fault tolerance reason per example).

Pods are meant to be ephemeral in fact **Kubernetes can scale the number of these pods** to adapt for incoming traffic, consequently creating or deleting pods on demand.

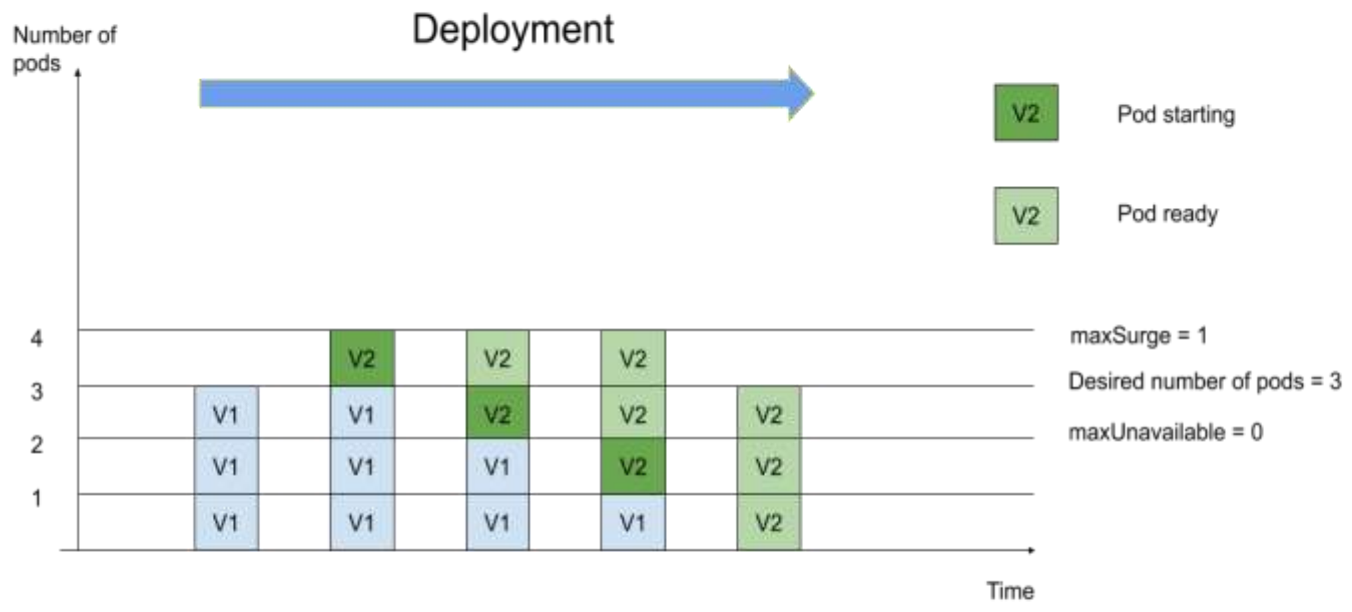


Kubernetes Objects

Deployments: Deployments are Kubernetes objects that manage pods. They allow you to deploy a specific version of your application and specify the number of pods you require for it to be operational.

When a new version is ready to go live in production, **deployment can easily manage this upgrade with no downtime** by applying two basic rules:

- maxSurge specifies the maximum number of pods that can be created beyond the desired pod count.
- Max Unavailable specifies the maximum number of pods that can be unavailable during the deployment.



Kubernetes Objects

Configmap is a Kubernetes object that **maintains a key value store** that can easily be used by other Kubernetes objects such as pod, deployments, and services.
Thus you can define a configmap composed of all the specific environment variables.

```
ENVIRONMENT=STAGING
GIT_BRANCH=DEV
DATABASE_PASSWORD=xxxx
```

Secrets allows us to **store sensitive information safely within kubernetes** thanks to **role based access** and a specific way of sharing the secret with the pod that does not write it to the disk. You can encrypt all these secrets at rest with EncryptionConfiguration.

Cronjobs are used to **execute a task periodically**, which in Kubernetes means **launching an idempotent pod at a specific time of the day and then delete it** when the job is finished. Hence not having pod continuously consume cpu resources for no reason.

```
schedule: "*/1 * * * *“   # Min Hour Dayofmonth Month DayofWeek
```

Kubernetes Objects

Volume:

- **Description:** A directory accessible to containers in a pod, providing persistent storage.
- **Example:** Using a persistent volume to store data for a database container.

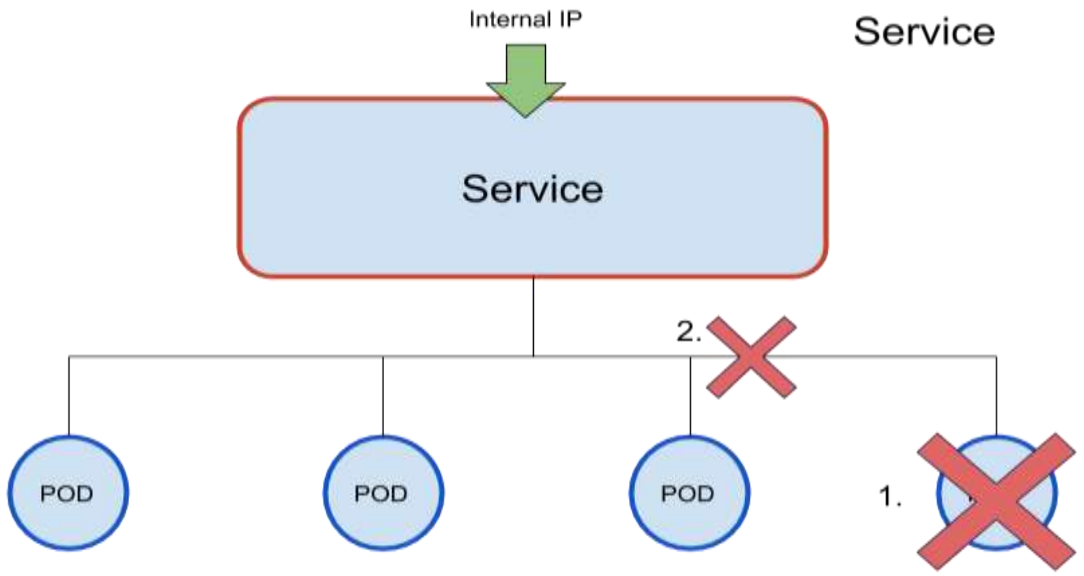
Namespace:

- **Description:** A way to divide cluster resources between multiple users or teams. It is a logical isolation of resources, network policies, RBAC and everything.
- **Example:** Separate namespaces for development, testing, and production environments to isolate resources.

Kubernetes Objects

Services/Ingress: The mortal nature of pods makes them unreliable for traffic delivery. Kubernetes' solution is **Service**. It is an abstraction that **maintains a logical set of pods that accept incoming traffic** and expose a service port to access the underlying pods. Kubernetes service can hence manage changes in traffic by modifying the number of pods.

If your application needs to be accessed from outside your cluster, Kubernetes offers a component called **Ingress**. It sits on top of a service and **manages external traffic with SSL encryption**.



PADOK

Kubernetes Objects – Service Types

Cluster IP: Exposes the service on cluster-internal IP
NodePort: Exposes the service on each node’s IP at a static port
Load Balancer: Exposes the service externally using a cloud provider’s load balancer
External Name: Maps the service to the DNS Name mentioned with the ExternalName service

https://www.harness.io/blog/kubernetes-services-explained

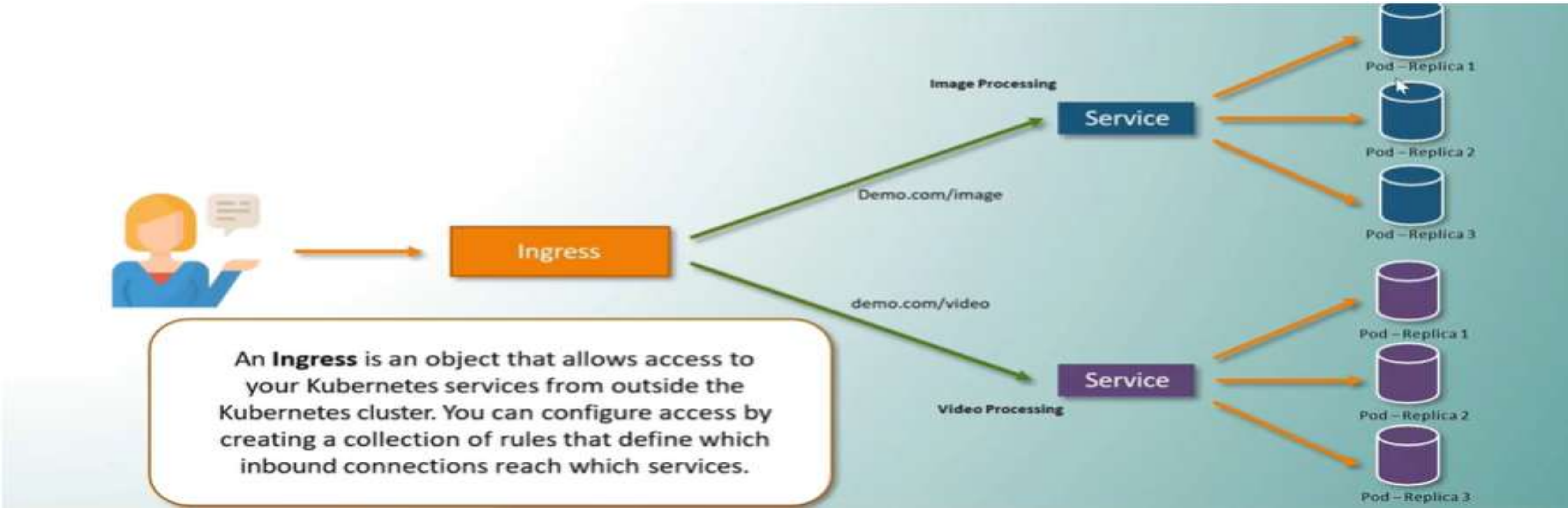
Creating a NodePort Service:

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

Kubernetes Objects – Service Types

Kubernetes Ingress

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



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.

Installing an Ingress Controller

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-v1.12.0-beta.0/deploy/static/provider/cloud/deploy.yaml
```

Kubernetes Commands

minikube start –> To start the Kubernetes cluster

kubectl get [pods/deploy/all] –> To list the resources in namespace

kubectl delete [pods/deploy] –> To delete the resources in particular namespace

Kubectl delete –f pod.yml → To delete the resources created using the pod.yml file

kubectl get all –> To list the resources in all the namespaces

kubectl get pods –o wide → will give more information regarding the pod

kubectl apply –f pod.yml -> To create the pod based on the pod.yml file

kubectl create –f pod.yml → To create the pod based on the pod.yml file

minikube ssh <ipaddress> → *For connecting to the pod container*

kubectl describe pod nginx → *Lists all the information of the pods (Can be used for debugging the pod)*

Kubernetes Sample Code

```
//pod.yml file
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - name: nginx
    image: nginx:latest
    ports:
    - containerPort: 80
```

- *kubectl apply –f pod.yml ->* To create the pod based on the pod.yml file
- *kubectl create –f pod.yml* → To create the pod based on the pod.yml file

References

Kubernetes Installation

How to run multiple Kubernetes clusters with Minikube

https://minikube.sigs.k8s.io/docs/handbook/vpn_and_proxy/

https://medium.com/geekculture/cheatsheet-for-kubernetes-minikube-kubectl-5500ffd2f0d5

Misc Commands:

```
minikube ssh <primary-node>
minikube ssh –n <secondary-node>
```