

KUBERNETES

Kubernetes is also know as ‘K8S’ it is an open source container orchestration platform developed by the GOOGLE but later it is hand over to CNCF (cloud native computing foundation).

Kubernetes is an open-source container orchestration engine that automates the process of deploying, scaling, and management of containerized applications across a cluster of nodes.

Kubernetes provides a consistent and reliable way to manage applications, regardless of whether they are running on on-premises, in the cloud, or in hybrid environments.

That mean Kubernetes is built to be used anywhere, allowing you to run your applications across on-site deployments and public clouds; as well as hybrid deployments in between. So you can run your applications where you need them.

Kubernetes can be used with or without Docker.

Kubernetes uses Docker to deploy, manage, and scale containerized applications.

Kubernetes is amazing it simplifies complex tasks, making it easier for DEVOPS engineers to manage applications seamlessly (smoothly). So big companies like NASA, chatgpt, Microsoft, google, Spotify almost every one rely (depend) on Kubernetes.

KUBERNETES CLUSTER

A Kubernetes (K8s) cluster is a group of computing nodes, or worker machines, that run containerized applications. Containerization is a software deployment and runtime process that bundles an application’s code with all the files and libraries it needs to run on any infrastructure.

A Kubernetes cluster is a group of one or more nodes with running pods. Within the cluster, the Kubernetes control plane manages nodes and pods.

Kubernetes is an open source container orchestration software with which you can control, coordinate, and schedule containers at scale. Kubernetes places containers into pods and runs them on nodes.

A Kubernetes cluster has, at a minimum, a master node running a container pod and a control plane that manages the cluster. When you deploy Kubernetes, you are essentially running a Kubernetes cluster.

**ARCHISTRUCTURE OF KUBERNETES CLUSTER**

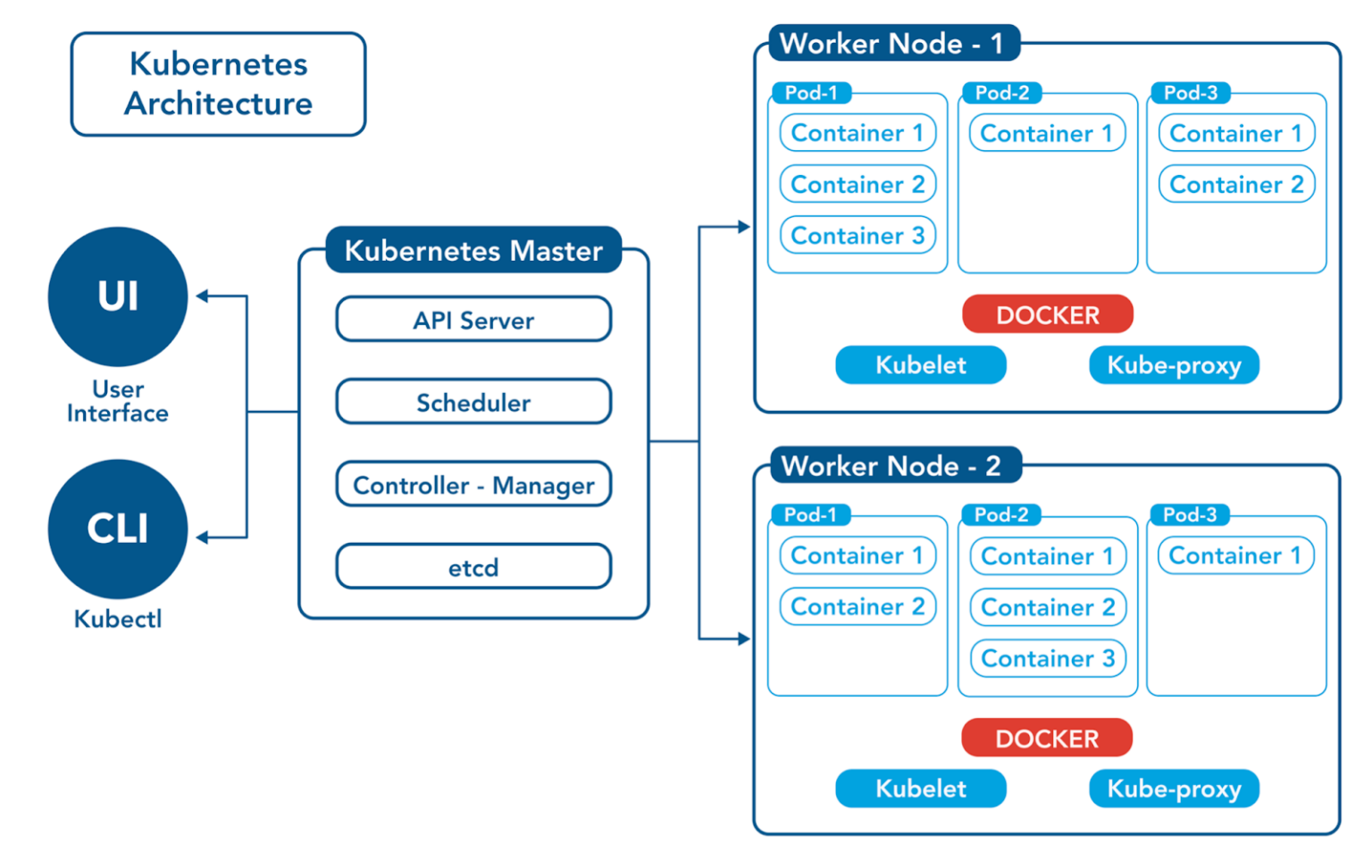
****

Fig: Architecture of Kubernetes cluster

The architecture of Kubernetes cluster mainly consist of two nodes

1. Master Node (control plane).
2. Worker Node (worker machine).

A Kubernetes cluster required at least one worker node, where our containers will be deployed.

1. **Master node (control plane):** Manages the cluster

The master node is a centralized control plane that manages the entire cluster. It act as a brain of Kubernetes cluster, making worldwide decisions about the cluster (for example, scheduling), as well as detecting and responding to cluster events (like starting a new pod while a deployment’s replicas field is unsatisfied). And it manages worker nodes and pods.

The key components that are present in master node are

1. API Server
2. Controller manager,
3. Scheduler,
4. Etcd

**Note:** A master node cannot run any user applications

1. **API (application programing interface) Server. (second important component)**

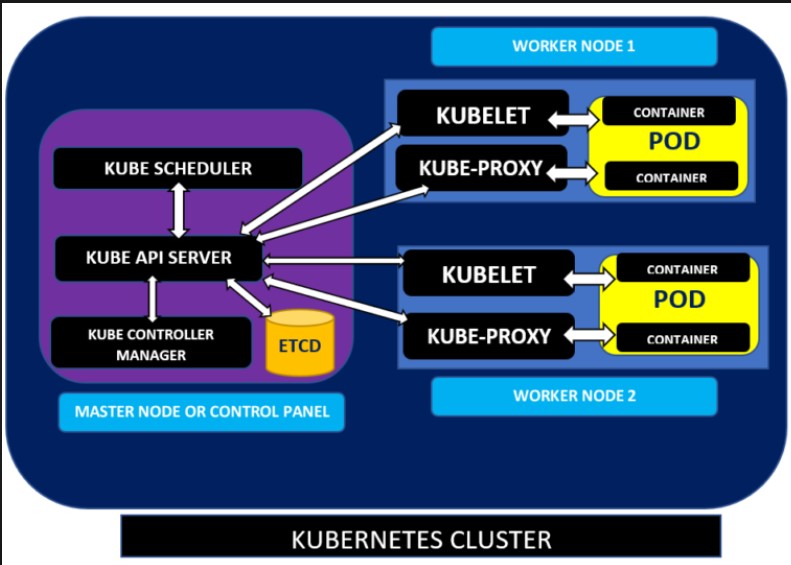
**KUBE API server is nothing but KUBCTL command which is used in cloud shell.**

It act as the primary interface for cluster communication.

It is the center of communication between all the components inside the cluster. That means it is the central management entity in the Kubernetes cluster.

All other components are communicate with the API server to read or modify the cluster state.

API server will receive all the requests from the outside of the cluster and this information is stored/updated in the ETCD. And also it informs the Scheduler about the requests and then the scheduler will assigns the task to the worker nodes.



The API server is the front end for the Kubernetes control plane.

It is like an initial gateway to the cluster that listens to updates or queries via CLI (command line interface) like Kubectl. Kubectl communicates with API Server to inform what needs to be done like creating pods or deleting pods etc.

It also act as a gatekeeper. It generally validates requests received and then forwards them to other processes. No request can be directly passed to the cluster, it has to be passed through the API Server.

Kube-API Server validates, process the requests, and configure the data for the API objects. And then updates the corresponding object state in etcd.

1. **Etcd: (first important component)**

Etcd is a key-value data store (no SQL DB) used to store and manage the critical information that distributed systems need. It provides a reliable way of storing the configuration data.

ETCD basically stores whole cluster information, like pod, worker node and replica set information’s

The name “etcd” comes from a naming convention within the Linux directory structure: In UNIX, all system configuration files for a single system are contained in a folder called “/etc;” “d” stands for “distributed.”

It is a key-value store of a Cluster, used as Kubernetes' backing store for all cluster data. Etcd offers a stable backend for cluster state and configuration management.

The main goal of Etcd is to securely store important data within a distributed system via the Raft consensus system.

It stores the configuration data that represents the state of entire cluster including information about nodes, pods, services, and more.

By using the Etcd we can store and retrieve the configuration data for all cluster components.

**Some of the features of Etcd:**

* **Fully replicated:** Every node in an etcd cluster has access to the full data store.
* **Highly available:** etcd is designed to have no single point of failure and gracefully tolerate hardware failures and network partitions.
* **Reliably consistent:** Every data ‘read’ returns the latest data ‘write’ across all clusters.
* **Fast:** etcd has been benchmarked at 10,000 writes per second.
* **Secure:** etcd supports automatic Transport Layer Security (TLS) and optional secure socket layer (SSL) client certificate authentication.

**C). SCHEDULER:**

A scheduler is a control plane component that watches or looks for newly created pods with no assigned node**.** And it intelligently decides on which node to schedule the pod for better efficiency of the cluster.

That means it can perform

* Watches for newly created pods with no assigned node.
* Selects an appropriate node for each pod based on resource requirements, node capacity, and other constraints.
* Updates the pods information in API server, assigning it to a specific node.
* Regularly checks the resource utilization of every node and schedules pods to hold the desired state.
* Enhances cluster efficiency by using distributing workloads intelligently.
* And assigns the tasks to the KUBELET which is present in worker node.

**D). CONTROLLER MANAGER:**

The Kubernetes Controller Manager (also called kube-controller-manager) is a “daemon” that acts as a continuous control loop in a Kubernetes cluster. The controller monitors the current state of the cluster via calls made to the API Server, and changes the current state to match the desired state described in the cluster’s declarative configuration.

The Controller Manager does not directly modify resources in the Kubernetes cluster. Instead, it manages multiple controllers responsible for specific activities—including replication controllers, endpoint controllers, namespace controllers, and service account controllers.

 It act as a control center for maintaining the desired state of your cluster.

The Kubernetes control plane consists of a core component called kube-controller-manager. This component is responsible for running multiple controllers that maintain the desired state of the cluster. These controllers are packaged together with the kube-controller-manager daemon.

Kubernetes resources are defined by a manifest file written in YAML. When the manifest is deployed, an object is created that aims to reach the desired state within the cluster. From that point, the appropriate controller watches the object and updates the cluster’s existing state to match the desired state.

The controller adjusts resources in the cluster based on the existing nodes, the resources they have available, the currently running workloads, and the policies defined for their behavior.

Some of the features are given below

* Watches the state of the cluster via the API server.
* Ensures that the current state matches the desired state.
* Runs controller processes responsible for handling tasks such as node and replication controller management.

**WORKER NODE:** runs the applications

When u create a Kubernetes cluster, you typically work with one or more worker nodes. As your applications demands grow, you can add more worker nodes to the cluster to scale your application horizontally.

Worker node is the second main block in the Kubernetes cluster. It is responsible for executing the containerized applications and tasks.

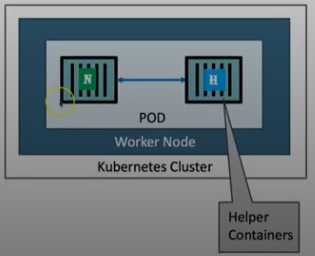
A single worker node consists of four main blocks

1. Pods
2. Kubelet
3. Kube-proxy
4. Docker.
5. Pods:

* A POD is a single instance of an application.
* A POD is the smallest object that we can create & manage in the Kubernetes.
* Podsare the smallest deployable units of computing that you can create and manage in Kubernetes.
* It is an **abstraction over a container** and represents **one or more containers** that share the same network and storage.
* Pods are assigned to IP address.
* We can have multiple containers in a single POD, but they are not same kind.

(**EX**: Two Nginx containers in a single pod serving same purpose is not recommended).

* These container are called as helper/supporting/side car containers.
* These containers can communicate each other easily as they share same network space. And uses the pods localhost interface to communicate with each other.
* They can also easily share same storage space.
* POD generally have one to one relationship with containers.
* We can easily scale up and scale down the POD’s.



Pods in a Kubernetes cluster are used in two main ways:

* **Pods that run a single container (recommended)**:
* The "one-container-per-Pod" model is the most common Kubernetes use case; in this case, you can think of a Pod as a wrapper around a single container; Kubernetes manages Pods rather than managing the containers directly.
* **Pods that run multiple containers that need to work together (rare use case)**.
* A Pod can summarize an application composed of multiple co-located containers that are tightly coupled and need to share resources. These co-located containers form a single cohesive unit.
* Grouping multiple co-located and co-managed containers in a single Pod is a relatively advanced use case. You should use this pattern only in specific instances in which your containers are tightly coupled.
* You don't need to run multiple containers to provide replication (for resilience or capacity); if you need multiple replicas.

**Note1:** One of main advantage of using pod is it is independent of any container engine, that means it can run on any container engine (containerization platform) such as Docker engine, podman engine, Linux engine and so on.

**Note2:** The containers which are present in the same pod, shares the same network and storage resources.

1. Kubelet: (node agent)

To deploy and manage the pods inside the worker node we have an agent named as Kubelet.

The Kubelet is an agent that runs on each or all worker nodes and communicates with Kubernetes master node (API server) to ensure that containers are running as expected.

It provide the information to the master node thought the API server about state of pods and how the containers are running inside the pods etc. And also it receives the commands or control signals from master node about where should be the pods running and to configure the containers inside the pods.

1. Kube-Proxy: (networking component)

Kube-proxy is a network proxy that runs on each node in the Kubernetes cluster. It maintains network rules on nodes which allow network communication to your Pods from network sessions inside or outside of your cluster.

That means Kubeproxy have all URL information of our applications which are deployed in the pods.

Kube-proxy manages networking in the Kubernetes cluster by directing traffic between pods and services, which makes load balancing and service discovery easier. It keeps an eye on the cluster constantly, dynamically adjusting load balancing and routing rules to achieve peak efficiency.

Kube-proxy is mainly used for load balancing, to establish pod to pod communication, and to make our application accessible by end users.

1. Docker: (container run time)

In order to create and manage the containers inside the worker node or machine of pods we need to have a container runtime interface (CRI) or software like Docker.

A container runtime, such as Docker is responsible for running containers on the worker node.

POD’S

Master node

KUBE Scheduler

Container

API Server

Helper container (or)

Side-car container

Controller

KUBELET

KUBE-Proxy

ETCD

Worker node

Fig: Kubernetes cluster.

**Note:** By default, Kubernetes **limits the number of Pods per node to 110** (for most cloud providers like AWS, GCP, and Azure).

NOTE-1:

Kubernetes are the best resource manager, it manages our servers, if these servers are in the cloud then they are called virtual servers, the fact is that different cloud providers have different names for this servers.

AWS----------EC2 (Elastic compute cloud)

Google--------Compute engine

Azure----------Virtual machines

Kubernetes----Nodes

NOTE-2:

Kubernetes have different names on clouds based on services.

Azure -----------AKS (azure Kubernetes services)

Amazon----------EKS (elastic Kubernetes services)

Google-----------GKE (google Kubernetes engine)

This all are come under the partially PAAS models, because here we not only control/manage the application but the worker node also in our control, so that way it is partially PAAS model.

**Working with google Kubernets engine GKE**

Fist in order to work with GKE fist we have to create on account in google cloud.

Then create a cluster inside the google cloud platform ether standard or autopilot mode cluster.

After by using cloud shell, which is present at top right cornet terminal logo. Then we have to deploy the Docker image from Docker hub. By using command

**Kubectl create deployment “repository name” –image=repository name: tag**

**Ex: kubectl create deployment hello-world-rest-api --image=in28min/hello-world-rest-api:0.0.1.RELEASE**

**Note: kubectl= Kube controller**

And then we can expose this Docker image by using the command

{Kubectl expose deployment hello-world-rest-api - -type= LoadBalancer

--port=8080}

All these commands are executed in google cloud shell only

**Note:** when we deploy image into GKC cluster it will creates deployments, replica set and pods.

Kubectl create deployment----> deployments, replica set & pods

And then when we expose our image in cloud shell it6v will creates services.

Kubectl expose deployment----> services.

In Kubernetes cluster the pods, deployments, replica sets and services are linked with labels and selectors.

**Replica set:**

A Replica Set (RS) is a Kubernetes is an object used to maintain or ensure a stable set of replicated pods running within a cluster at any given time.

Some of the benefits of using replica set are:

* Increased availability
* Load balancer
* Set-based selector
* And provides basic scaling mechanisms

That means replica set is mainly used to manage, monitor scale up and down of pods, this is done by replica set automatically based on need.

Some of basic command of replica set are:

Ex;

Kubectl explain replicasets >> shows basic information of replica sets (pods)

Kubectl get replicasert or rs >>shows the number of replicas created (pods)

Kubectl scale deployment hello-world-rest-api-5ffc8fdfb8 –replicas=3 >> it creates three replicated pods.

**Deployment:**

It is used to update the exist application with a zero down time.

The command is given as:

Kubectl set image deployment container-name container-name=new-image-name:tag

EX: 1) kubectl set image deployment tainee tainee=harish1000/tainee:v5

1. kubectl set image deployment hello-world-rest-api hello-world-rest-api=in28min/hello-world-rest-api:0.0.2.RELEASE.

NOTE:

Here we updated old version 0.0.1.RELEASE to new version 0.0.2.RELEASE.

Let us consider that we have an application or image V1 in Kubernetes cluster with 3-pods, then if you want to update it to V2 we use above command, this command launches up new single pod at a time then gradually it will creases the no of pods of V2 equal to V1-pods=3 by decreasing V1pods to zero. This method of updating is called rolling updates method.

**SERVICE:**

A service is created when we expose an application or image in Kubernetes cluster.

When we expose an application then only we can browse in any search engine

Service allows our application to receive traffic through a permanent life time address. It provides a constant front end interface irrespective of whatever changes are happing in the back end.

Questions

Q1: can you run only Docker containers in Kubernetes cluster?

A: no

Q2: does master node run any of user application?

A: “no” because master node can only controls worker nodes.

Q3: what happen to an application when master node goes down?

A: the application can continuously run even when master node is goes down.

ROLLOUT:

1. Kubectl rollout history deployment hello-world-rest-api (image-name)

* It shows the revision and change cause of an application.

Revision change-cause

1 <none> (0.0.1.release)

2 <none> (0.0.2.release)

3 <none> (0.0.3.release)

And also it is used to Downgrade an application that means we can go previous version of any application

EX: kubectl rollout undo deployment hello-world-rest-api --to-revision=1

* It will change the version from 0.0.2,release to 0.0.1.release.

YAML FILE

YAML is a human-readable format of data serialization. In Kubernetes, YAML files are used to define and configure resources, such as pods, Kubernetes services and Kubernetes deployments.

The beauty of YAML is its simplicity as it uses key-value pairs and avoids complex syntax. YAML is smaller and simpler than JSON and XML files making them easier to manage and process.

Generally managing containerized applications is tough in Kubernetes, hence, choosing right configuration format is necessary.

YAML solves this problem for the Kubernetes as it is simpler than its alternatives like JSON (java script object notation).

**Key Components of YAML in Kubernetes**

### **Indentation Matter**

### Key-Value Pairs

### Lists and Arrays

1. **Indentation matter:**

Here indentation is nothing but the use of space or tabs

Indentation is more important in YAML file just like in python

By using the white space we can define the structure of code blocks such as loops, function..

1. **Key-Value pair:**

* **Fundamental Data Unit:**YAML uses key-value pairs to represent data. Each key is followed by a colon (:), and the value is placed on the next line.
* **Common Keys In Kubernetes**
  + **apiVersion:**It specifies with what version the Kubernetes API configuration is compatible with.
  + **Kind:**It is used to identify the type of Kubernetes resource being defined.
  + **Metadata:** It contains information about the resource itself, such as its name, labels, and annotations.

1. **List and arrays**

* **Hyphenated Lists**: In Kubernetes, Lists are created using hyphens (-), with each item starting on a new line and indented at the same level.
* **Examples In Kubernetes:**
  + Container Specifications: A pod’s spec section includes a list of containers which are to be run within the pod.
  + Deployment Containers: A deployment manifest includes a list of containers to be deployed and managed as a group.

**EXAMPLE:**

apiVersion: v1  
kind: Pod  
metadata:  
 name: my-pod  
spec:  
 containers:  
 - name: container1  
 image: nginx  
 - name: container2  
 image: busybox

**How to create YAML file:**

If we have a repository in the Docker hub when we pull it from hub to local machine. Then we can create a deployment.yaml file and service.yaml file by using following commands

1. Kubectl get deployment deployment-name o yaml > deployment.yaml

Ex: kubectl get deployment hello-world-rest-api o yaml > deployment.yaml

1. kubectl get service deployment-name o yaml > service.yaml

Ex: kubectl get service hello-world-rest-api o > service.yaml

If we modify anything in the YAML file (deployment or service) manually we can deploy it by using a command is

* Kubectl apply –f deployment.yaml
* Kubectl apply –f service.yaml

INGRESS

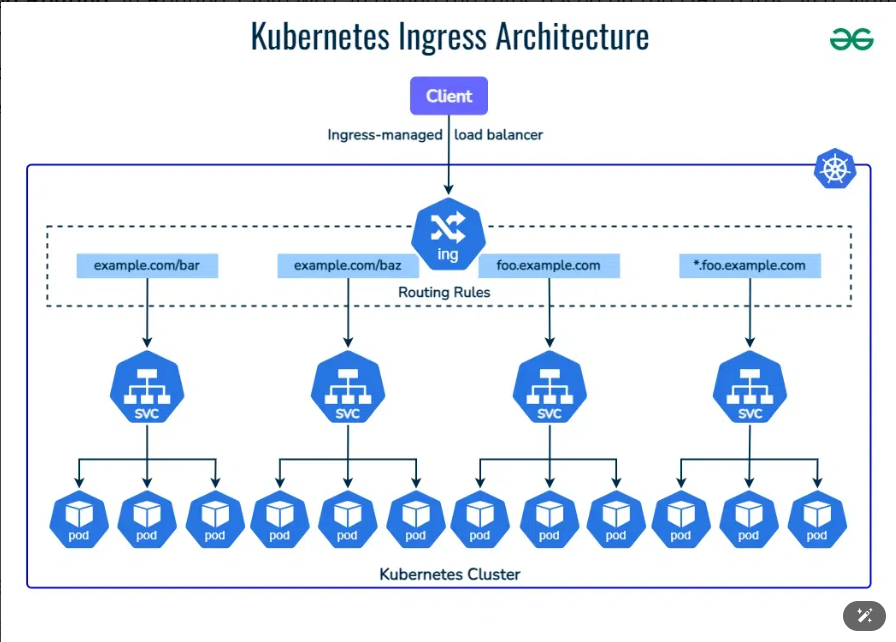
**Ingress is an**[**API**](https://www.geeksforgeeks.org/what-is-an-api/)**object that manages external access to the services in a Cluster**“. The role of Ingress is instead of Service, the request goes first to Ingress and it does the forwarding then to the Service.

Kubernetes Ingress acts like a traffic controller for managing the incoming traffic to the Kubernetes cluster. It manages the external access to the services within the Kubernetes.

It provides a single entry point into a cluster hence making it simpler to manage applications and troubleshoot routing issues.

 Kubernetes Ingress defines the rules in the Routing table for external HTTP and HTTPS traffic to the inside Kubernetes cluster services based on the hostname and paths.

It act as centralized routing device



Example:

If we have too many micro services in on YAML file format, using a LaodBalancer as a “type” for every micro service is too expansive to use, so instead of LoadBalancer we uses the ingress (NodePort) as a ‘type’ in the service.yaml file.

Ex:

apiVersion: v1

kind: service

metadata:

labels:

app: currency-conversion

name: currency-conversion

namespace: default

spec:

ports:

- Port: 8000

Protocol: TCP

targetport: 8000

Selector:

app: currency-conversion

sessionAffiinity: None

type: NodePort