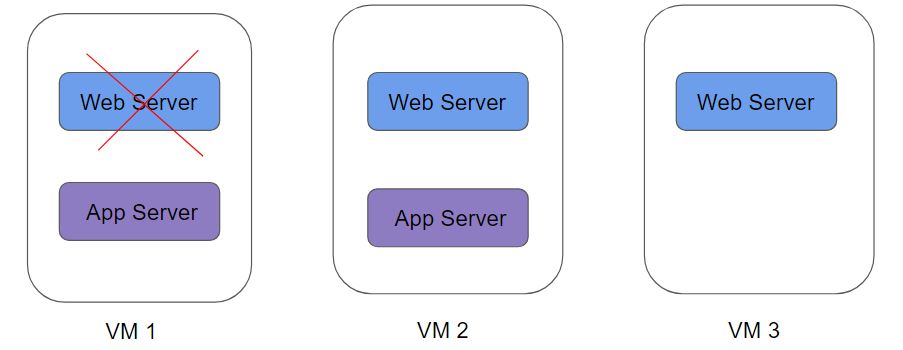
**Module 1:  Overview of Container Orchestration**

Container orchestration is all about managing the life cycles of containers, especially in large, dynamic environments.



Container Orchestration can be used to perform a lot of tasks, some of them includes:

* Provisioning and deployment of containers
* Scaling up or removing containers to spread application load evenly
* Movement of containers from one host to another if there is a shortage of resources
* Load balancing of service discovery between containers
* Health monitoring of containers and hosts

There are many container orchestration solutions which are available, some of the popular ones include:

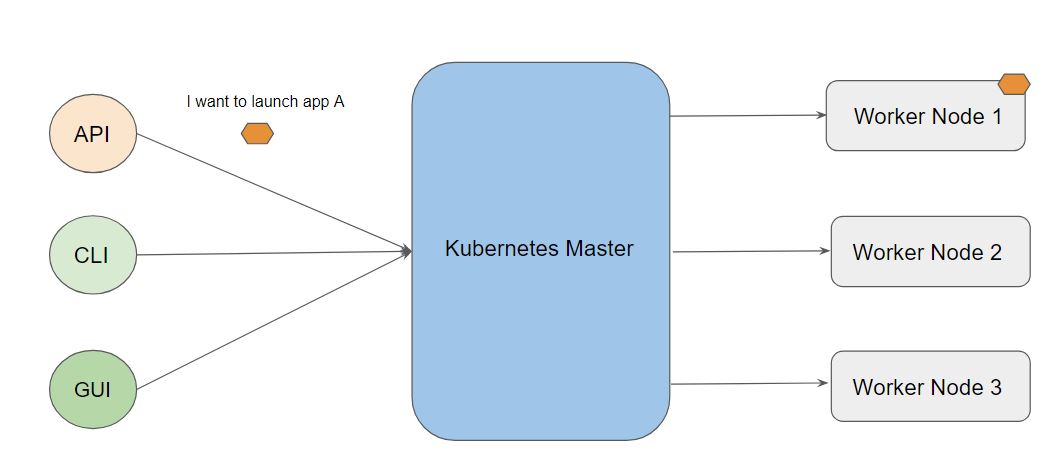
* Docker Swarm
* Kubernetes
* Apache Mesos
* Elastic Container  Service (AWS ECS)

There are also various container orchestration platforms available like EKS.

**Module 2:  Introduction to Kubernetes**

Kubernetes  (K8s)  is an open-source container orchestration engine developed by Google.

It was originally designed by Google and is now maintained by the Cloud Native Computing Foundation.



**Module 3:  Installation Options for Kubernetes**

There are multiple ways to get started with a fully functional Kubernetes environment

1. Use the Managed Kubernetes Service
2. Use Minikube
3. Install & Configure Kubernetes Manually (Hard Way)

3.1 Managed Kubernetes Service

Various providers like AWS, IBM, GCP, and others provide managed Kubernetes clusters.

Most organizations prefer to make use of this approach.



3.2 Minikube

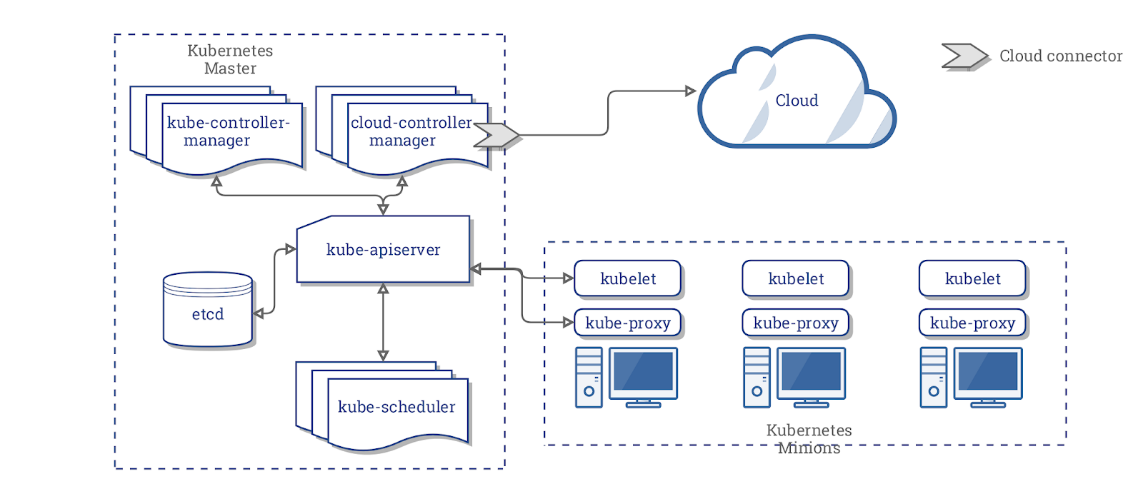
Minikube is a tool that makes it easy to run Kubernetes locally.

Minikube runs a single-node Kubernetes cluster inside a Virtual Machine (VM) on your laptop for users looking to try out Kubernetes or develop with it day-to-day.



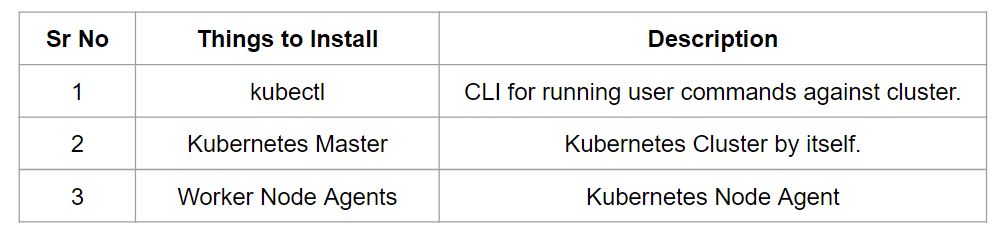
3.3 Kubernetes the Hard Way

In this approach, you install and configure components of Kubernetes individually.



3.4 Installation Configuration:

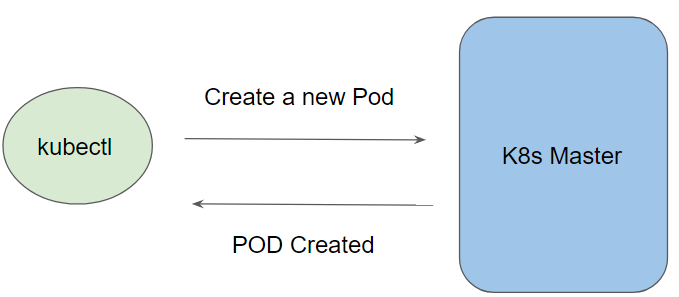
Things to configure while working with Kubernetes.



**Module 4:  Overview of kubectl**

The Kubernetes command-line tool, kubectl, allows you to run commands against Kubernetes clusters.

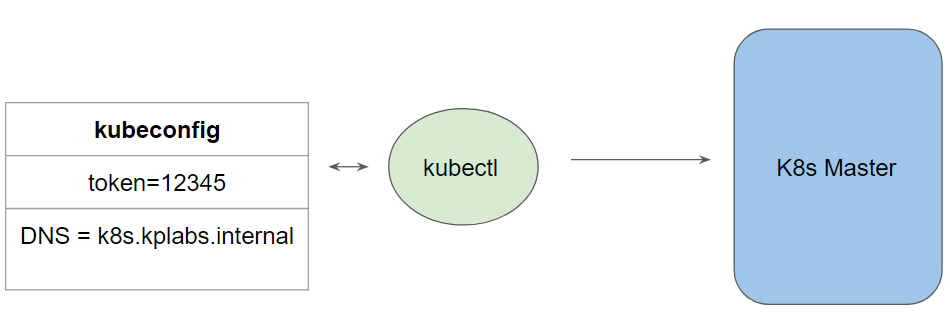
You can use kubectl to deploy applications, inspect and manage cluster resources, and view logs.



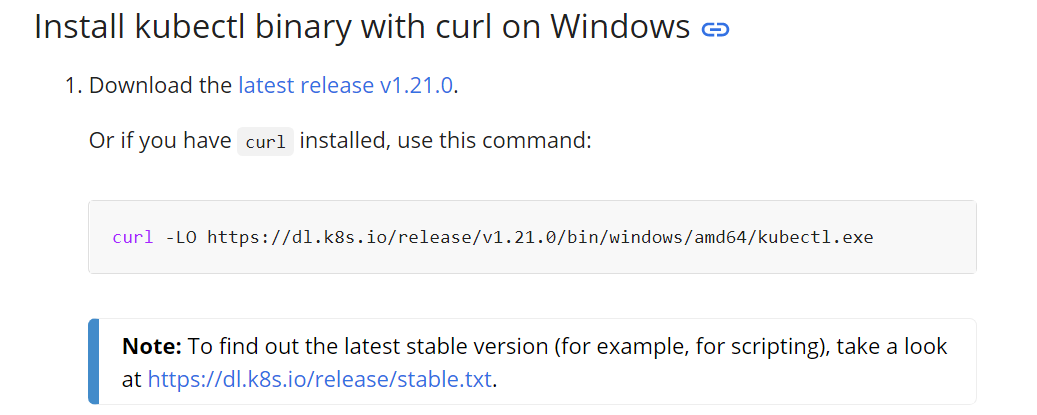
Let us understand the high-level workflow:

To connect to the Kubernetes Master, there are two important data which kubectl needs:

* DNS / IP of the Cluster
* Authentication Credentials

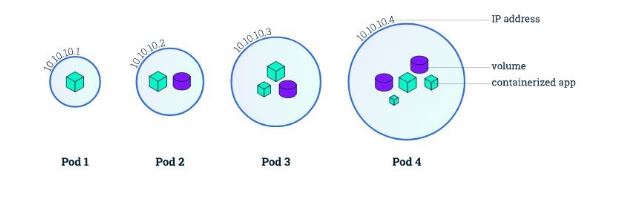


The overall installation of kubectl is straightforward and can be installed on a variety of Linux platforms, macOS and Windows

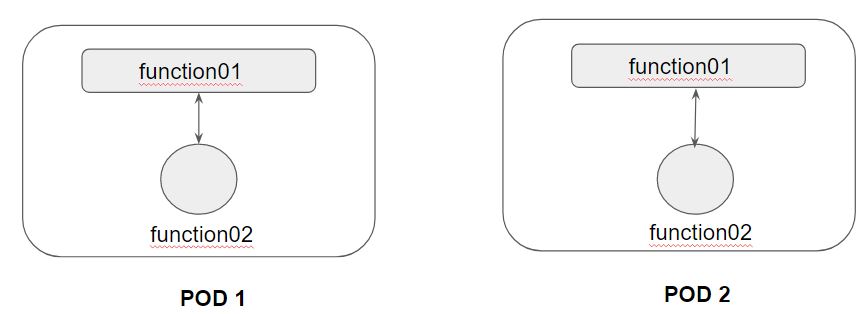


**Module 5:  PODS**

A Pod in Kubernetes represents a group of one or more application containers and some shared resources for those containers.



Containers within a Pod share an IP address and port space and can find each other via the localhost.



A Pod always runs on a Node.

A Node is a worker machine in Kubernetes.

Each Node is managed by the Master.

A Node can have multiple pods.

**Module 6:  Kubernetes Object**

Kubernetes Objects is basically a record of intent that you pass on to the Kubernetes cluster.

Once you create the object, the Kubernetes system will constantly work to ensure that object exists.

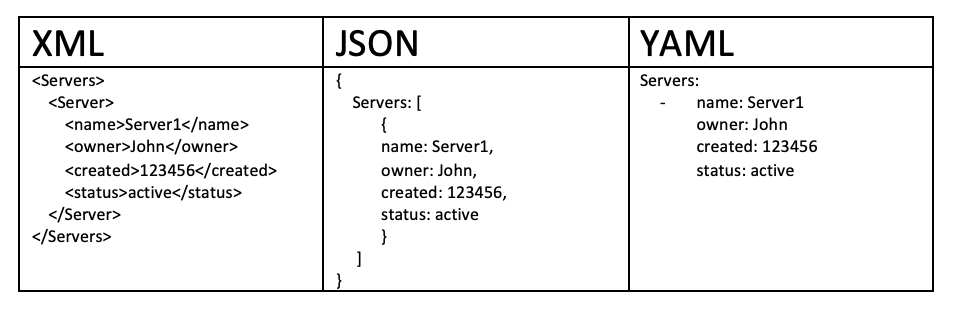
There are various ways in which we can configure a Kubernetes Object.

* The first approach is through the kubectl commands.
* The second approach is through a configuration file written in YAML.

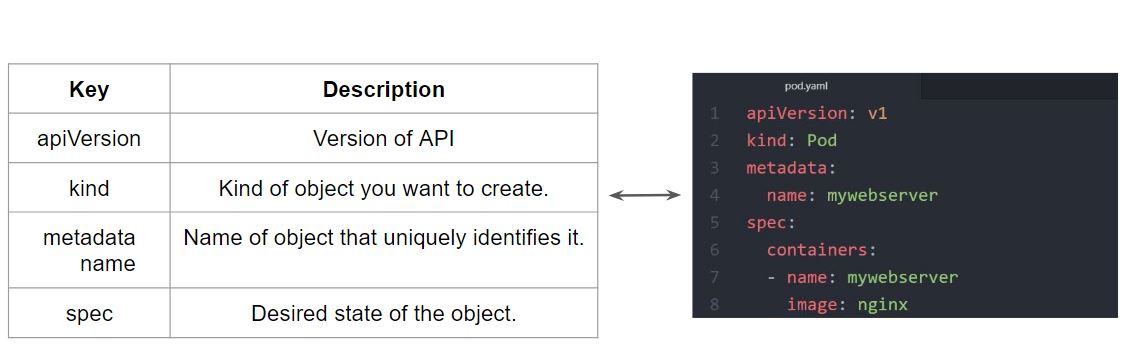


YAML is a human-readable data-serialization language.

It designed to be human friendly and works perfectly with other programming languages.

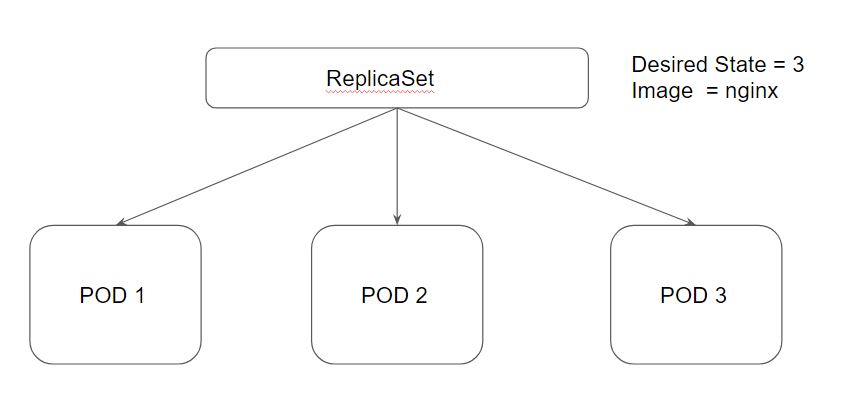


**Module 7:  Creating First POD Configuration in YAML**

****

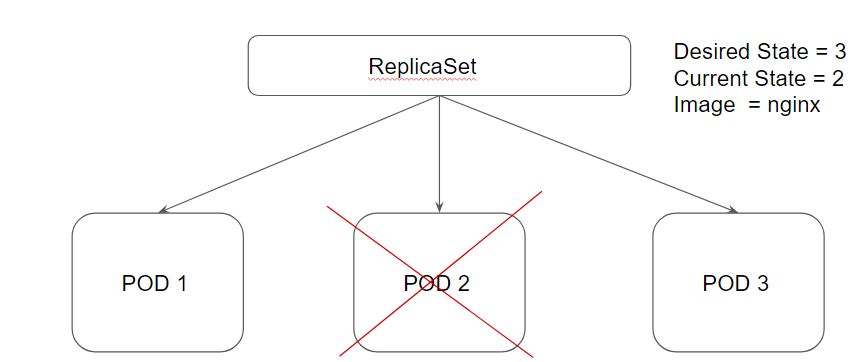
**Module 8:  ReplicaSets**

A ReplicaSet purpose is to maintain a stable set of replica Pods running at any given time.



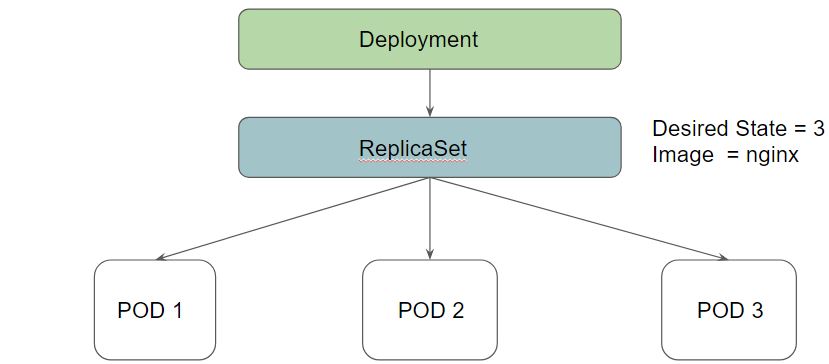
Desired State -  The state of pods which is desired.

Current State - The actual state of pods that are running.



**Module 9:  Deployments**

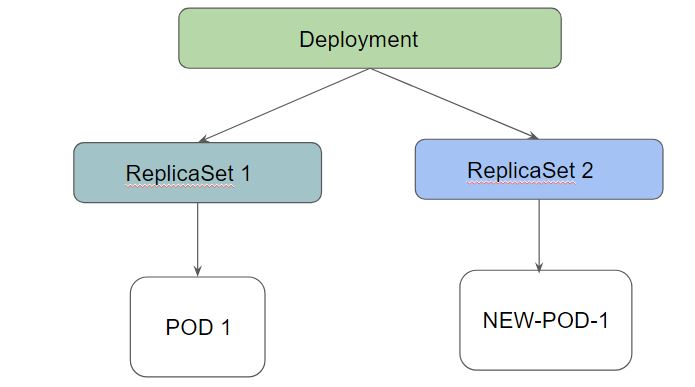
Deployments provide replication functionality with the help of ReplicaSets, along with various additional capability like rolling out of changes, rollback changes if required.



3.1 Benefits of Deployment - Rollout Changes

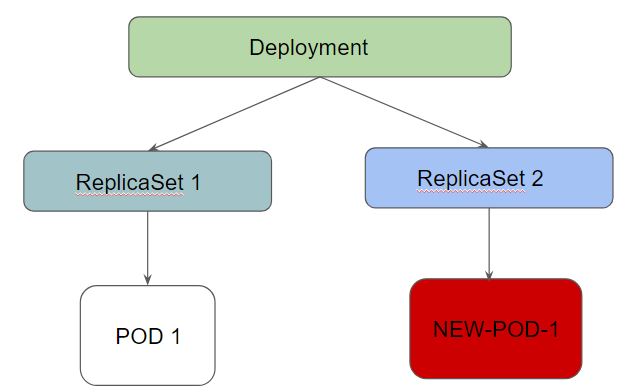
We can easily roll out new updates to our application using deployments.

Deployments will perform an update in a rollout manner to ensure that your app is not down.



3.2 Benefits of Deployment - Rollback Changes

Sometimes, you may want to rollback a Deployment; for example, when the Deployment is not stable, such as crash looping



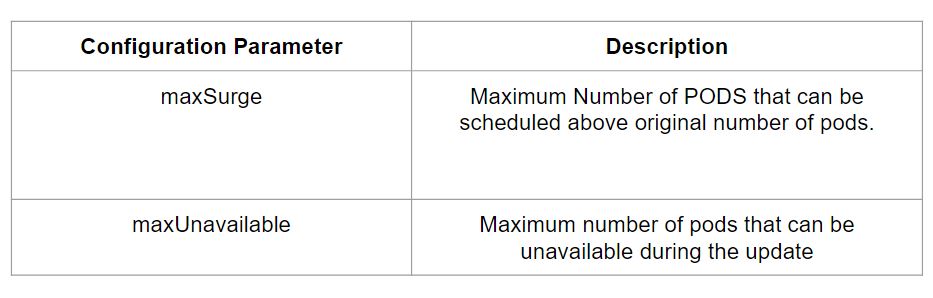
Deployment ensures that only a certain number of Pods are down while they are being updated.

By default, it ensures that at least 25% of the desired number of Pods are up (25% max unavailable).

Deployments keep the history of revision which had been made.

**Module 10:  Deployment Configuration**

While performing a rolling update, there are two important configurations to know.



maxUnavailable=0 and maxSurge=20%  << Full Capacity is maintained.

maxUnavailable=10% and maxSurge=0   << Update with no extra capacity. In-place updates.

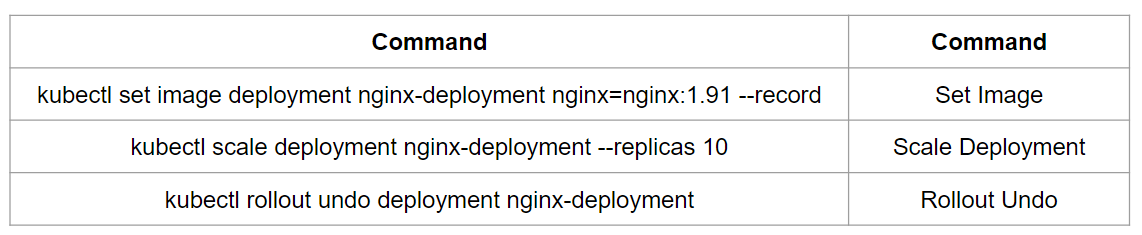
If you want fast rollout, make use of maxSurge.

If there might be a resource quota in place and partial unavailability is acceptable, maxUnavailable can be used.

**Module 11: Important Pointer - Deployments**

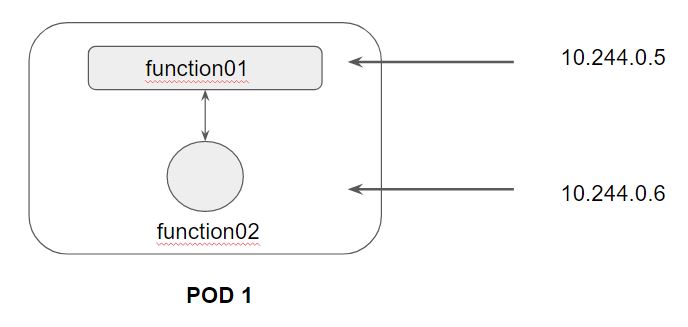
1. You should know how to set a new image to deployment as part of rolling update.
2. You should know the importance of --record instruction.
3. You should know how to rollback a deployment.
4. You should be able to scale the deployment

Following are some of the important commands to remember:

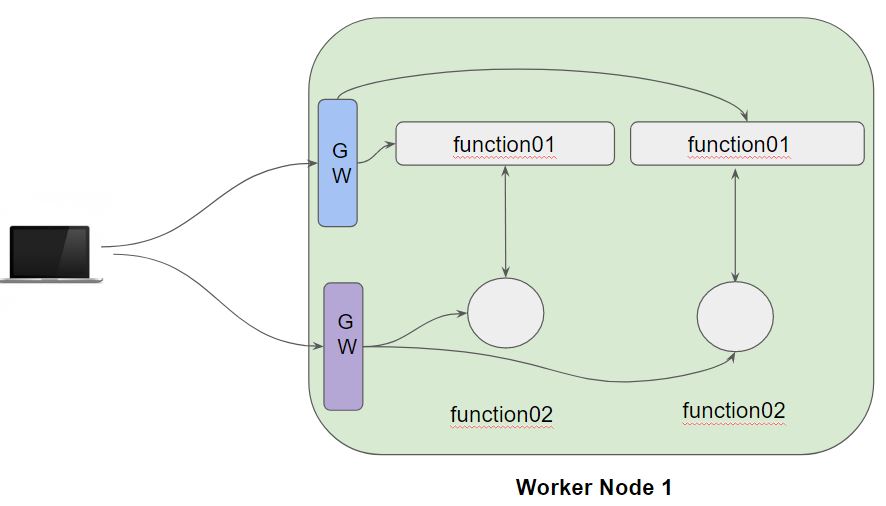


**Module 12:  Overview of Service**

Whenever you create a Pod, the containers created will have Private IP addresses.



Following is a high-level diagram on the functionality of Service:



In a Kubernetes cluster, each Pod has an internal IP address.

Pods are generally ephemeral, they can come and go anytime.

We can make use of service which acts as a gateway and can get us connected with right set of pods.

Service is an abstract way of exposing application running in the pods as a network service.

There are several types of Kubernetes Services which are available:

* NodePort
* ClusterIP
* LoadBalancer
* ExternalName

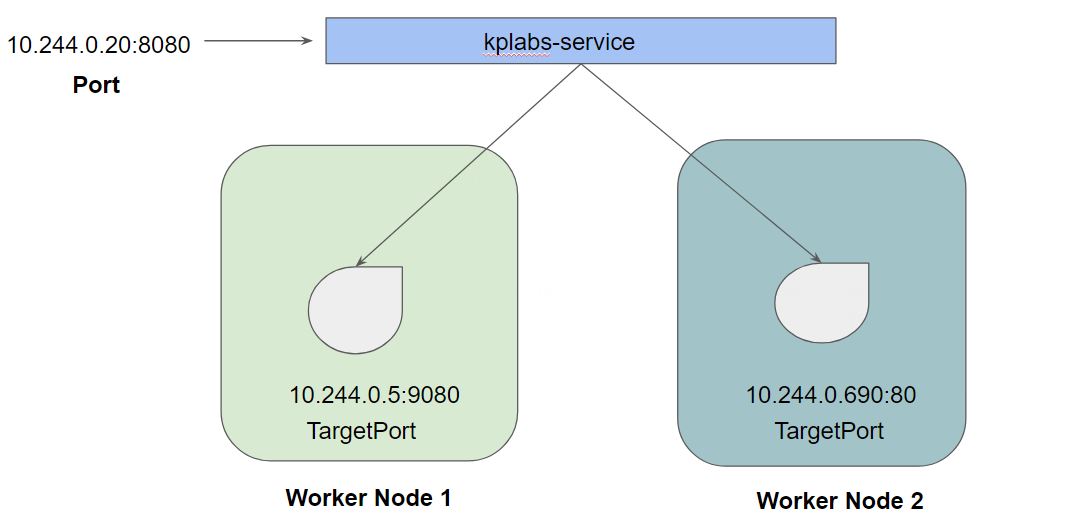
**Module 13:  Service Type - ClusterIP**

Whenever the service type is ClusterIP, an internal cluster IP address is assigned to the service.

Since an internal cluster IP is assigned, it can only be reachable from within the cluster.

This is a default ServiceType.

**Module 14:  Port vs TargePort**

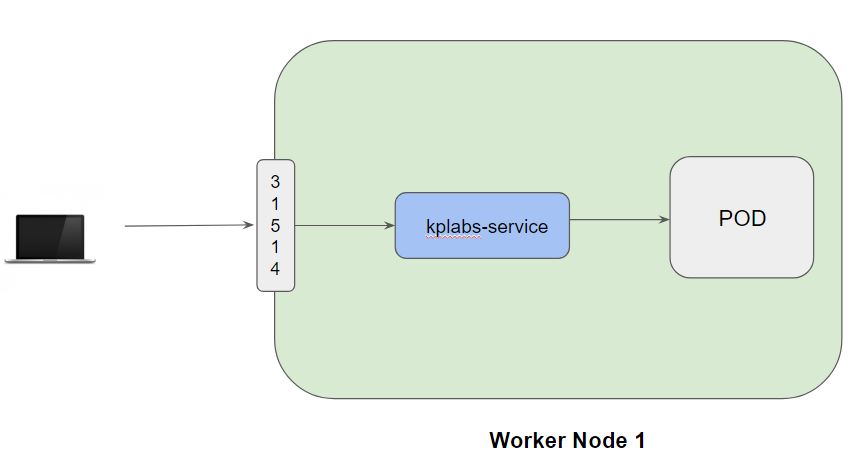
****

**Module 15:  Service Type - NodePort**

From the name, we can identify that it has to do with opening a port on the nodes.

If the service type is NodePort, then Kubernetes will allocate a port (default: 30000-32767) on every worker node.

Each node will proxy that port into your service.



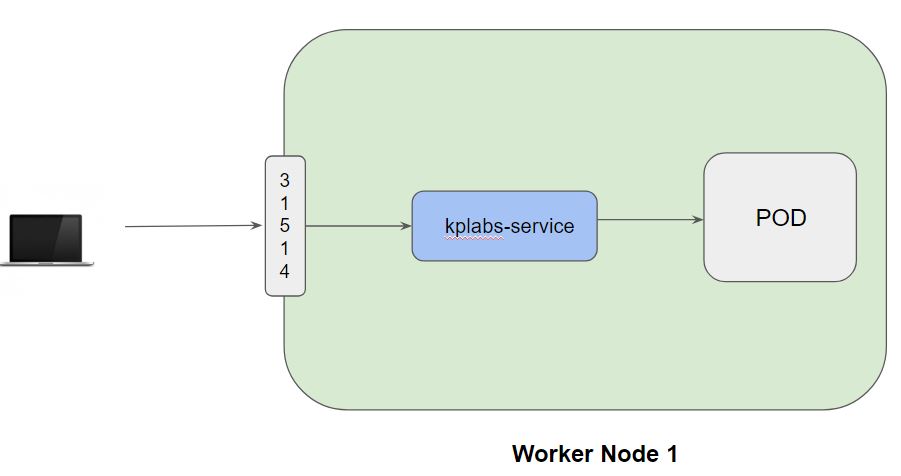
**Module 16:  Service Type - LoadBalancer**

16.1 Challenges with NodePort

We know that NodePort ServiceType will assign a node in all the worker node which can forward the traffic to the underlying service.

Challenge in NodePort:  We need to access it via IP/DNS:Port

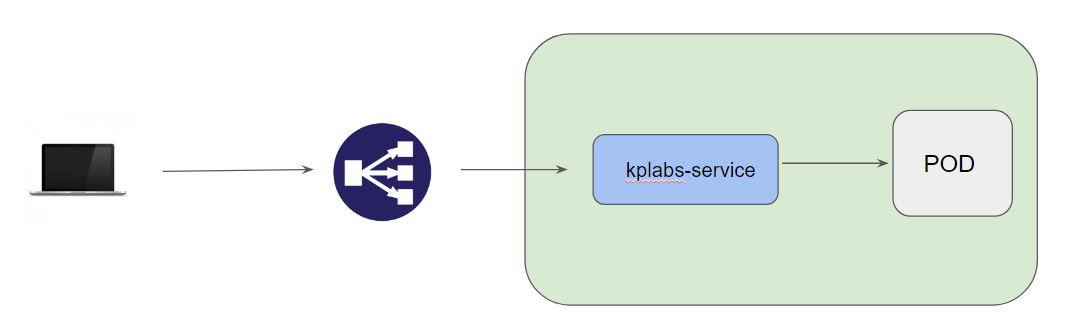
Example:  google.com:31514



16.2 Understanding LoadBalancer Service Type

LoadBalancer Service Type will automatically deploy an external load balancer.

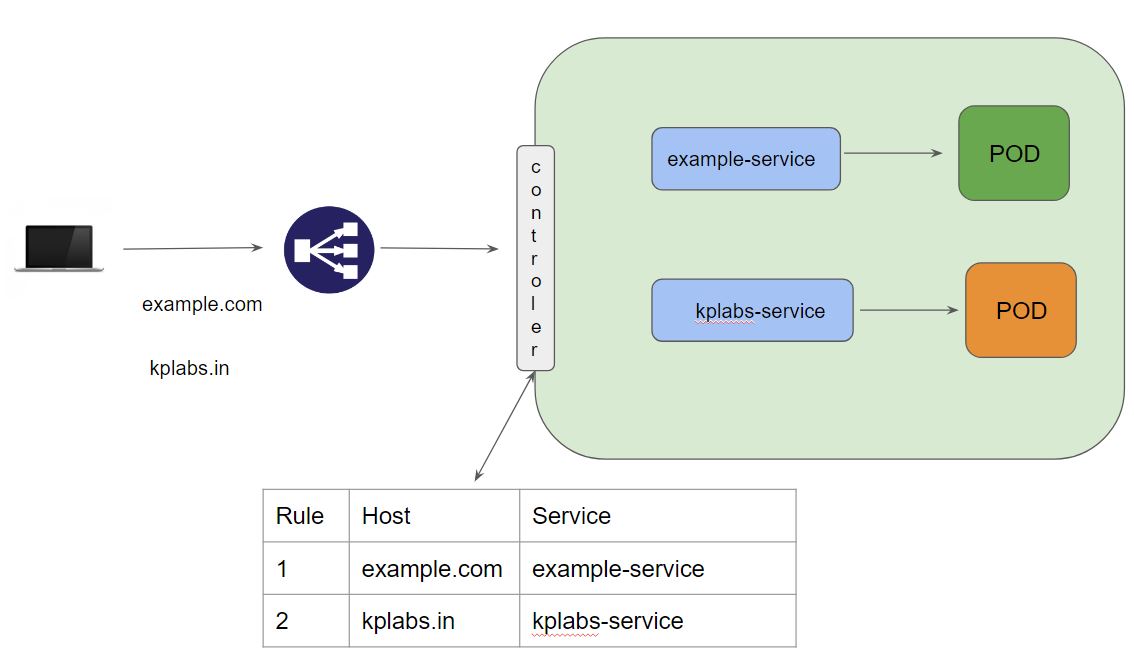
This load balancer takes care of routing requests to the underlying service.



16.3 Important Pointers

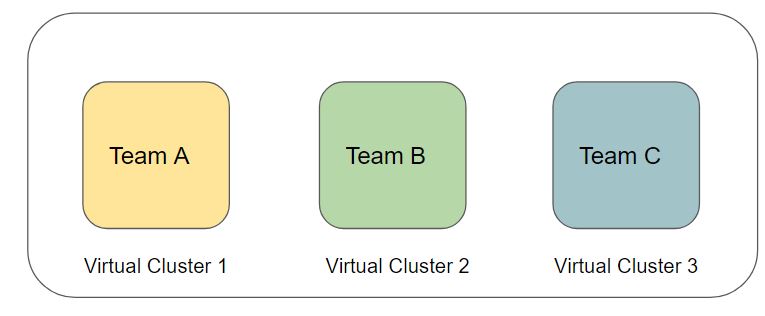
The overall implementation of LoadBalancer depends on your Cloud Provider.

If you plan to use it in bare-metal, then you will have to provide your own load balancer implementation.

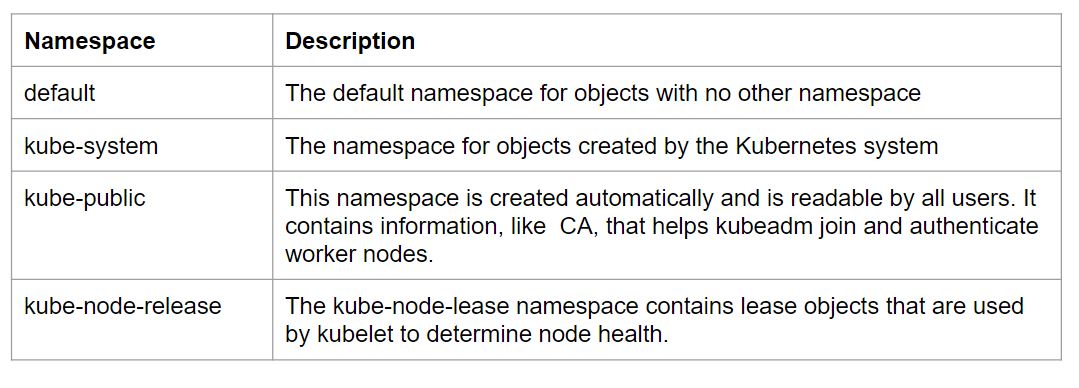


**Module 17:   Namespace**

Kubernetes supports multiple virtual clusters backed by the same physical cluster. These virtual clusters are called namespaces.



Following is the list of namespaces that are available in Kubernetes:



**Module 18:   Ingress**

[Ingress](https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.23/#ingress-v1-networking-k8s-io) exposes HTTP and HTTPS routes from outside the cluster to [services](https://kubernetes.io/docs/concepts/services-networking/service/) within the cluster. Traffic routing is controlled by rules defined on the Ingress resource.

Here is a simple example where an Ingress sends all its traffic to one Service:

