**K8S (Kubernetes)**

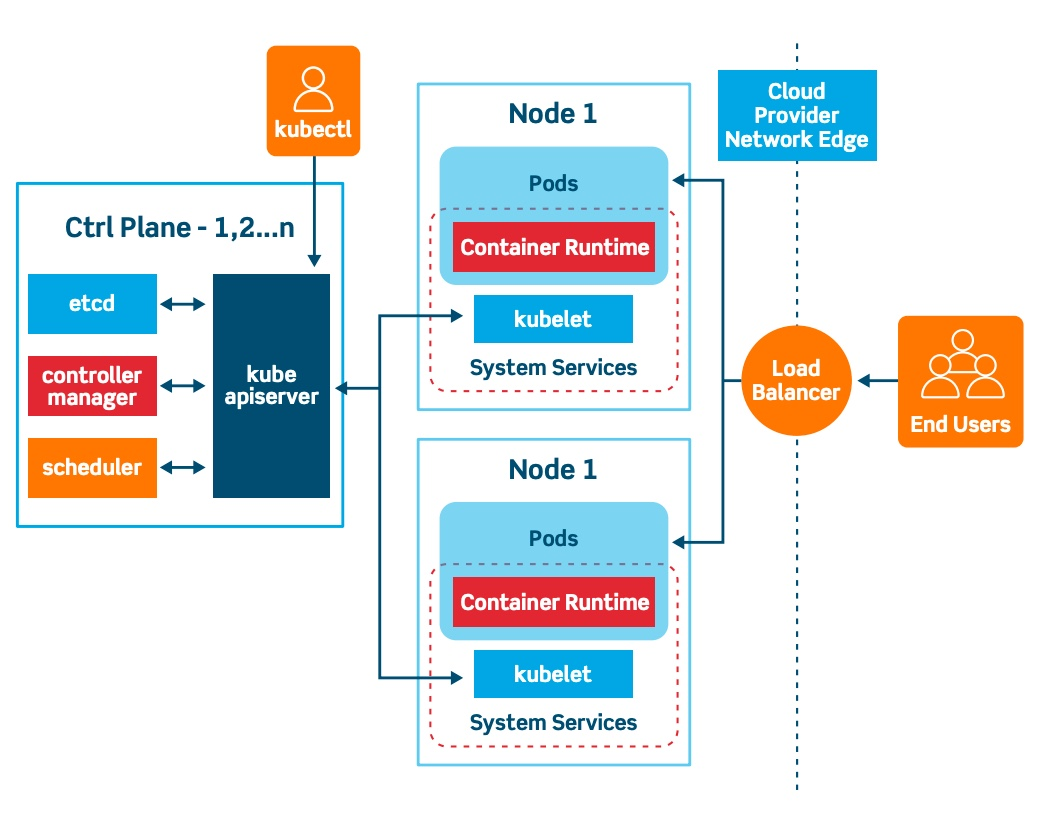
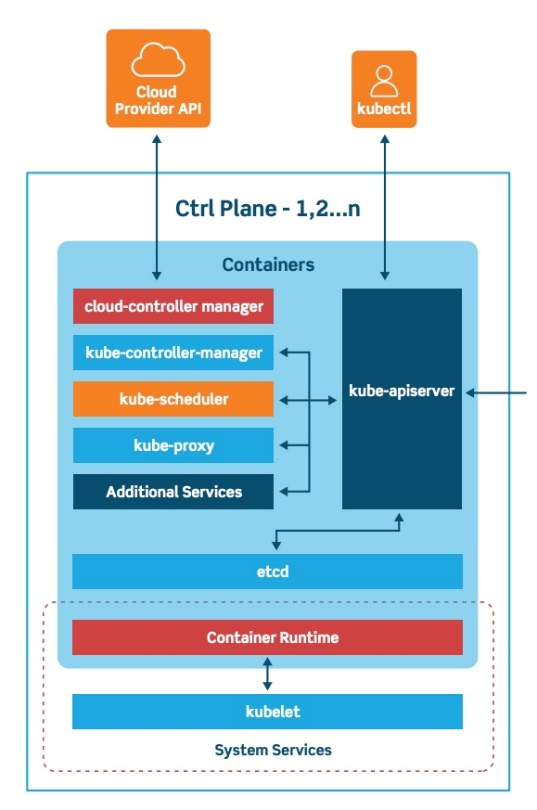
Kubernetes is an open-source container orchestration tool developed by Google to help you manage the containerised / dockerised application supporting multiple deployment environments like on-premises, cloud, or virtual machines.

What are the main features of K8s offers?

* + - 1. Assures high availability with zero downtime.
      2. High-performance and scalable
      3. Reliable infrastructure to support data recovery.

Disadvantages: it is very complex or takes a lot of time for setting up the k8s infra

**K8s architecture or cluster:**

**K8s cluster has two major components**

Master node

Worker node

**Node:** physical machine or virtual worker machine where containers will be deployed by k8s.

**Cluster:** set of nodes grouped together.

The master or control panel manages the cluster

**Components of K8s:**

API server, scheduler, control manager etcd, kubelet, container runtime, and kube-proxy.

**Kubernetes Basics**

we will learn some important Basics of Kubernetes:

**Cluster:**

It is a collection of hosts(servers) that helps you to aggregate their available resources. That includes ram, CPU, disk, and their devices into a usable pool.

**Master:**

The master is a collection of components that make up the control panel of Kubernetes. These components are used for all cluster decisions. It includes both scheduling and responding to cluster events.

**Node:**

It is a single host which is capable of running on a physical or virtual machine. A node should run both kube-proxy, minikube, and kubelet which are considered as a part of the cluster.

**Namespace:**

It is a logical cluster or environment. It is a widely used method which is used for scoping access or dividing a cluster.

**Kubernetes Architecture:**

Kubernetes has two nodes – the master node and the worker/slave node.

**Master node:**

The master node is the first and most vital component which is responsible for the management of the Kubernetes cluster. it is the entry point for all kinds of administrative tasks. there might be more than one master node in the cluster to check for fault tolerance.

The master node has various components like an API server, Controller manager, kubectl, scheduler, and ETCD.

**ETCD**:

* This component stores the configuration details and essential values in **key value form**

{

“name”: “xyz”,

“part”: “ght”

}

* It communicates with all other components to receive the commands to perform an action.
* It is also called as meta data storage.

**Controller Manager:**

* A daemon (server) that runs in continuous loop and is responsible for gathering information and sending it to the API server.
* Works to get the shared set of clusters and change them to the desired state of the server.
* The key controllers are the replication controllers, endpoint controllers, namespace controllers and service account controllers.
* The controller manager runs controllers to administer nodes and endpoints.

**Scheduler**

* The Schedular assigns the tasks to the slave nodes.
* It is responsible for distributing the workload and stores resource usage information on every node.

**API Server**

* Kubernetes uses the API server to perform all operations on the cluster.
* It is central management entity that receives all REST requests for modifications, serving as frontend to the cluster.

**Kubectl**

* Kubectl controls the Kubernetes cluster manager.

**Worker/Slave nodes**

Worker nodes are another essential component which contains all the required services to manage the networking between the containers, communicate with the master node, which allows you to assign resources to the scheduled containers.

* Kubelet: gets the configuration of a Pod from the API server and ensures that the described containers are up and running.
* Docker Container: Docker container runs on each of the worker nodes, which runs the configured pods.
* Kube-proxy: Kube-proxy acts as a load balancer and network proxy to perform service on a single worker node.
* Pods: A pod is a combination of single or multiple containers that logically runs together on nodes.

**Pods:**

Pods are smallest objects which can be created in Kubernetes cluster.

In terms of docker concepts, a pod is similar to a group of docker containers with shared namespace and shared file system volumes.

**YAML (yet another mark-up language) File: YAML** is a human-readable data serialization language. It is commonly used for configuration files and in applications where data is being stored or

Kubectl create -f filename

**The life cycle of k8s:**

Waiting: if the container is not either in a running or terminated state it is waiting.

Running: the running status indicates the container is running without issues

Terminated: the container in the terminated began execution and then either run to completion or fail for some reason.

**Publishing service:**

K8s service type allows you to specify what kind of service you want, the default is cluster IP

Type values and their behaviors are:

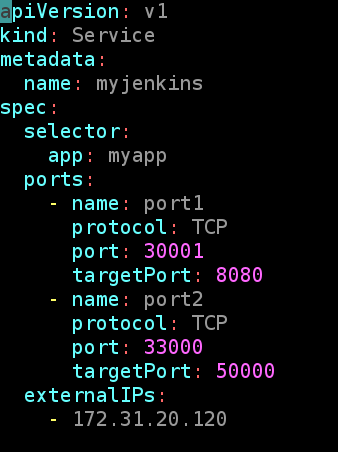
* + - 1. ClusterIP: Exposes the service on a cluster internal IP. Using this value makes the service only reachable from within the cluster. This is the default service type.
      2. NodePort: exposes the service on each nodes IP at a static port.

A cluster IP service, to which the nodes port service routes are automatically created.

* + - 1. LoadBalancer: exposes the service externally using a cloud providers load balancer.

Node port and cluster IP services, to which the external load balancer routes or automatically created.

* + - 1. ExternalName/ExternalIP: maps the service to the content of external fields by written a C name records within a value.



**Namespaces:**

Namespace helps in creating a virtual cluster inside your Kubernetes cluster.

Namespace helps you isolate your resources between different teams. Allocate resources or apply a quota of resources.

**Advantages of the namespace:**

Helps in easy management of applications of each environment.

Control multiple apps with a single name in a single cluster.

Cost-effective

**Default k8s namespaces are.**

Default: adding an object to a cluster without providing a namespace will place it within the default namespace

Kube-system: kube system namespace is used for k8s components managed by k8S

Kube-public: kube public it is readable by everyone, but the namespace is reserved for system usage.

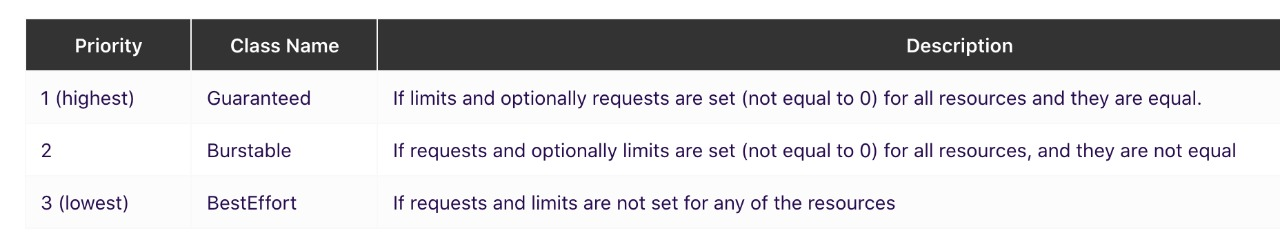
Daemonset: it ensures that all nodes are running exactly one copy of a pod

Daemonset will even create the pod on new nodes that are added to the cluster

Daemonset tells Kubernetes to make sure there is one instance of the pod on nodes in your cluster

**Default limits:**

We can limit the memory and CPU which the container will be using.



**Taint and toleration**: these work together to ensure that pods are not scheduled onto inappropriate nodes.

One or more taints are applied to a node, this marks that the node should not accept any pods that do not tolerate the taints.

Tolerations are applied to pods and allow the pods to schedule onto nodes with matching taints.

Taint and toleration are only meant to restrict the nodes to accept certain pods.

Taint and toleration don’t tell a pod to go to a particular node, instead, it tells the node to accept pods with certain tolerations.

**kubectl taint nodes node1 key1=value1: NoSchedule**

**Node Selector:**

NodeSelector is the simplest recommended form of node selection constraint. NodeSelector is a

field of PodSpec. It specifies a map of key-value pairs. For the pod to be eligible to run on a node, the

node must have each of the indicated key-value pairs as labels (it can have additional labels as well).

Step #1 - First label the node using kubectl command

#kubectl label nodes = #kubectl label nodes node2.example.com size=large

Step #2: Now create a pod definition, specifying the node selector for the pod.

There are limitations. You cannot use the condition that "place the pod on node either "large" or

"medium" but not "small".

You cannot use multiple checks. For that, we use "affinity" and "anti-affinity"

**Affinity:**

For a variety of reasons, a service or container should only run on a specific type of hardware. Maybe

one machine has faster disk speeds, another is running a conflicting application, and yet another is

part of your bare-metal cluster.

A “Hard rule” – Node Affinity Required During Scheduling Ignored during Execution, which means

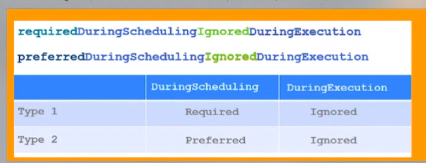
the rule is “required during scheduling” but has no effect on an already-running Pod.

A “Soft rule” – NodeAffinity Preferred during Scheduling Ignored during Execution, which means the

rule is “preferred during scheduling” but likewise has no effect on an already-running pod.

Together, these rules are called NodeAffinity because they indicate a Pod’s “attraction” to certain

Nodes



**ReplicaSet:**

A replicaset is a set of pod templates that describes a set of pod replicas. It uses a template that describes what each pod must contain.

The replicaset ensures that a specified number of pod replicas are running at any time.

We want to replicate containers (and thereby applications) for several reasons, including.

#1 - Reliability: By having multiple versions of an application, you prevent problems if one or more fail. This is particularly true if the system replaces any containers that fail.

#2 - Load balancing: Having multiple versions of a container enables you to easily send traffic to different instances to prevent the overloading of a single instance or node. This is something that Kubernetes does out of the box, making it extremely convenient.

#3 - Scaling: When the load does become too much for the number of existing instances, Kubernetes enables you to easily scale up your application, adding additional instances as needed.

However, a Deployment is a higher-level concept that manages ReplicaSets and provides declarative updates to Pods along with a lot of other useful features.

Therefore, it’s recommended to use Deployments instead of directly using ReplicaSets, unless you require custom update orchestration or don’t require updates at all.

**Deployment:**

A deployment is an object in Kubernetes that lets you manage a set of identical pods. Without a deployment, you’d need to create, update, and delete a bunch of pods manually.

With a deployment, you declare a single object in a YAML file.

This object is responsible for creating the pods, making sure they stay up to date, and ensuring there are enough of them running.

**Rollingupdate:**

Kubernetes deployments rollout pod version updates with a rolling update strategy.

This strategy aims to prevent application downtime by keeping at least some instances up and running at any point in time while performing the updates. Old pods are only shut down after new pods of the new deployment version have started up and become ready to handle the traffic.

MaxSurge: it indicates the maximum number of pods that need to be created before terminating the old pods.

maxUnavailable: this indicates the number of unavailable pods during rolling updates

**Persistent volume:**

Piece of storage in the cluster that has been provisioned by the administrator or dynamically provisioned using storage classes. It is a resource in the cluster just like a node in the cluster resource.

PVs are volumes plugins like volumes but have a life cycle independent of any individual pod that uses the PV.

**K8s persistent volume has the following attributes:**

It is provisioned either dynamically or by an administrator.

Created with a particular filesystem.

Has a particular size.

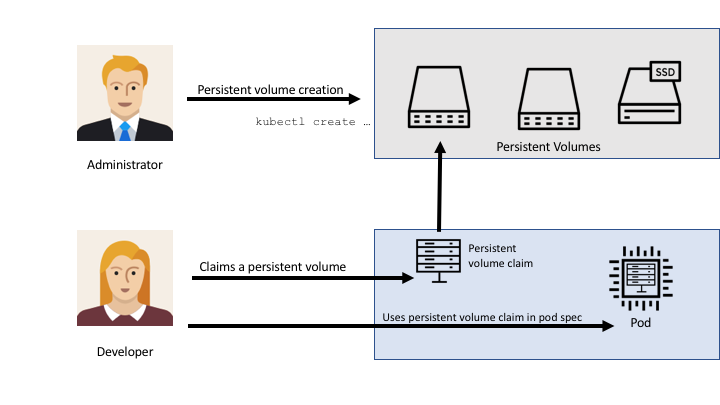
Has identifying characteristics such as volume ids and a name.

**Persistent volume claim (PVC):**

PVC is a request for storage by a user it is similar to Pod. Pods consume node resources and PVCs consume PV resources. Pods can request a specific level of requests (CPU & Memory). Claims can request specific sizes and access modes (for example: they can be mounted once read/write or many times read/write).

Persistent volume claim describes the amount and characteristics of the storage required by the pod, finds any matching persistent volumes, and claims this.

K8s persistent volume remains available outside of the pod life cycle this means that the volume will remain even after the pod is deleted. It is available to claim by another pod if required and the data is retained.



Kubernetes Secrets:

A secrete is an object that contains a small amount of sensitive data such as a password, a token, or a key.

To use a Secret, a Pod needs to reference the Secret. A Secret can be used with a Pod in three ways:

1. As files in a volume mounted on one or more of its containers.
2. As container environment variable.
3. By the kubelet when pulling images for the Pod.

ConfigMaps: Configmaps are similar to secrets.

They can be created and shared in the containers in the same way. The only big difference between them is base64 encoding obfuscation.

Configmaps are intended for non-sensitive data, configuration data like config files, and environment variables and are a great way to create customized running services from generic container images.

**StateFulSet:**

It is introduced from the 1.19 version of k8S

Statefulset is the workload API object used to manage stateful applications

Manages the deployment and scaling of a set of pods and provides guarantees about the ordering and uniqueness of these pods.

Like deployment, a statefulset manages pods that are based on an identical container spec. unlike deployment, a statefulset maintains a sticky identity for each of their pods. These pods are created from the same spec but are not interchangeable. Each has a persistent identifier that it maintains across any rescheduling.

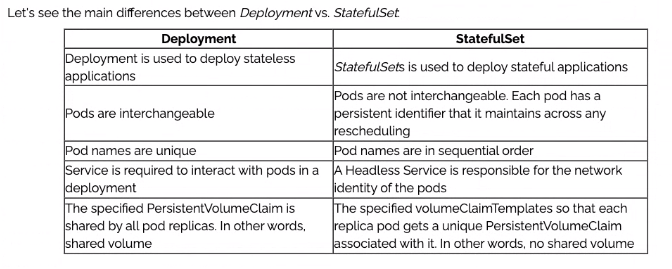
StatefulSets are valuable for applications that require one or more of the following.

Stable, unique network identifiers.

Stable, persistent storage.

Ordered, graceful deployment and scaling.

Ordered, automated rolling updates.



**Kubernetes networking:**

There are 5 essential things to understand about networking in Kubernetes.

Communication between containers on the same pod

Communication between pods on the same node

Communication between pods on different nodes

Communication between pods and services

How does DNS work and how do we discover IP addresses?

**What is a network namespace:**

It is a collection of network interfaces (the connection between two pieces of equipment on a network) and routing tables (instructions for where to send network traffic)

**There is a secret container that runs on every pod in k8s.**

These containers first job is to keep the namespace open in case all the other containers or pods die. Its called pause containers.

Every pod has a unique IP.

This pod IP is shared by all containers in this pod, and its route table from all the other pods.

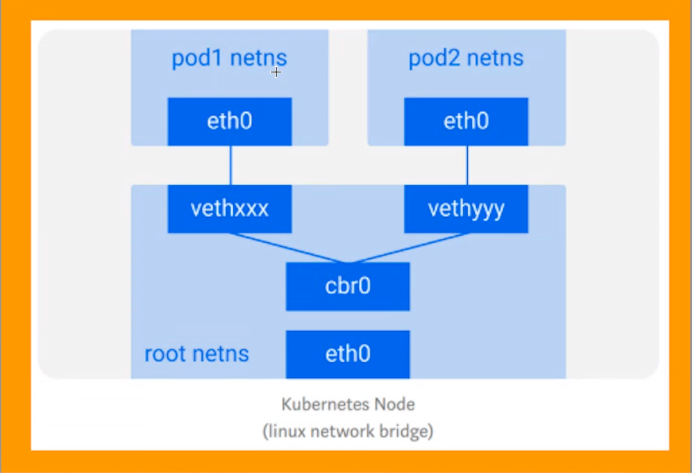
Pause containers are called sandbox containers, whose only job is to reserve and hold a network namespace which is shared by all the containers in a pod. This way, a pod IP doesn’t change a container dies and a new container is created in a space.

A huge benefits of the IP per pod is there are no IP of port collision with the underlying ports and we don’t have to worry about what port the application use.

**Communication between containers on same pod (intranode communication)**

On every k8s node, which is a Linux machine, there is a root network namespace- root netns. The main network interfaces eth0 is root netns.

Similarly, each pod has its own netns with a virtual ethernet pair connecting it to the root netns.



Assume a packet is going from pod1 to pod2.

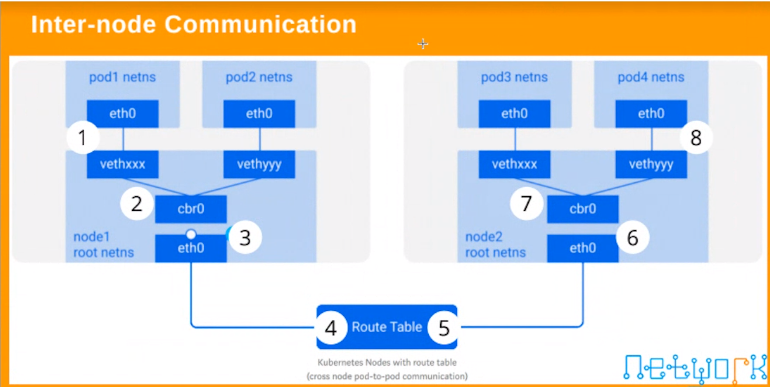
1. It leaves pod1’s netns at eth0 and enters the root netns at vethxxx.

2. It’s passed on to cbr0, which discovers the destination using an ARP request, saying “Who has this IP?”

3. vethyyy says it has that IP, so the bridge knows where to forward the packet.

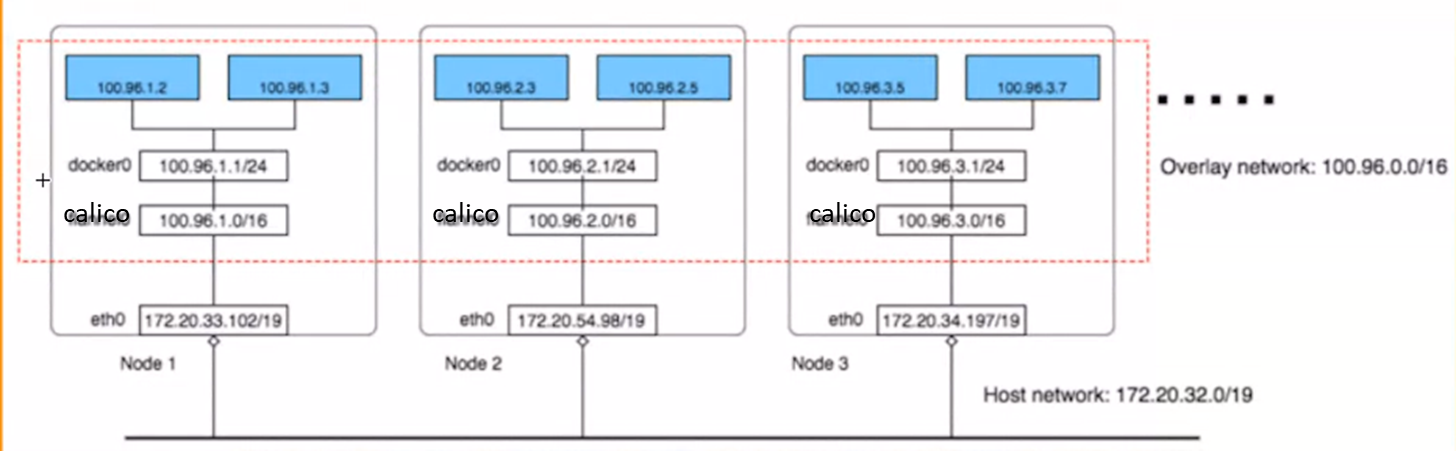
4. The packet reaches vethyyy, crosses the pipe-pair, and reaches pod2’s netns.

**Inter-node communication:**



**Communication between pods and services:**

Overlay network:



Kubernetes does not provide any default network implementation, rather it only defines the model and leaves it to other tools to implement it.

There are many implementations nowadays, calico is one of them and one of the simplest.

Overlay networks are not required by default, however, they help in specific situations. Like when we don’t have enough IP space, or the network can’t handle the extra routes, or when we want some extra management features the overlays provide.

There are three networks in this cluster:

**Host network:** all instances are in one VPC subnet 172.20.32.0/19. They have been assigned IP addresses in this range; all hosts can connect to each other because they are on the same LAN.

**Flannel/calico overlay network:** flannel has created another network 100.96.0.0/16, it’s a bigger network that can hold 216(65536) addresses, and it’s across all Kubernetes nodes, each pod will be assigned one address in this range, later we will see how flannel achieves this.

**In-Host docker network:** inside each host, flannel assigned a 100.96.x.0/24 network to all pods in this host, it can hold 28(256) addresses. The docker bridge interface docker0 will use this network to create new containers.

**Liveliness**

Liveness and Readiness probes are used to control the health of an application running inside a Pod’s container. Both are very similar in functionality and usage.

**Readiness probe:**

In some cases, we would like our applications to be alive but not serve the traffic unless some conditions are met. In such cases we use a readiness probe if the condition inside the readiness probe passes, then only our applications can serve the traffic.

The readiness probe runs on the container during its whole life cycle.

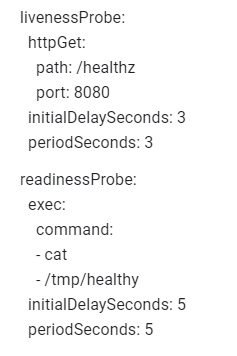
Note: liveliness probe doesn’t wait for the readiness probe to succeed. If you want to wait before executing a liveliness probe you should use an initial delay second.

initialDelaySeconds: 3

This is the delay which tells kubelet to wait for 3 seconds before performing the first probe

periodSeconds: 5

This field specifies that kubelet should perform a probe every 5 seconds.

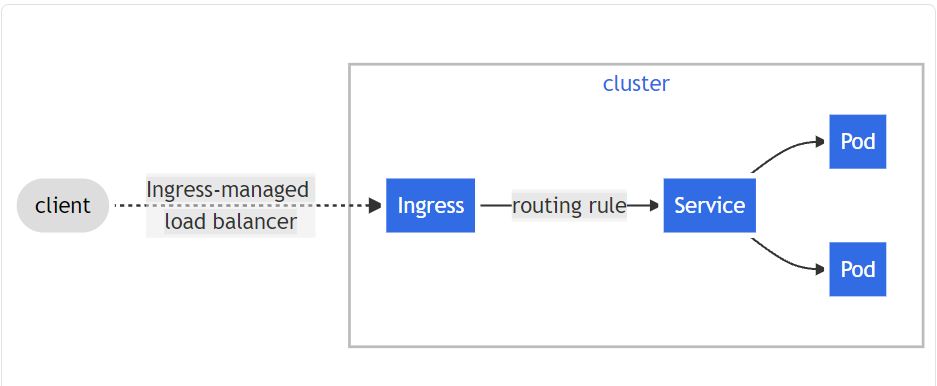


Both liveliness and readiness probe are used to control the health of the application. Failing liveliness probe will restart the container, whereas failing readiness probe will stop our application from serving the traffic.

**ECR (Elastic container registry)- it’s a AWS Service**

**What is Ingress?**

Ingress exposes HTTP and HTTPS routes from outside the cluster to services within the cluster. Traffic routing is controlled by rules defined on the Ingress resource.



An Ingress does not expose arbitrary ports or protocols. Exposing services other than HTTP and HTTPS to the internet typically uses a service of type Service.Type=NodePort or Service.Type=LoadBalancer.

**Helm:**

In simple terms, the helm is a package manager for Kubernetes. Helm is the k8s equivalent of yum or apt.

Helm deploys charts, which you can think of as a packaged application. It is a collection of all your versioned, preconfigured application resources that can be deployed as one unit.

Helm helps in three keyways:

* + - 1. Improves productivity.
      2. Reduces the complexity of deployment of microservices.
      3. Enables the adoption of cloud-native applications.

Helm charts are simply k8s yaml manifest combined into a single package that can be advertised to your k8s cluster.

Helm -There are 3 concepts we need to get familiar with

Chart: a package of preconfigured k8s resources.

Release: a specific instance of a chart that has been deployed to the cluster using helm.

Repository: a group of published charts that can be made available to others.

How to create heml: helm create (applicationname)

Chart. yaml: this is where you put all the information about the chart of your packaging. So, for example, your version number, name of the application, etc. is where you put all the details.

Charts: this is where you store other charts that your charts depend on. You might be calling another chart that your chart needs to function properly.

Templates: this folder is where you put the actual manifest (yam file) you are deploying with the chart. For example, you might be deploying nginx deployment that needs a service, config map, and secrets. You will have your deployment. yaml, service.yaml, and secret.yaml are all in the template directory this will all get their values from values.yaml file.

Values.yaml: this is where you define all the values you want to inject into your templates.