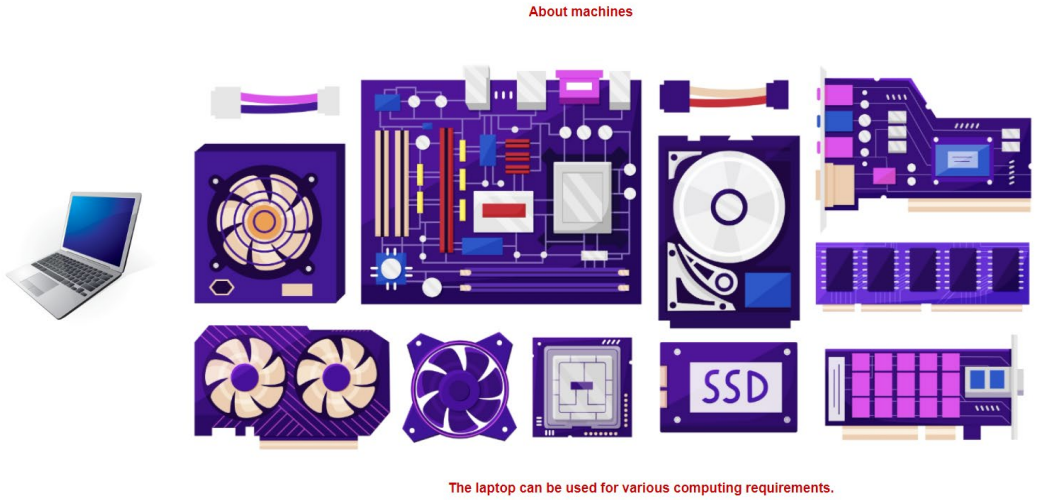
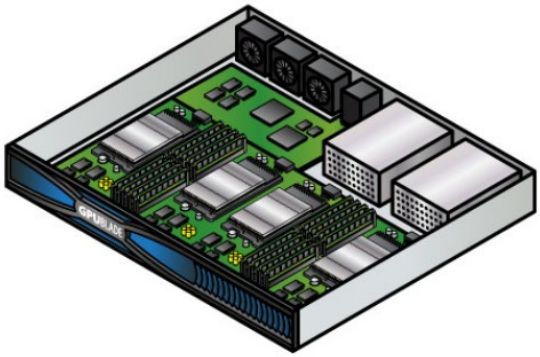
Introduction

Let's build our foundation- About machines



The laptop can be used for various computing requirements.

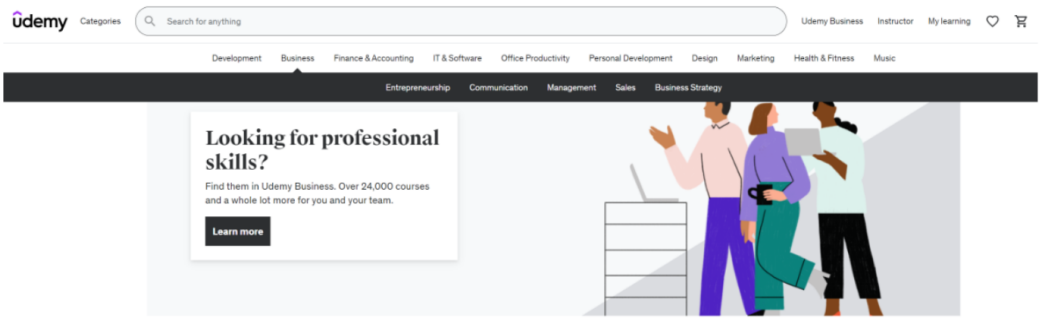


Server machines - These have more computing resources and memory for hosting applications.

Let's build our foundation- About applications

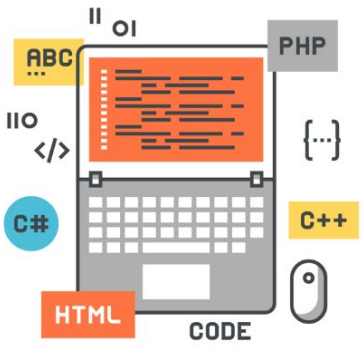
Applications

There are different types of applications - You have desktop-based , web-based, mobile, gaming applications.



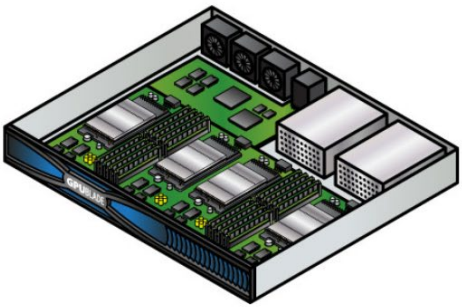
This is a web application that is used by users across the world

Its used to distribute video content in the form of online courses



PROGRAMMING LANGUAGE

The application needs to be developed



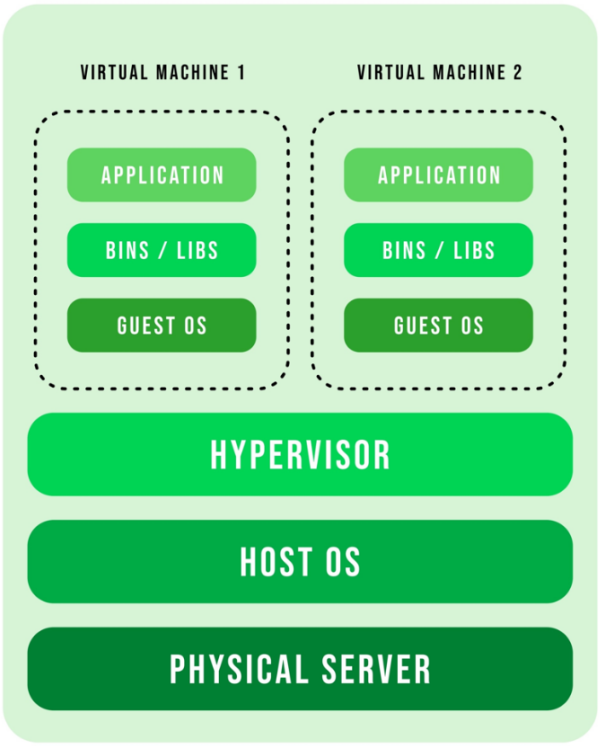
This application needs to be hosted somewhere

We are focusing on the hosting aspect for the application

The advent of virtual machine

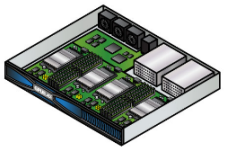
Virtual Machines

These are virtual computers that run on physical machines



VMware / Hyper-V

Linux or Windows



Each virtual machine can make use of resources such as CPU/Memory from the underlying physical server.

Each virtual machine is isolated from each other.

The advent of containers

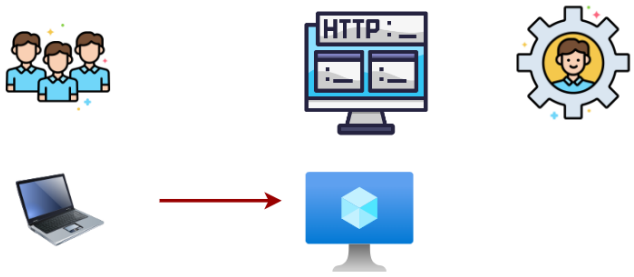
Containers

Having virtual machines was a big breakthrough

Companies could host multiple virtual machines on a physical server and make use of the server.

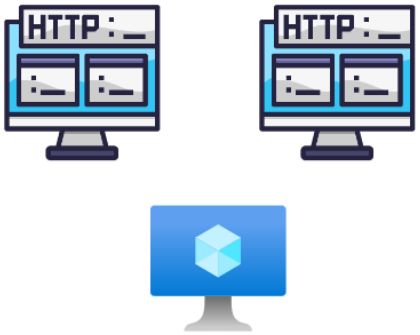
But then there were issues when it came to deploying applications.

Developers develop the application on their machines



When the application is deployed to a virtual machine it does not work as intended.

This could be because of differences in machine software configuration, libraries not present etc.



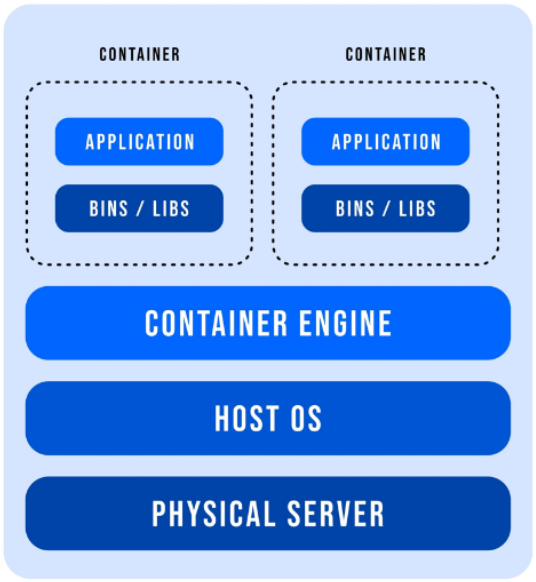
You have 2 applications on the same machine.

One application update requires a library/component to be installed.

This causes the other application to stop working.

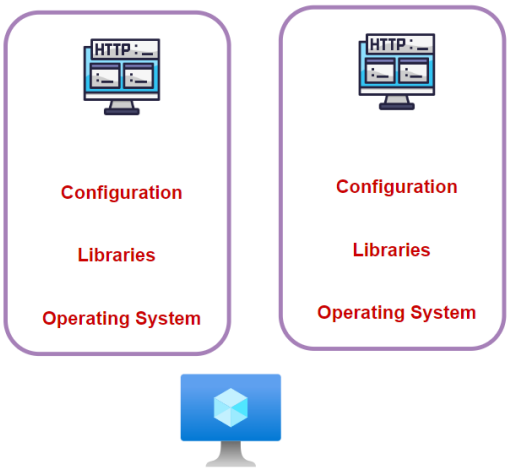
Welcome to containers

This is a unit of software that packages up all the code and dependencies that are required for the application to run.



The underlying container will have a light-weight operating system, the application, libraries etc.

Each container runs in isolation



Ubuntu 20.04

Ubuntu 22.04

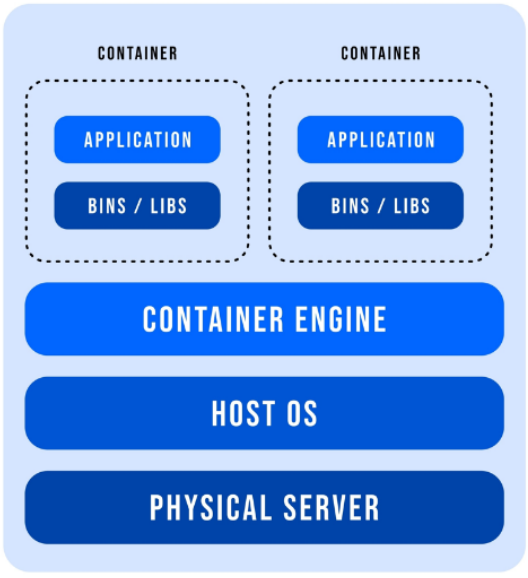
Ubuntu 22.04

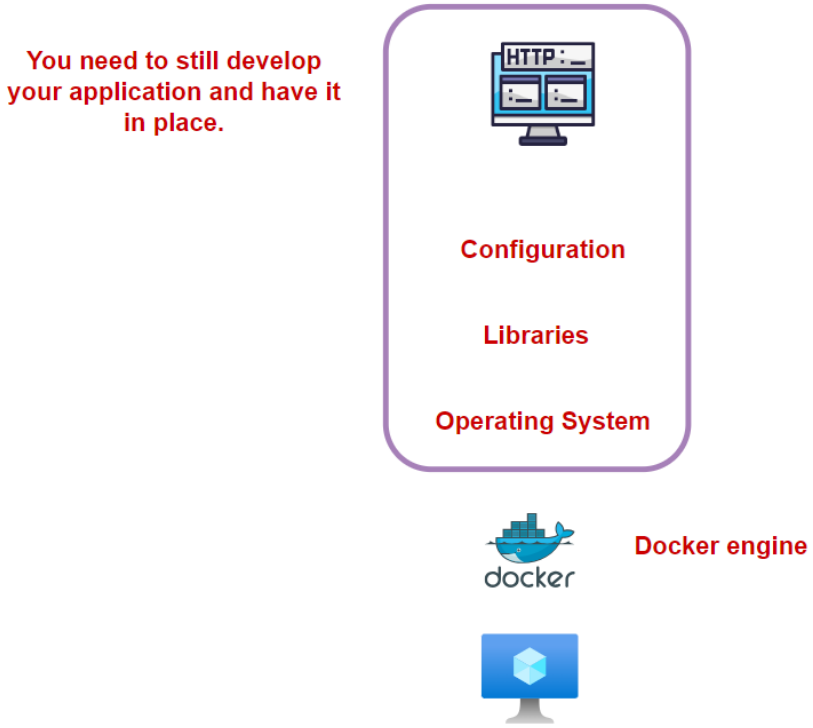
What is Docker

Docker

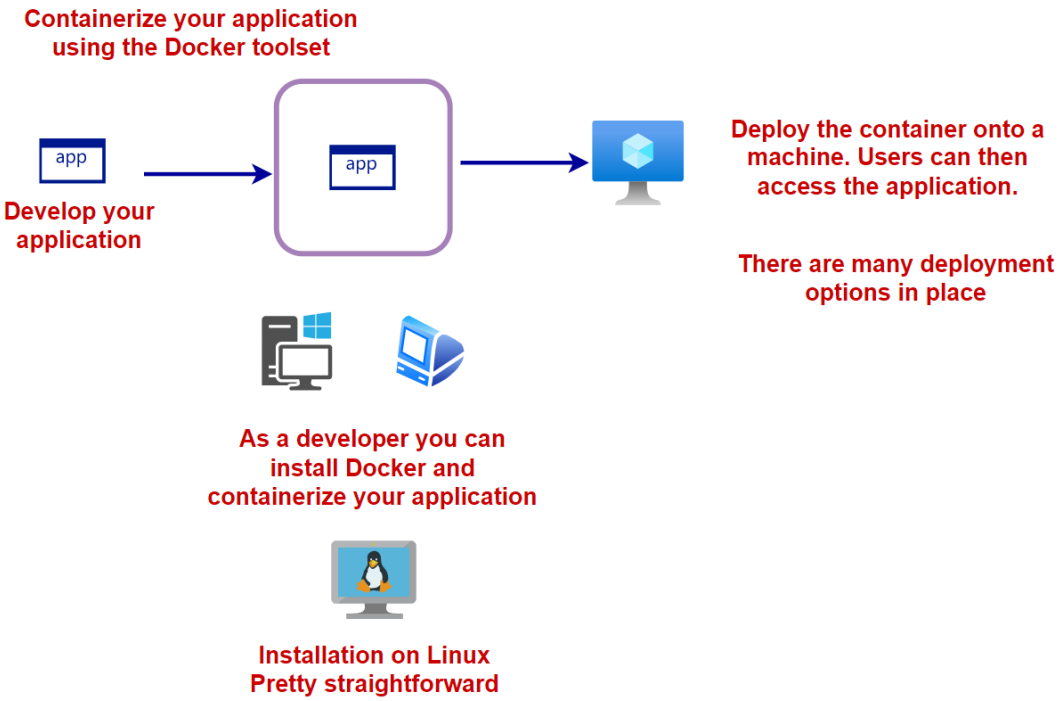
This is a platform that can be used to develop, ship and run applications as containers.

Here the container engine is the Docker toolset





Docker Installation



Just issue commands to install Docker

Windows machine

Install Docker Desktop - This will include the Docker runtime and required tools.

To run Windows-based containers you need to use Windows 10 or 11 Professional or Enterprise Edition.

We also need to use WSL (Windows subsystem for Linux) - This allows us to run Linux systems on Windows.

MacOS machine

We need to install Docker Desktop. This is supported for both machines that are built on Intel or Apple Silicon chips.

Docker- Images and Containers

Understanding what we implemented in the prior chapter

Docker Desktop

docker run -d -p 80:80 nginx

Here we ran an image based on the NGINX web server and ran it as a container.

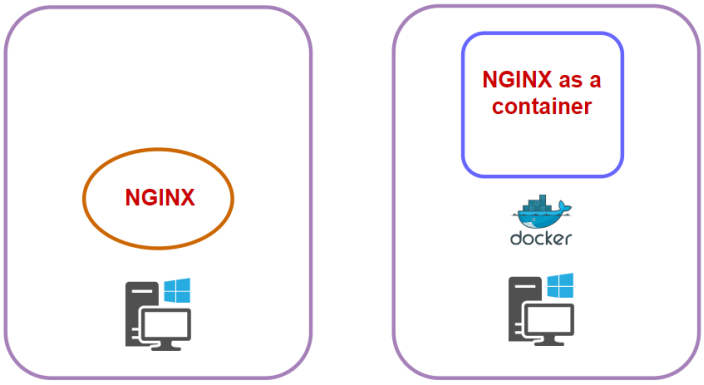
What is an image?

This is a read-only template that has instructions on how to create the Docker container

Browser
http://localhost

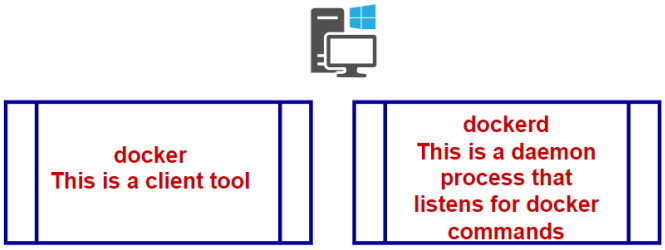
On my machine I can install NGINX. This is a web server.

But we are here to learn about Docker and containers.



What is the docker run command

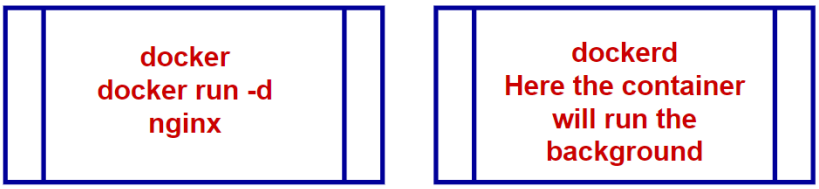
This is used to run containers



The dockerd process can manage Docker images, containers, networks and volumes.

All of this is available when we install Docker Desktop.

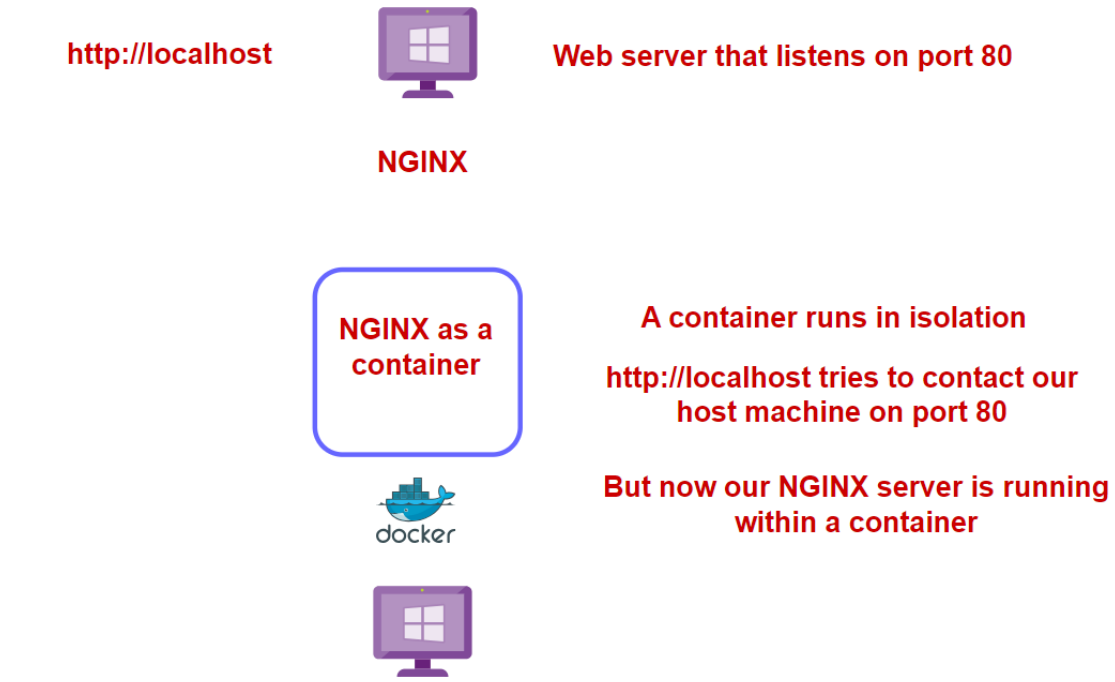
Running the container in detach mode



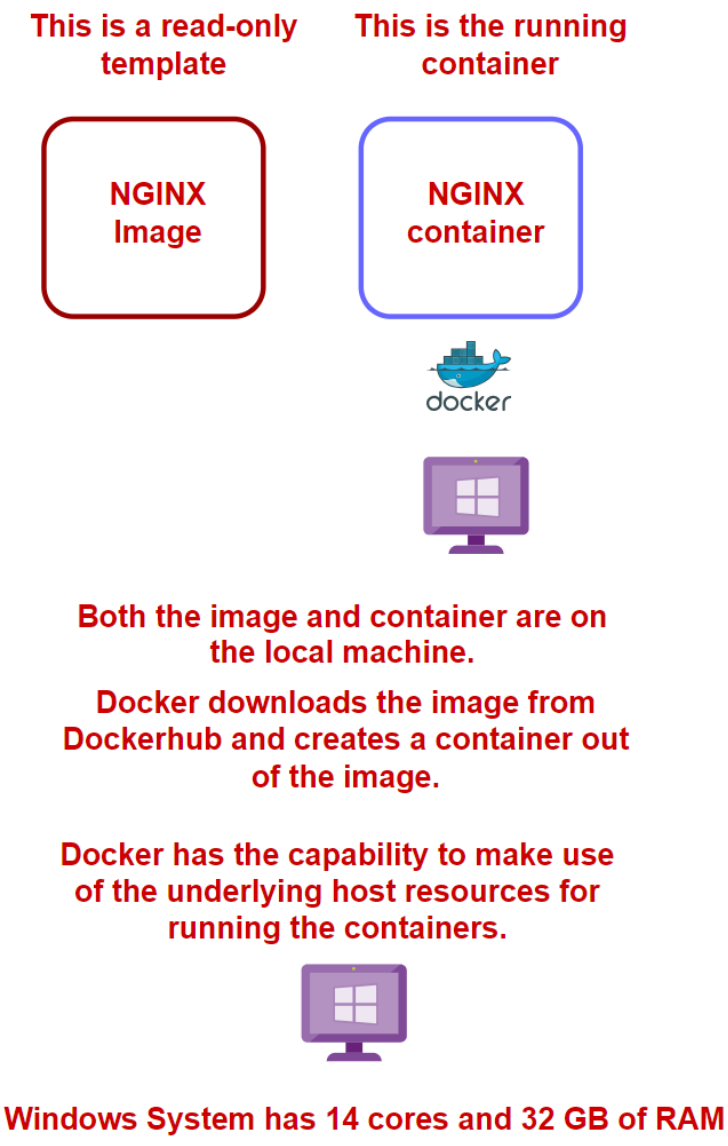
`--cpu-rt-runtime`		API 1.25+ Limit CPU real-time runtime in microseconds
`--cpu-shares`	`-c`	CPU shares (relative weight)
`--cpus`		API 1.25+ Number of CPUs
`--cpuset-cpus`		CPUs in which to allow execution (0-3, 0,1)
`--cpuset-mems`		MEMs in which to allow execution (0-3, 0,1)
`--detach`	`-d`	Run container in background and print container ID
`--detach-keys`		Override the key sequence for detaching a container
`--device`		Add a host device to the container

https://docs.docker.com/engine/reference/commandline/run/

Publishing the container port number



Docker images



Let's say that you want to host the NGINX web server on a Linux system.

First install Ubuntu 22.04 as an example - There is a requirement for having 25 GB of recommended storage for installing Ubuntu.

The NGINX server needs to run on an underlying operating system.

The operating system comes as part of the image and hence as part of the package.

Does this mean our system needs to have 25 GB of storage for each and every image and container?

NGINX Image

NGINX container

Name	Tag	Status	Created	Size
nginx 593aee2afb64	latest	In use	7 days ago	186.77 MB

NGINX web server

Some additional libraries

Light-weight distribution of Linux. Just has the necessary system libraries

The Docker image consists of many layers stacked on top of each other.

Docker ensures that all layers work together.

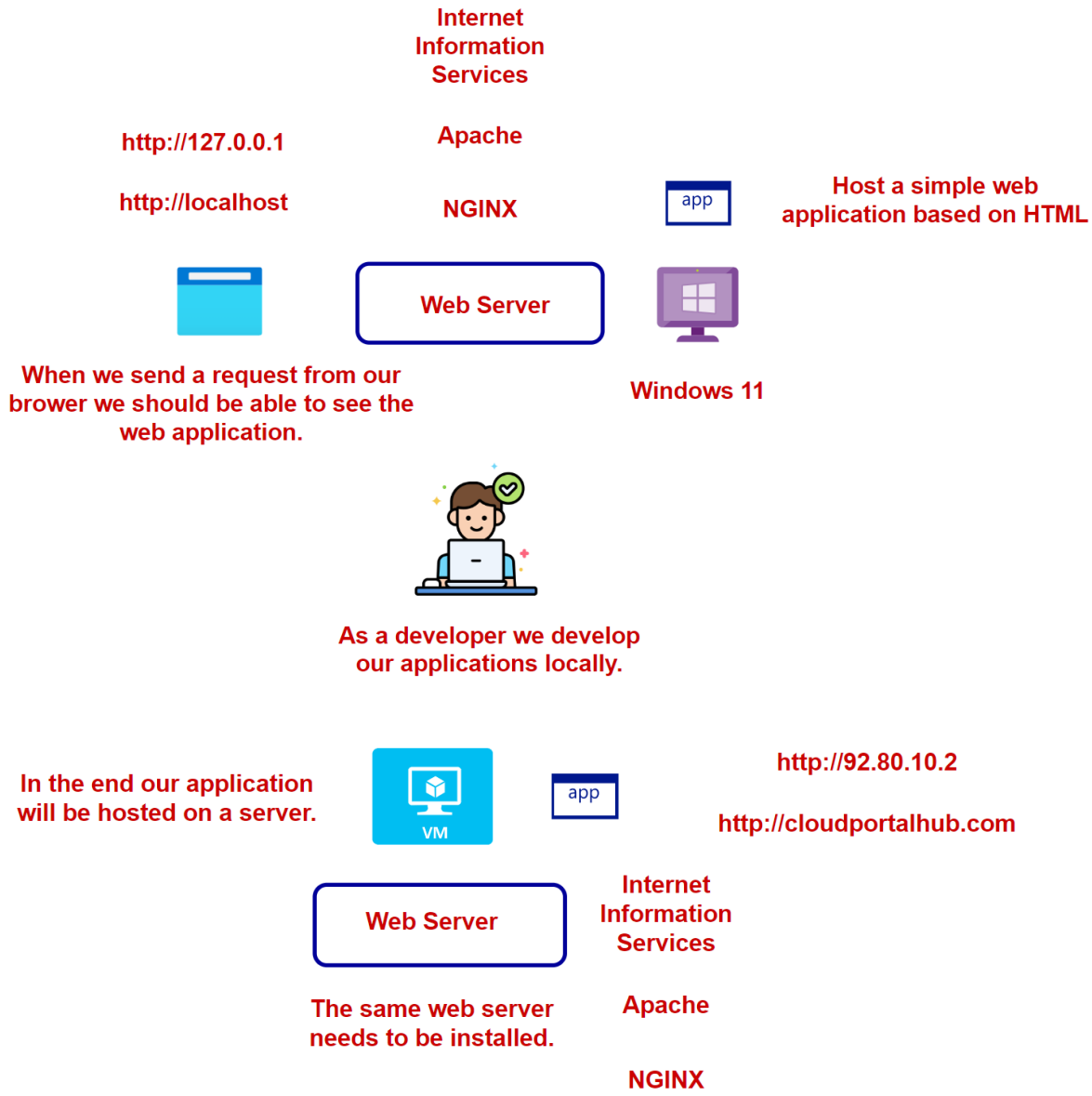
Docker runs the image as a container

NGINX web server

Some additional libraries

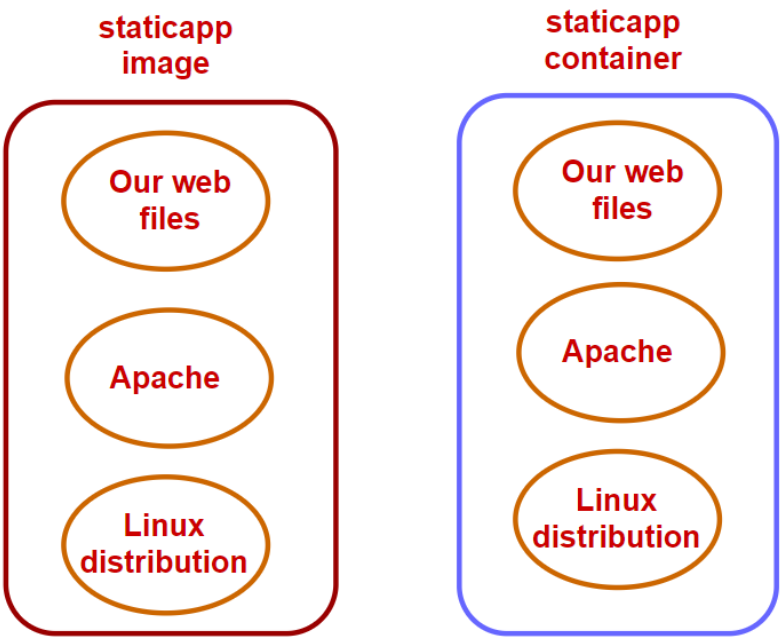
Light-weight distribution of Linux. Just has the necessary system libraries

About hosting web applications



Docker- Building our own custom images

About tagging Docker images



You can add an additional alias or tag for your image.

This is ideal when you have different versions for your images.

An image is referenced via the below format

[HOST[:PORT_NUMBER]/]PATH

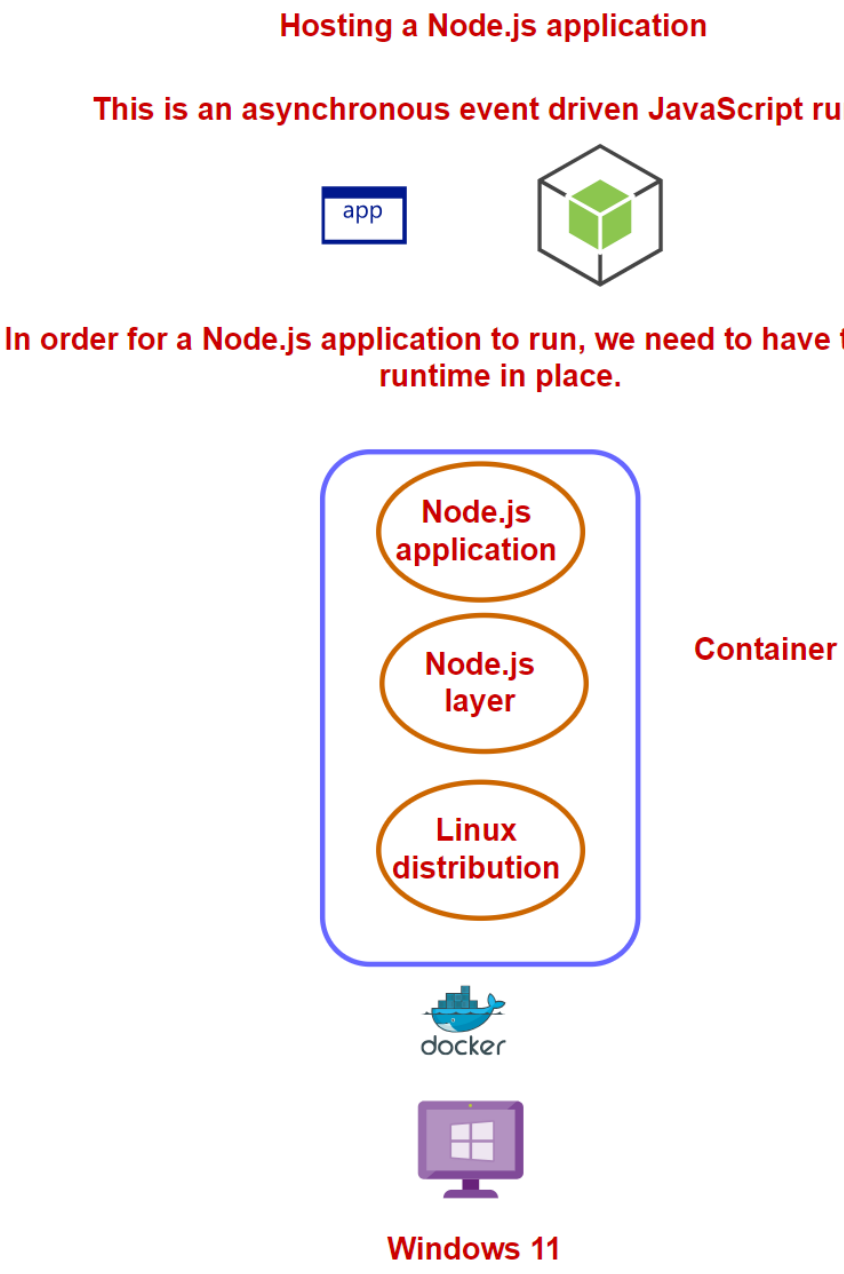
The host and port number is for the registry that is holding the images.

The default host is registry-1.docker.io

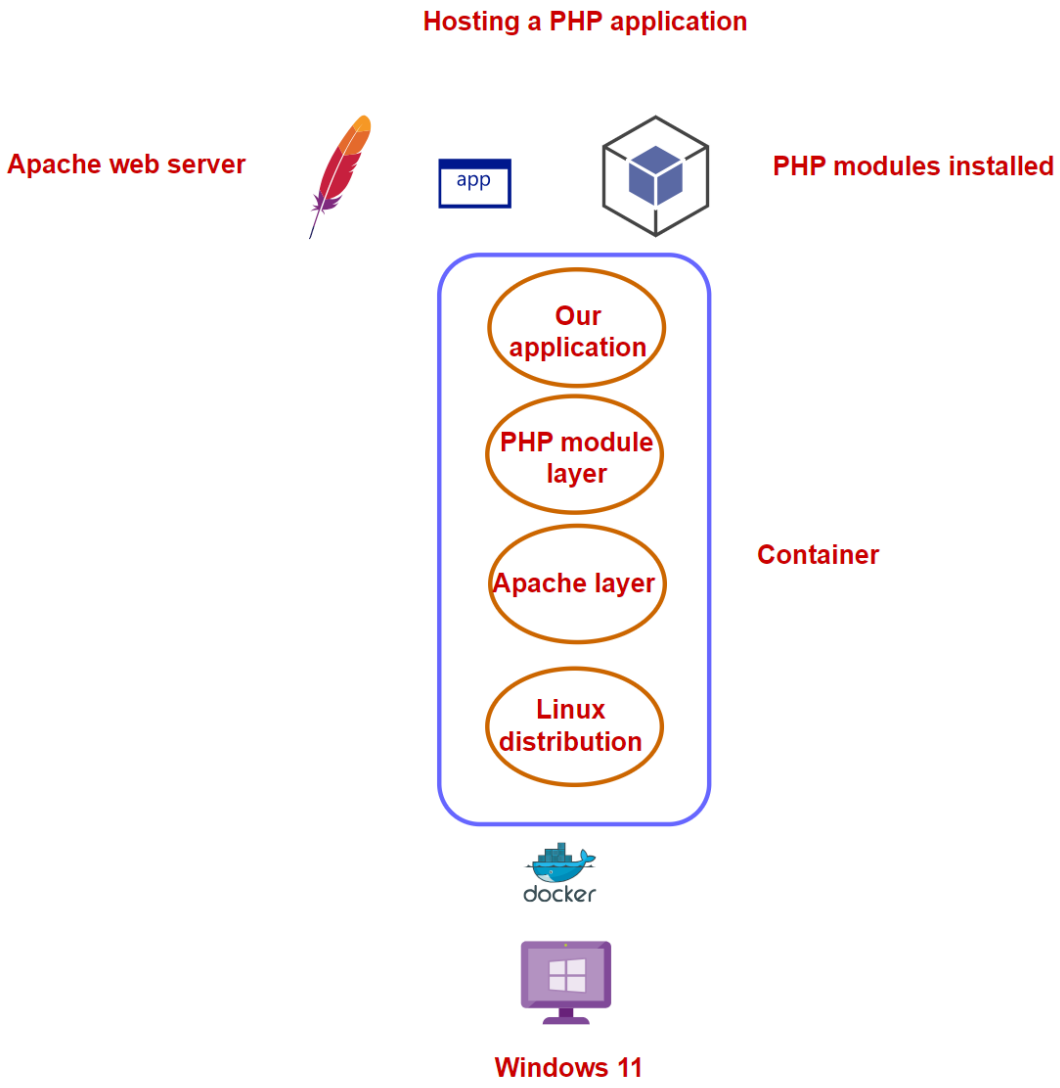
This is the public Docker registry

The path is the name of the image and the optional tag.

Using a Node.js app

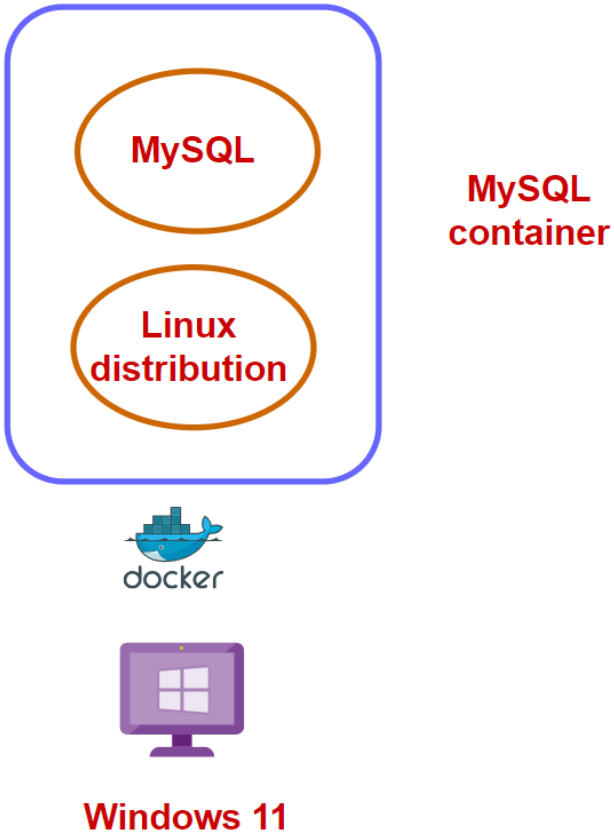


Lab- Using a PHP application

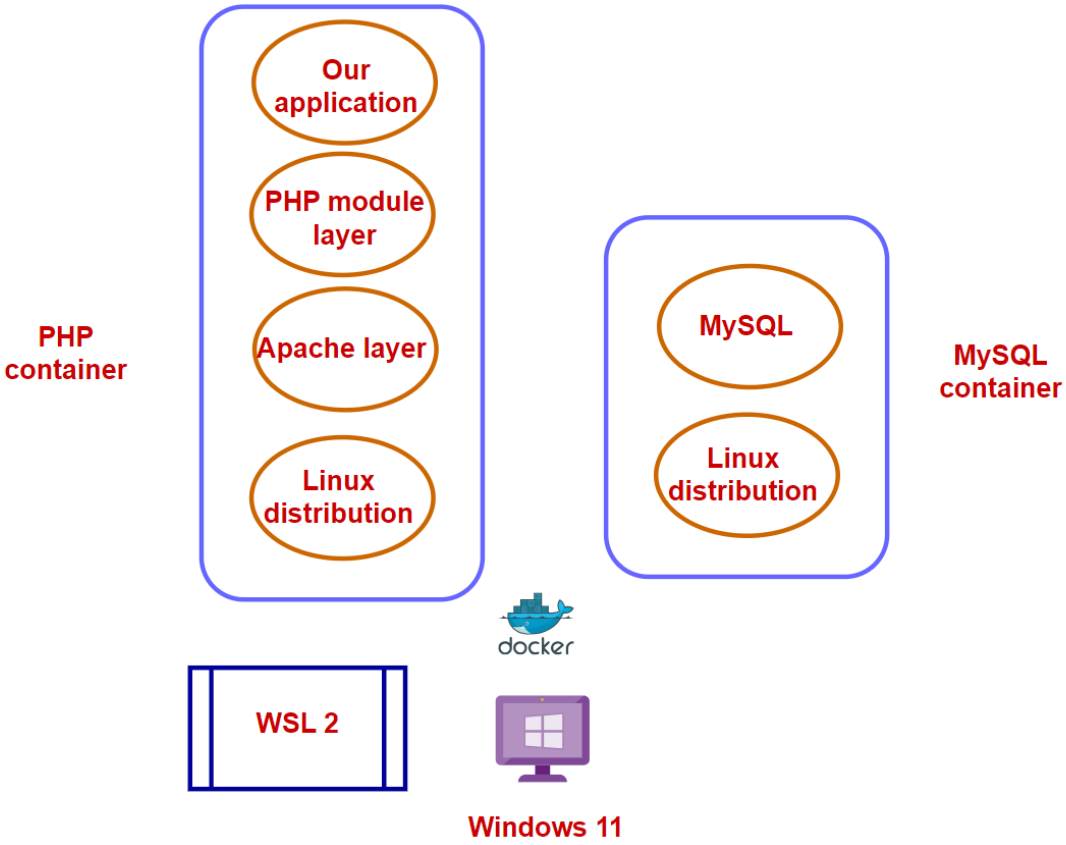


Lab- Now let's look at a MySQL container

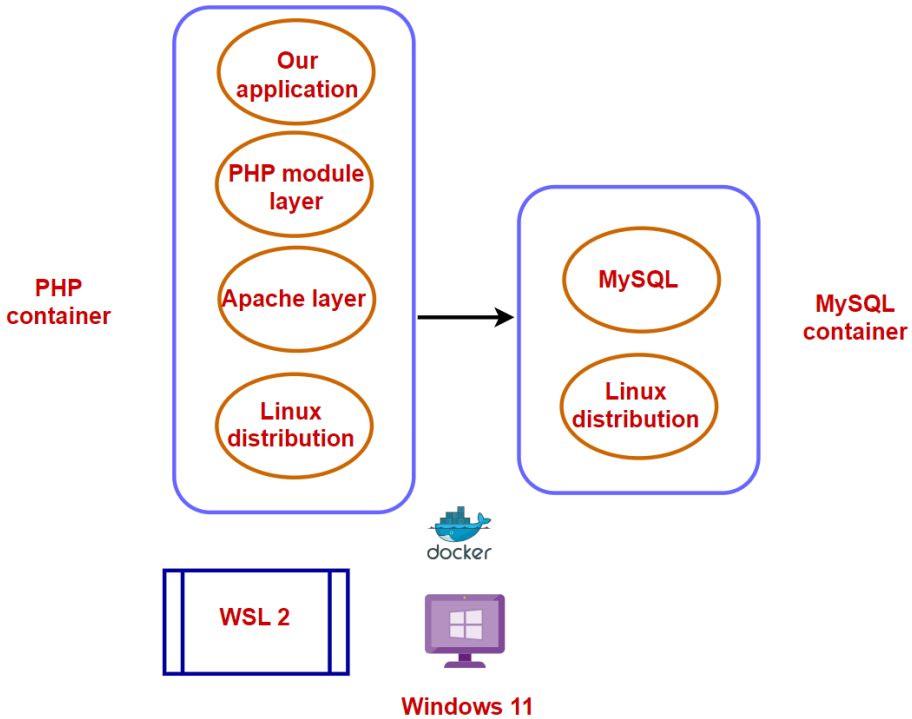
Normally a web application needs to work with a database for storage of data.



What have we done so far


Our application
PHP module layer
Apache layer
Linux distribution
PHP container
MySQL
Linux distribution
MySQL container
docker
WSL 2
Windows 11

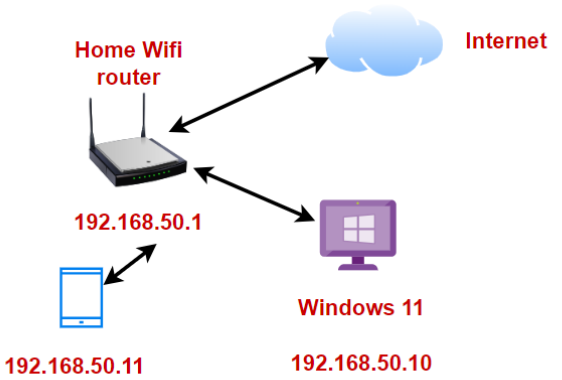
For both containers we have exposed the ports to the host machine so that we can connect to the services hosted in the containers.



But now we want our containers to talk to each other. We want our PHP application to fetch data from the underlying MySQL database hosted in the MySQL container.

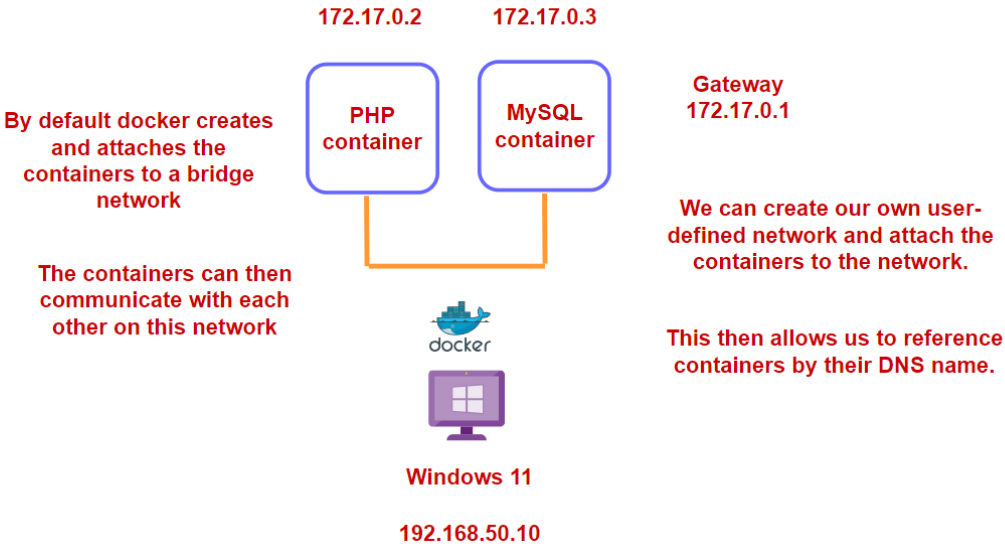
Docker- Networking and data

Understanding Networking

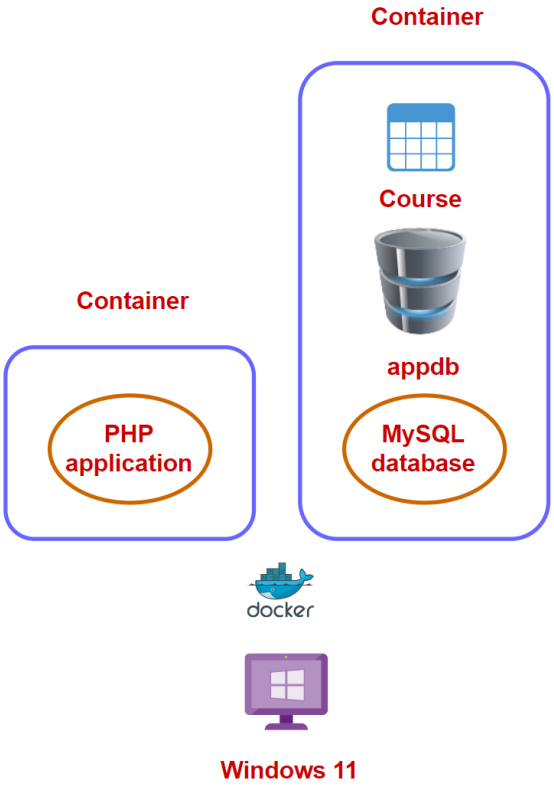


The IP address helps to uniquely identify the machine on the network.

IP addresses also help to identify devices on the Internet.



Lab- Creating a custom MySQL image



1. We created a MySQL user and gave permissions.

2. We created a database named appdb.

3. We created a table with data within the table.

If we stop and start the container, the data remains as it is. It's within the container.

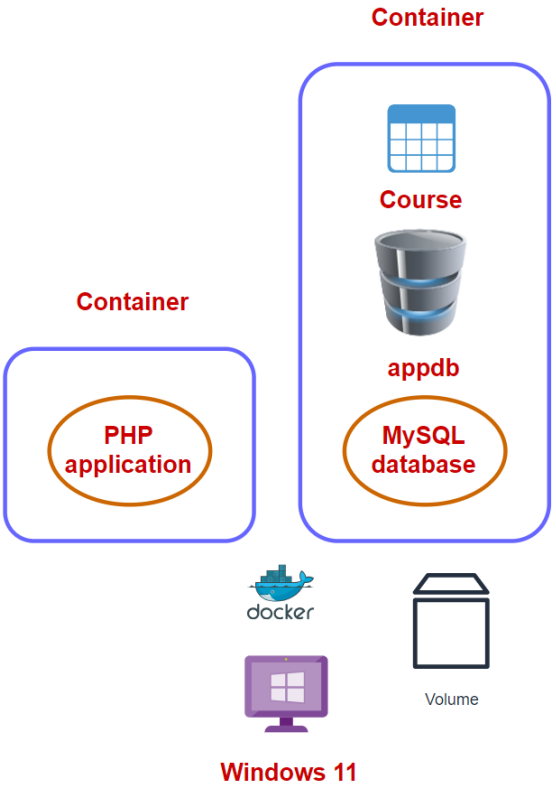
But if we delete the container and try to create the container again, we will have no data.

One method is to seed the data that we need at startup of the container.

We will create a SQL script named 01.sql and place all required SQL commands in that script.

The MySQL container has a folder named docker-entrypoint-initdb.d. All scripts placed inside this container will executed when the container is launched.

Lab- Using volumes in Docker



Right, so we seeded our database and table.

What happens if we add a new record to the table. And then we remove the container and run the container again, what happens.

We need some way of persisting our data.

We can create a volume and attach it to the container.

Data can be stored on the volume.

Even if the container is removed, the volume will still be intact.

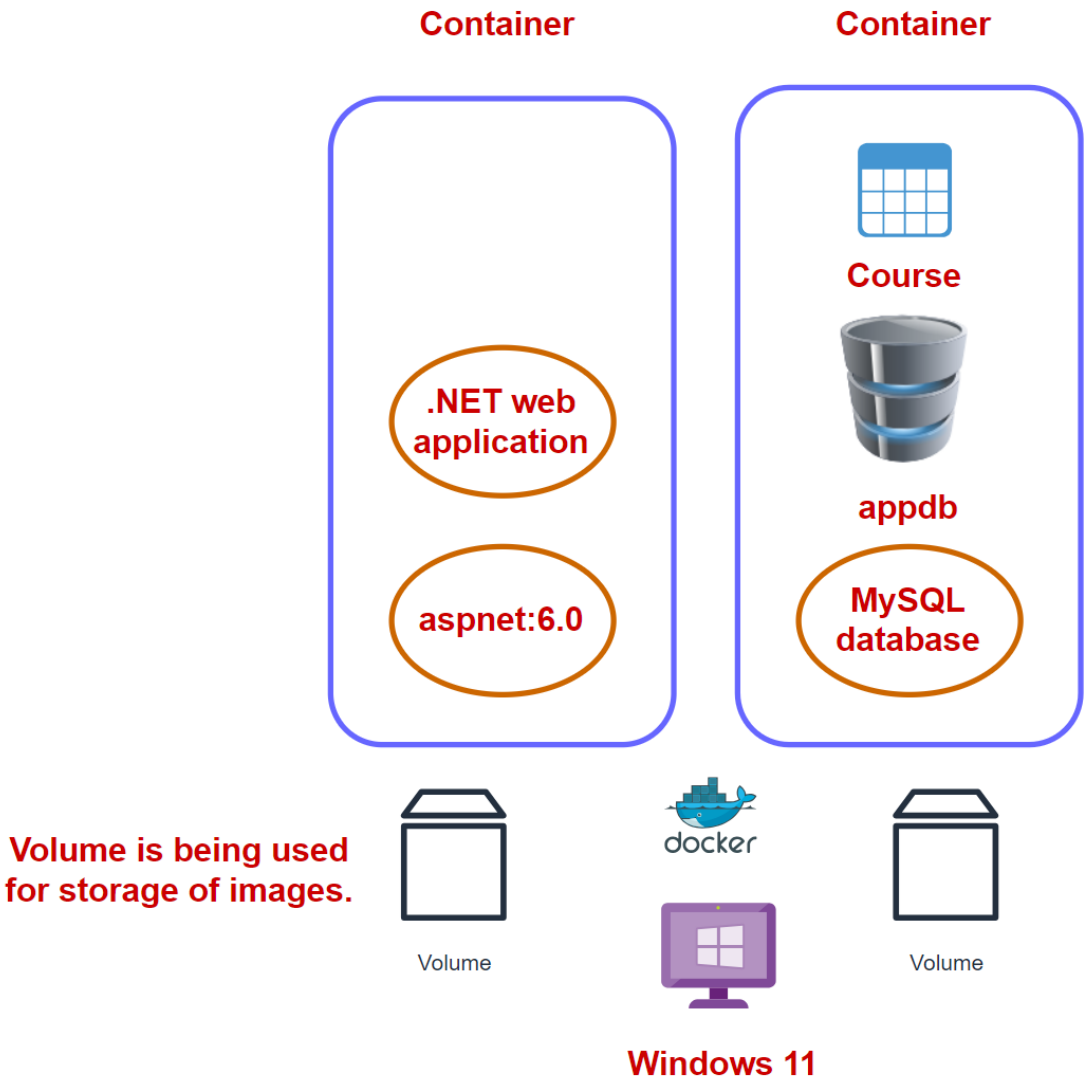
<u>Benefits of volumes</u>

Can be used for both Linux and Windows containers.

It can be shared across multiple containers.

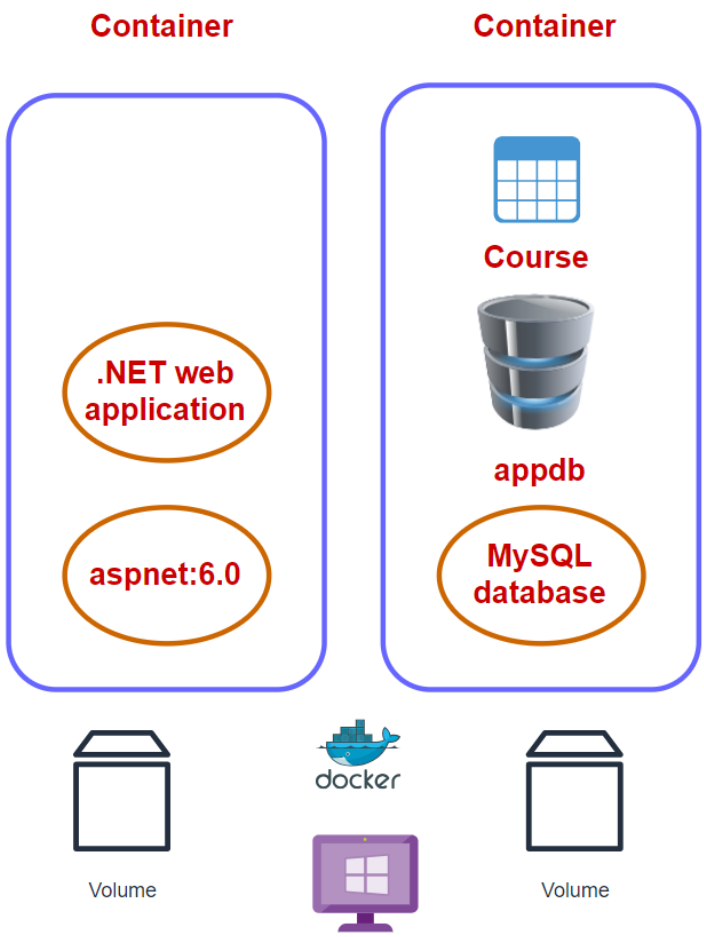
You can also have your volumes on remote locations such as the cloud.

Understanding what we implemented



Docker compose and Docker hub

Docker Compose



Now its great that we were able to have multiple containers talking to each other.

But normally you might have more containers. You might have one for the business logic layer, one seperately for logging etc.

Using Docker run each and every time you want an environment up and running can be tedious.

Welcome to Docker compose

With a configuration file you can specify all the services that need to run as part of your environment.

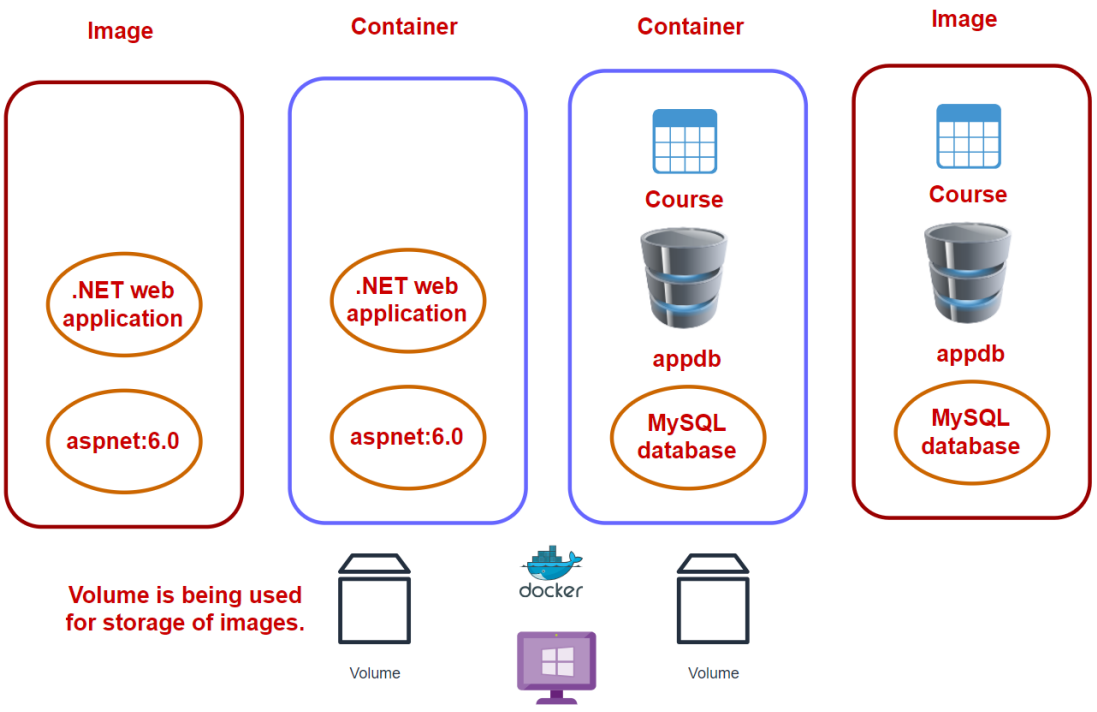
With a single command, all services can be brought up in an instant.

This is great when you want to bring up services on different environments.

Docker Desktop already has Docker Compose built in.

You can also install Docker Compose as a standalone unit on Linux and Windows servers.

Using Docker hub



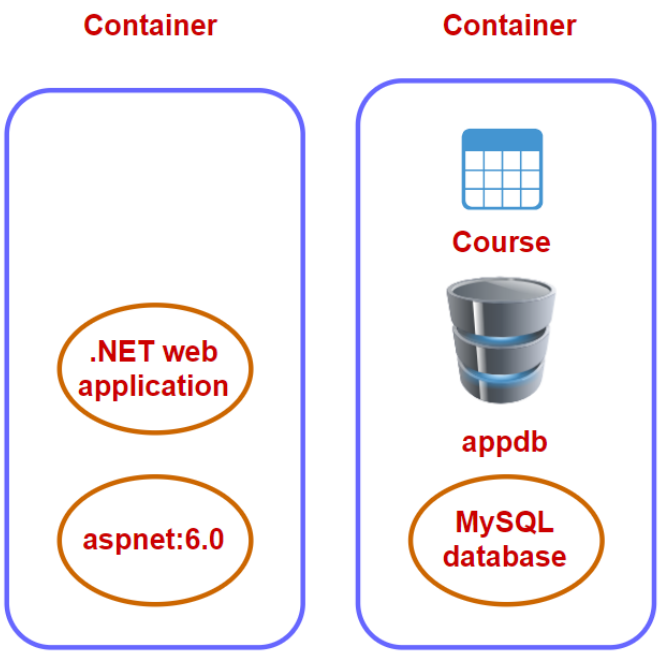
So far we have seen how to work with images and containers on our local machine. We are making use of Docker Desktop.

But the entire purpose of Docker is to have the ability to ship our applications via the use of containers.

We should be able to run our containers on other machines.

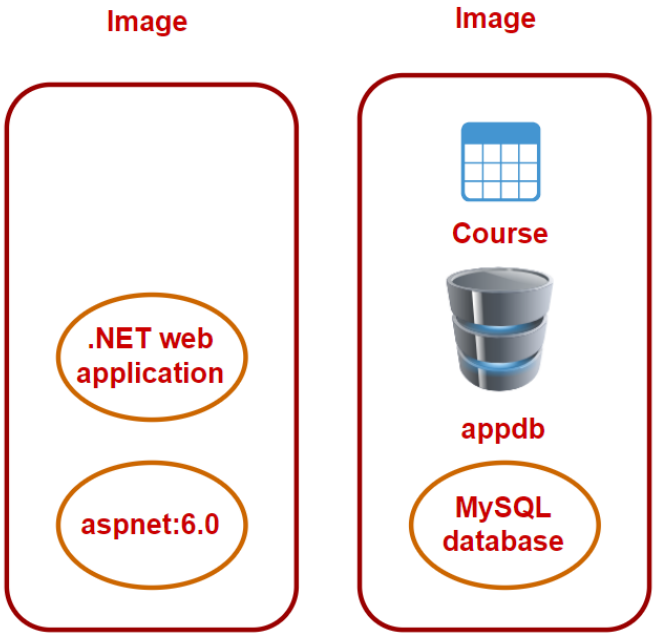
We could copy our images onto remote machines which has Docker installed.

But that is an inefficient process and is an issue in terms of security and maintainability.



We can publish our images onto a public repository such as Docker hub

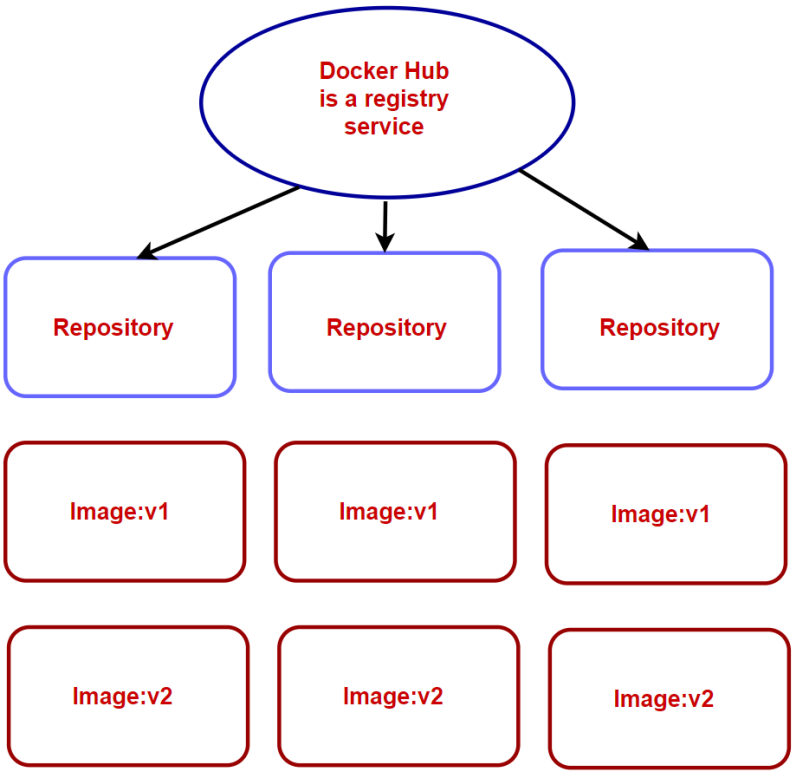
And then use Docker to run containers out of those images.



In the registry service you can create multiple repositories

A repository can be a public or private repository.

Each repository has images of different versions



Docker compose- Safeguarding secrets via files

docker-compose.yaml

```yaml
version: "3.8"
services:
  db:
    image: mysql-image:latest
    container_name: mysql-instance
    networks:
      - app-network
    volumes:
      - type: volume
        source: mysql-data
        target: /var/lib/mysql
    ports:
      - "3306:3306"
    environment:
      - MYSQL_ROOT_PASSWORD=sqlset1010
networks:
  app-network:
```

Its not a good practice to embed passwords in clear text in files.

We can make use of Docker secrets to make use of sensitive information such as our passwords.

To make use of Docker secrets, we need to make use of Docker swarm.

Docker swarm allows you to host multiple Docker hosts in a cluster.

You can then schedule the running of containers on the swarm cluster.

Why would we consider using multiple host machines for our containers.

Well lets say that one container is launched on one Docker node and that node crashes. Your container is no longer available.

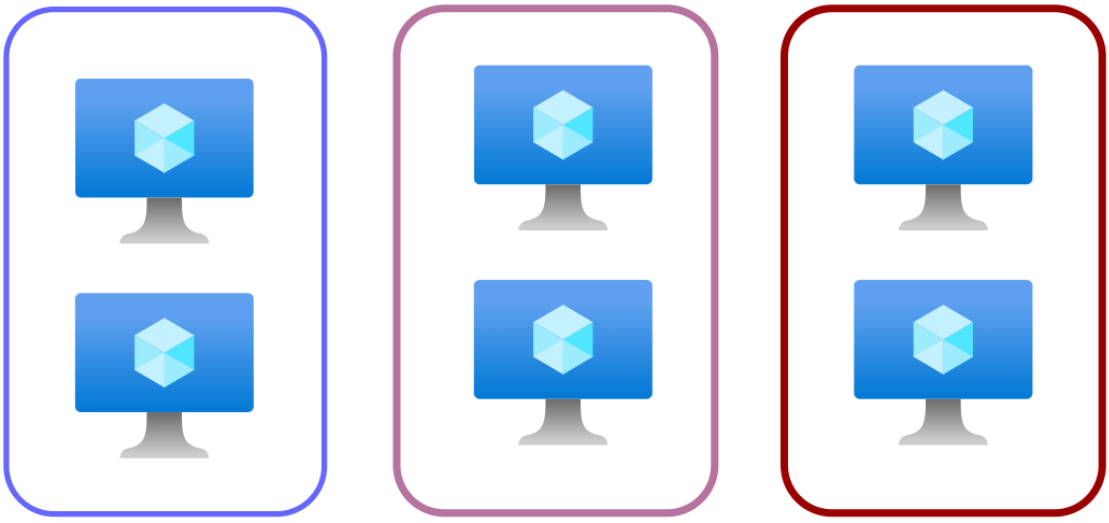
The Docker swarm cluster can detect that the node has crashed and bring up the container on another node on the Docker swarm cluster.

Docker compose- Another use of environment variables

Windows 11

As a developer you develop applications and package them into images to be run as Docker containers.

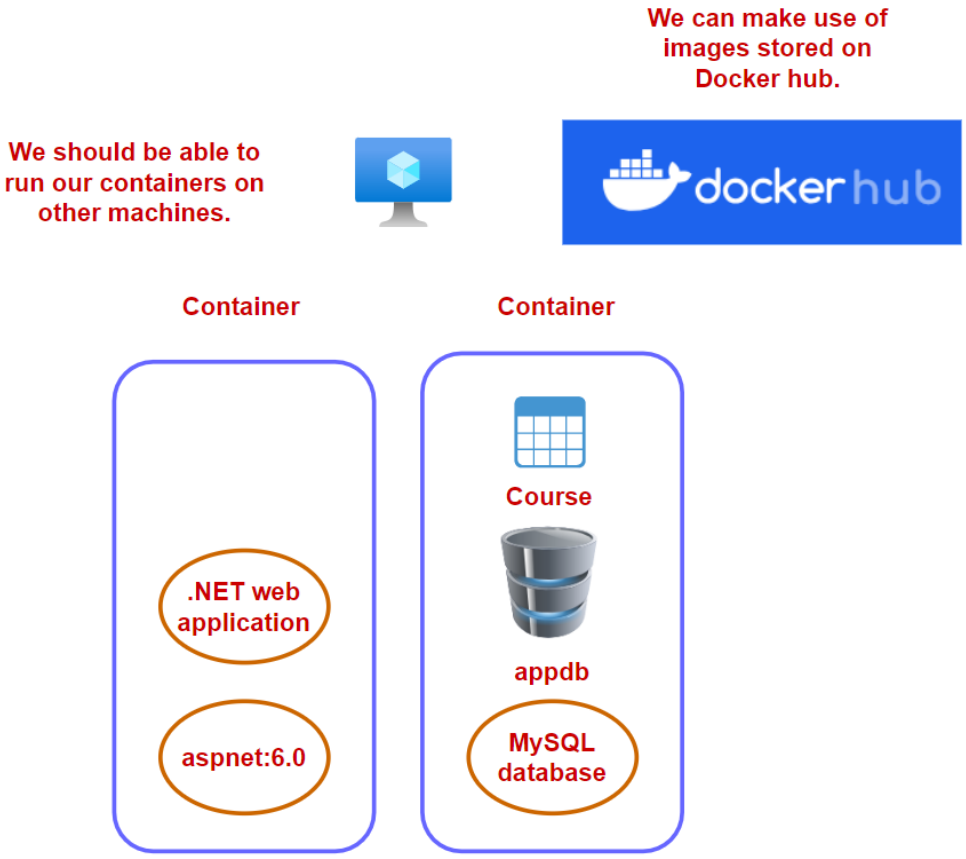
But then the application needs to be deployed onto servers



Development **Staging** **Production**

Docker- Exploring cloud platforms

Benefits of cloud platforms



Lab- Azure- Creating a Virtual Machine

Developer will create the application and package it as Docker images.

The images can be published to Docker hub

Staging server

Production server

Host the application via containers

Normally a company would need to invest in hardware - Buy servers, storage. Setup networking , cooling, a data center for many servers.

There are several costs involved.

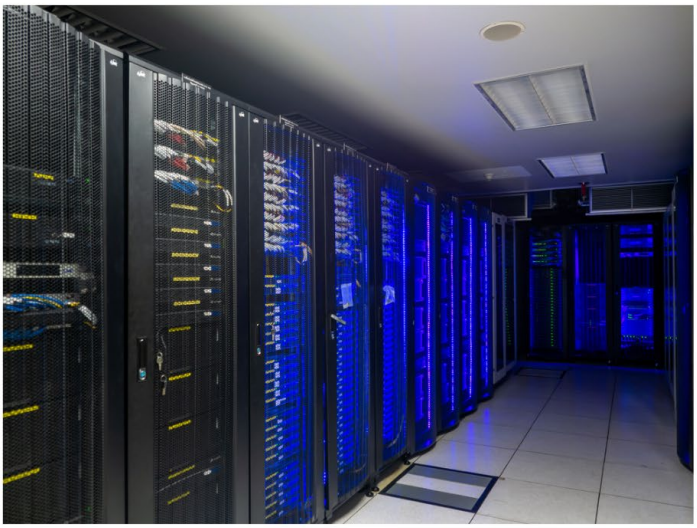
But with cloud plaforms, the providers takes care of the underlying infrastructure.

Examples are Microsoft Azure, Amazon Web Services.

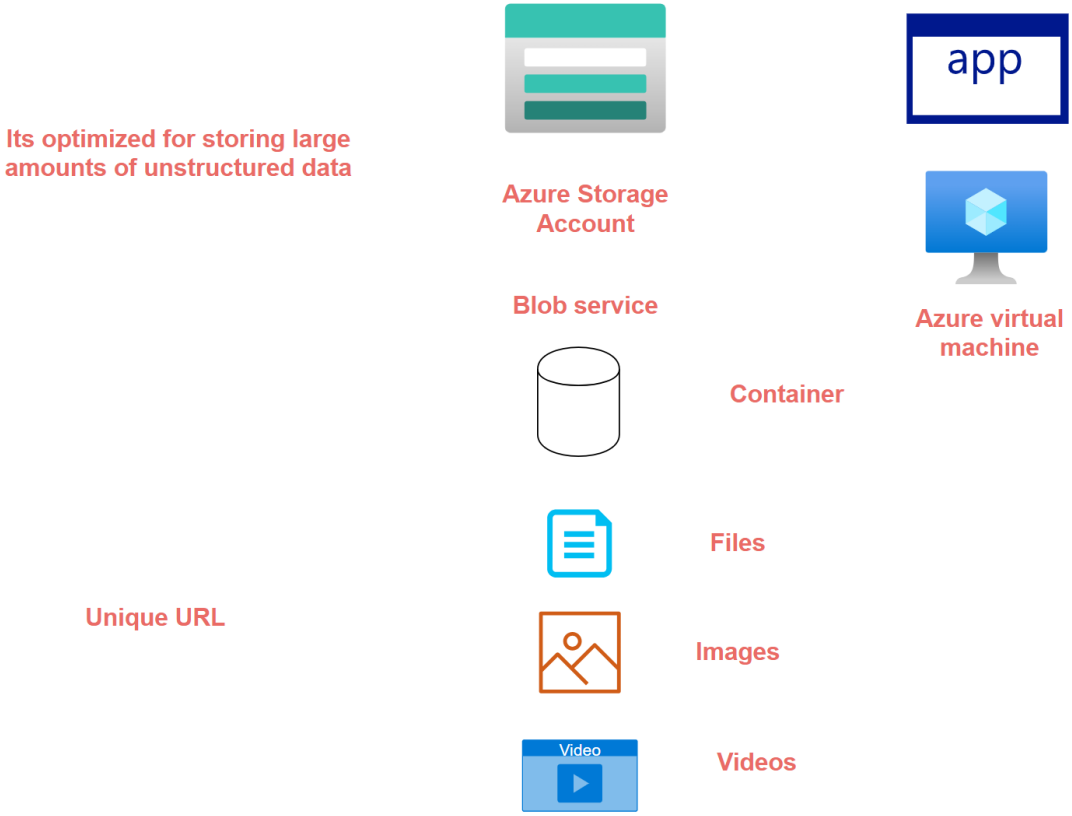
You can create virtual machines which can then be accessed via the Internet.

Here you pay based on how much you use.

You don't need to invest on any infrastruture head-on.

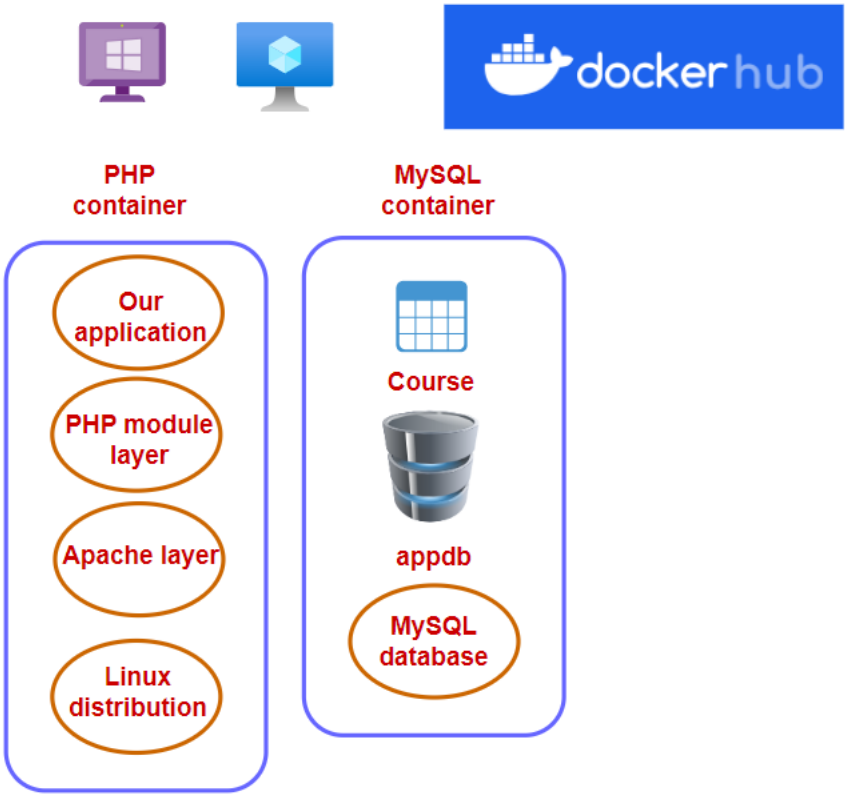


Lab- Using Azure Blob Storage- Building our image



Kubernetes- Let's understand the way it works

What is Kubernetes



We have seen how we can containerize our applications using Docker.

Docker helps you to build an isolated environment for your application.

This helps to ensure that applications running on the same environment don't clash with each other.

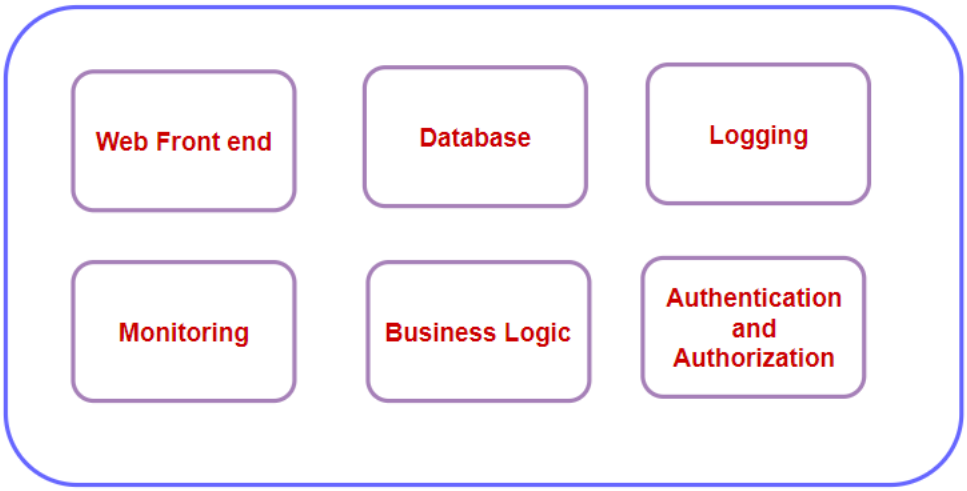
It also becomes easier to port applications between environments - From Development to Test.

As a developer you can build your application and then package it into a Docker Image. That Image can then run as a container.

We have looked at Docker compose. This can be used to deploy multi-container applications.

What you want to be deployed can be specified in a configuration file.

Applications can be built around multiple containers.



And when a company develops multiple container-based applications, things can become complicated.

Companies can then look towards using Orchestration tools.

You also have a tool known as Docker Swarm.

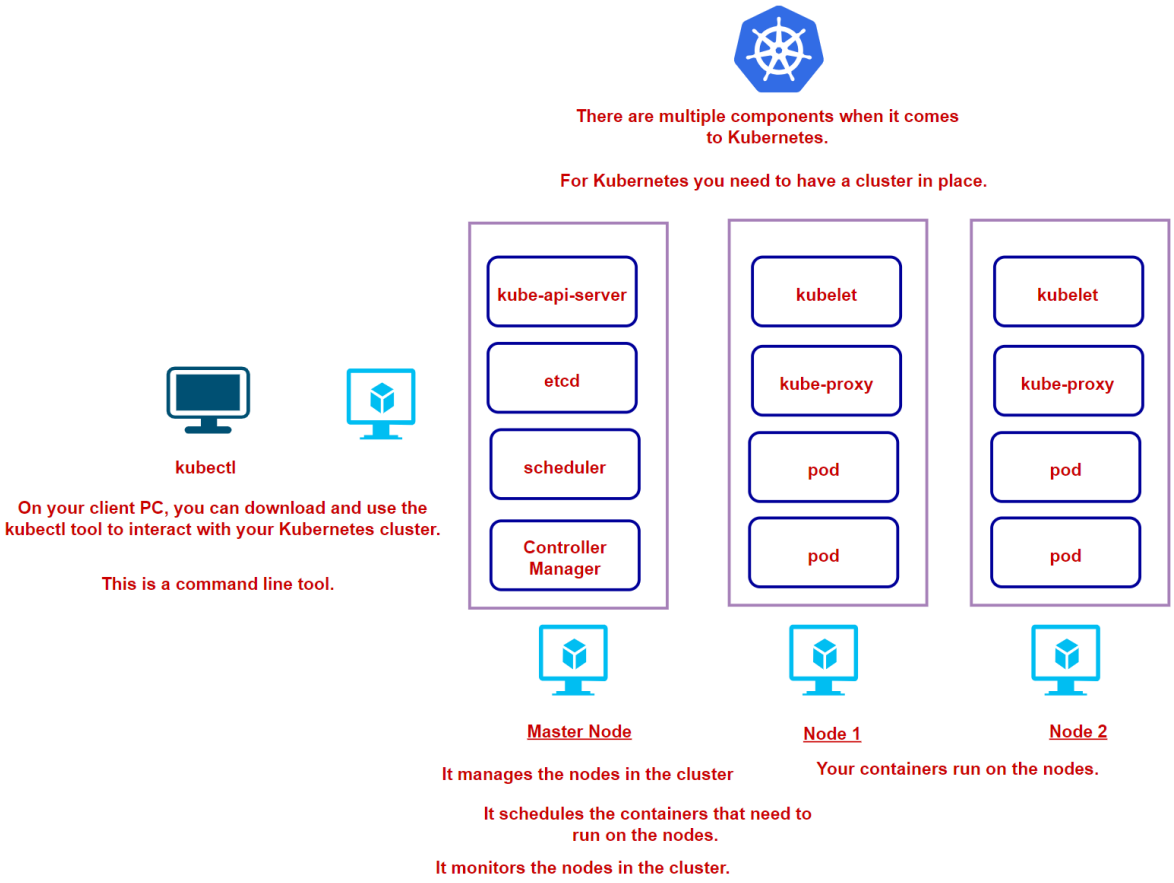
A popular tool when it comes to container-orchestration is Kubernetes.

It's an open source platform that is used for managing your containerized workloads and services.

Features provided

1. It can restart containers if they fail
2. You can load balance traffic across your containers.
3. You can dictate the state of the services that need to run.
4. You can mount different storage systems when it comes to persistence of data.
5. You can scale up your services whenever required.

Kubernetes Architecture



Having multiple nodes helps to achieve high availability. If one node fails, Kubernetes can bring up your application on another node in the cluster.

Cloud platforms have a managed version for Kubernetes.

kube-apiserver - This is part of the control plane. All requests to the Kubernetes cluster goes through the kube-apiserver.

etcd - This is a highly available key value store that is used for storing the cluster data.

kube-scheduler - This is part of the control plane that selects a node for a Pod to run.

The Control plane also has various controllers for managing different aspects of the cluster. For example you have the Node controller - This is responsible for managing the nodes. It understands the status of the nodes.

kubelet - This is an agent that runs on every node in the cluster. It is responsible for running the containers in a Pod. It is also responsible for the communication between the control plane and the node.

kube-proxy - This is a networking service that allows communication across the pods in the cluster.

In Kubernetes your containers run as part of a Pod.

This is the smallest deployable unit of computing in Kubernetes.

A Pod can consist of a single or multiple containers.

A Pod can also have shared storage and networking resources.

The Pod also uses the container runtime for running containers.

The company's development team will develop the application and package it into a Docker image.

You can make use of the Docker client, Docker Desktop to work with Docker.

Kubernetes is another tool that needs to be installed and configured.

You then deploy your container-based application onto the Kubernetes environment.

Normally the deployment of your application is done by the IT Administrative team.

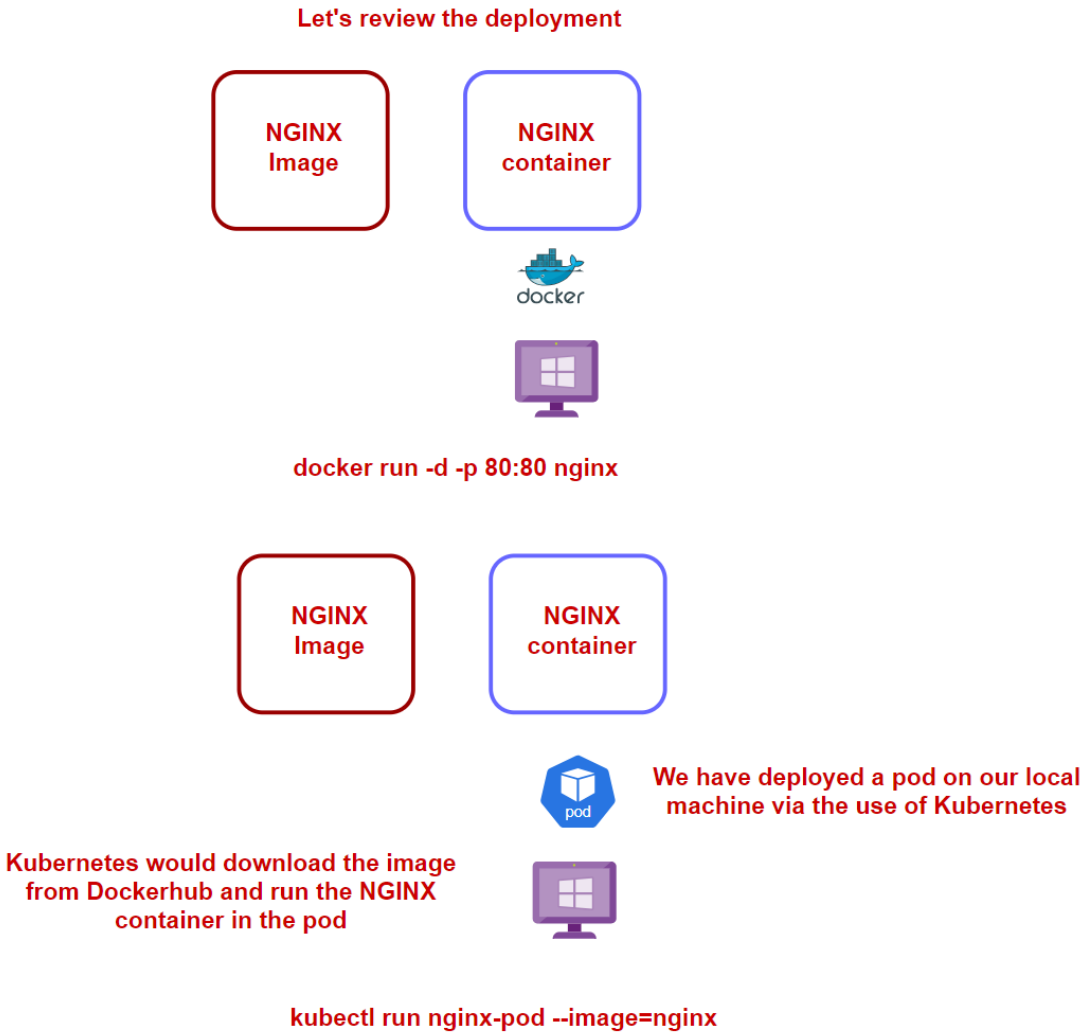
But nowdays as a developer, its ideal to know how you can deploy your container-based application onto Kubernetes.

And we will dvelve a lot into Kubernetes, because you could be interested in different roles , as a Developer/Administrator/Security Engineer.

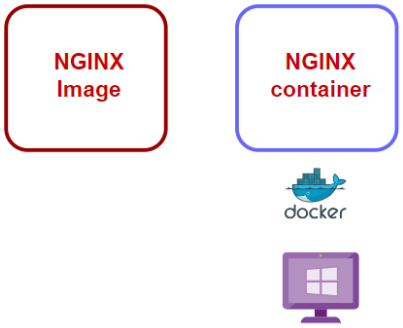
To get started with Kubernetes, we can use Docker Desktop. Or you can also use MiniKube.

When you finally want to deploy your application onto a production environment, you should make use of a fully fledged Kubernetes environment.

Lab- Creating a Pod

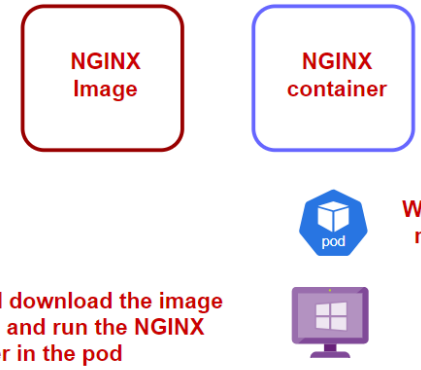


Exposing the Pod



Remember -p was used to expose the port of the container onto the port of the host machine.

docker run -d -p 80:80 nginx



We have deployed a pod on our local machine via the use of Kubernetes

Kubernetes would download the image from Dockerhub and run the NGINX container in the pod

kubectl run nginx-pod --image=nginx

Here our Pod only gets an internal Pod IP. And we cannot access the NGINX web server running as a container in the Pod from the outside world using this IP address.

What we can do is to execute a command for port forwarding.

Container port

kubectl port-forward nginx-pod 80:80

What's happening in the background

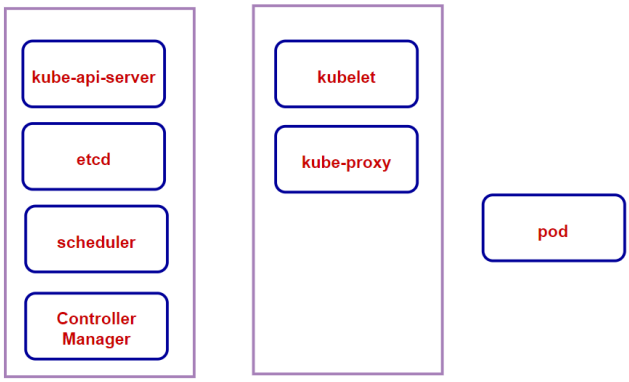
kubectl

1. Using the kubectl tool we submit our manifest file.

2. Kubectl will make a call to the kube-api-server with the manifest content converted to a JSON payload.

nginx.yml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx-instance
spec:
  containers:
  - name: nginx-instance
    image: nginx
    ports:
    - containerPort: 80
```



Remember everything is running on our local machine.

In Kubernetes control plane, the request for creating a Pod will go through the necessary checks.

If all of the checks are passed, the data is persisted in the etcd database.

Then the kube-scheduler decides on the ideal node to run the pod.

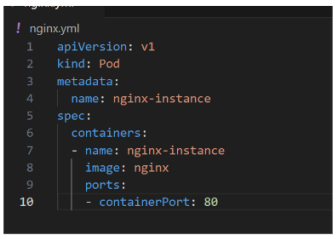
The kubelet services which runs on the nodes continuosly monitors for pod requests. Once it understands a request is present it then goes ahead and creates the Pod on the node.

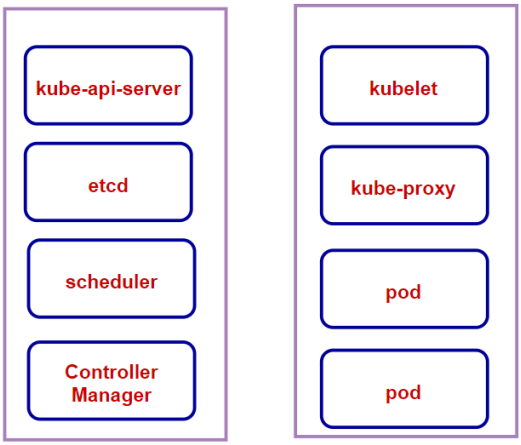
Using deployments in Kubernetes

kubectl

Earlier on, we directly created a Pod object.

Normally you will create a deployment object.





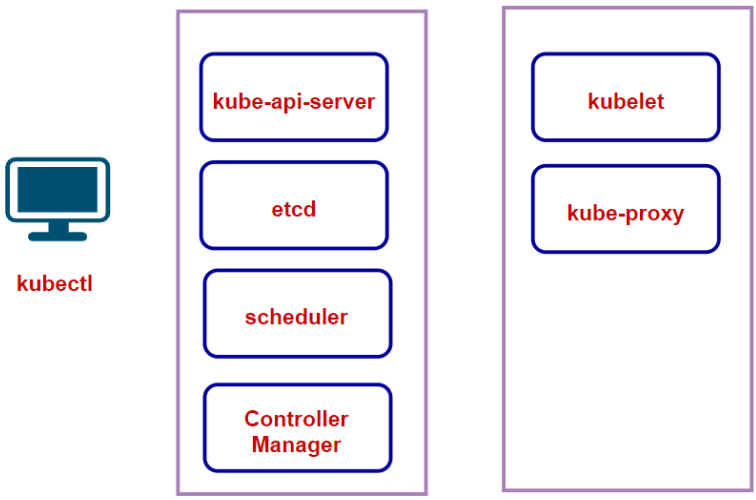
When you specify a deployment object, Kubernetes will again work to ensure it reaches a state which fulfils the requirements for the deployment.

You can define the Pods, what containers to run and the number of instances to run.

You can also delete deployments and roll back deployments.

The Deployment Controller is used to manage the deployment itself.

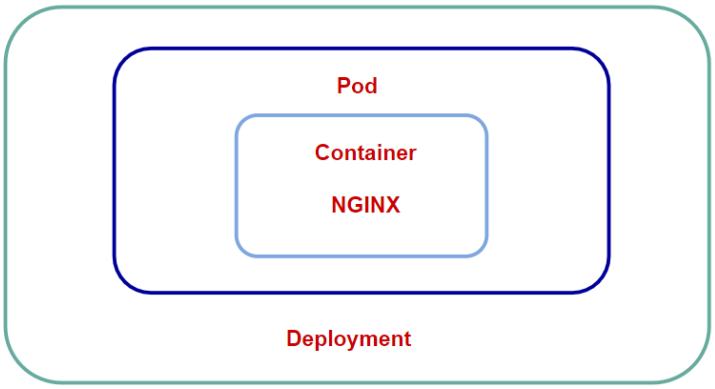
Understanding the deployment



Remember the entire Kubernetes setup is running on our machine via the use of Docker Desktop

nginx-deployment.yml

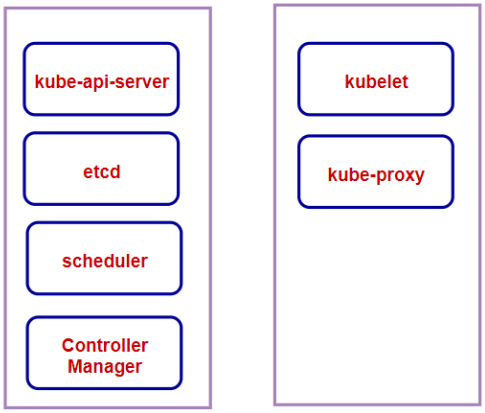
```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deplyment
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx-instance
  template:
    metadata:
      labels:
        app: nginx-instance
    spec:
      containers:
      - name: nginx-instance
        image: nginx
        ports:
        - containerPort: 80
```

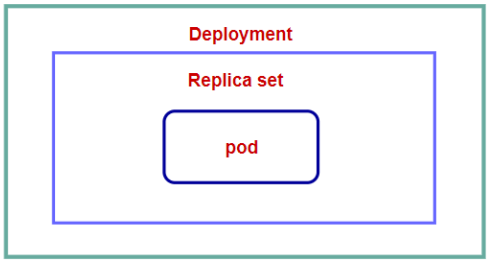


We have now submitted a deployment

Understanding ReplicaSets

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deplyment
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx-instance
  template:
    metadata:
      labels:
        app: nginx-instance
    spec:
      containers:
      - name: nginx-instance
        image: nginx
        ports:
        - containerPort: 80
```

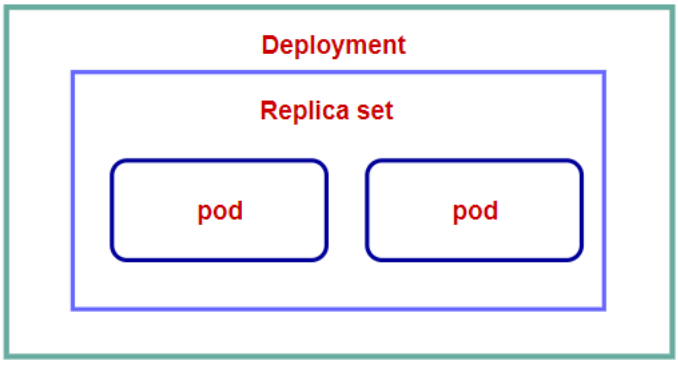




The deployment is the over arching object.

This manages replica sets. Replica sets ensures that a set of Pods are always running based on the number specified in the replica set.

Let's say for some reason the Pod goes down. The replica set will create a new instance of the Pod. Remember that the older Pod does not get recreated again.



You can also have multiple instances of your Pod running. This is based on the same Pod template specification.

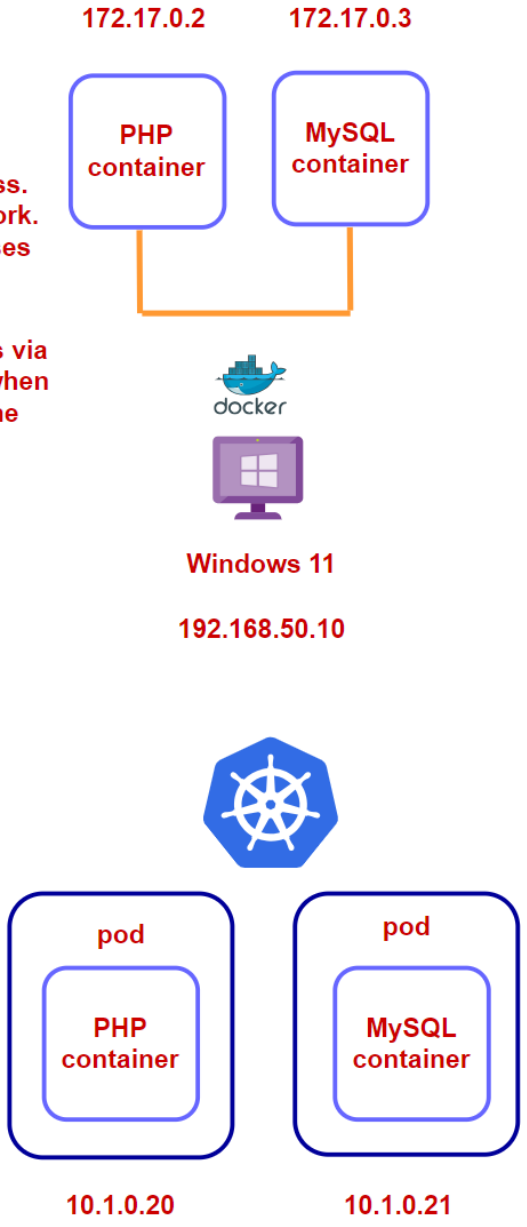
This helps in high availability. If one Pod goes down as part of your application, you still have another Pod that is in the running state.

Using Services in Kubernetes

When working with Docker we learnt about Networking.

By default containers get an IP address. You can create your own Docker network. And the containers receive IP addresses based on the new network.

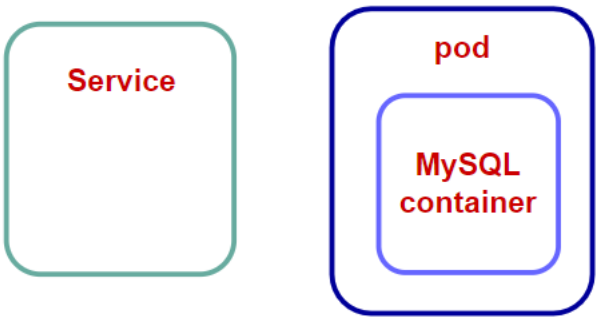
And we could also access containers via their container names - Like we did when our application needed to talk to the MySQL database server.



Similarly each Pod also gets an IP address

But we know Pods are ephermeral in nature. They can get terminated and Kubernetes can launch a new Pod that can get a new IP address.

Hence for pod to pod communication we should not rely on the IP addresses.



For this we can create a Service in Kubernetes.

We can contact the Service which in turn will communicate with the Pod , no matter what is the IP address of the Pod.

The labels are used to associate the Service with a Pod.

The kube-proxy service is responsible for creating a virtual IP address for the service.

The Services don't get killed like Pods. They persist.

There are different types of services in Kubernetes.

A service in the end is an object in Kubernetes.

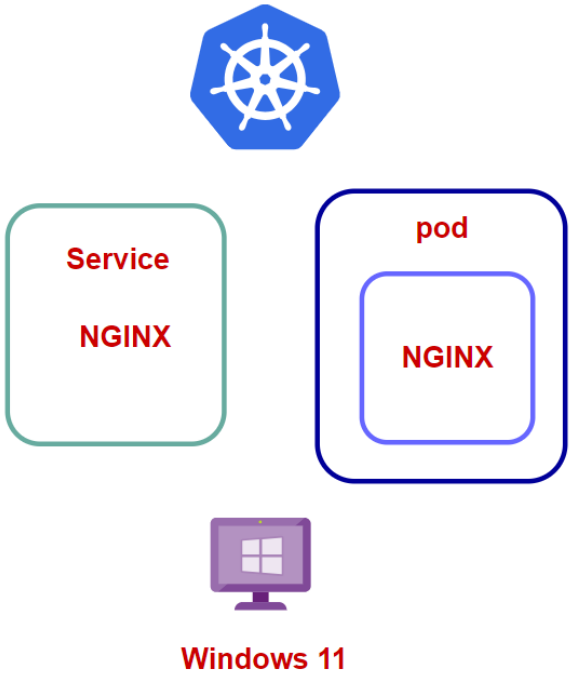
Lab- Using Services in Kubernetes

We are going to create a service via the use of a YAML definition file.

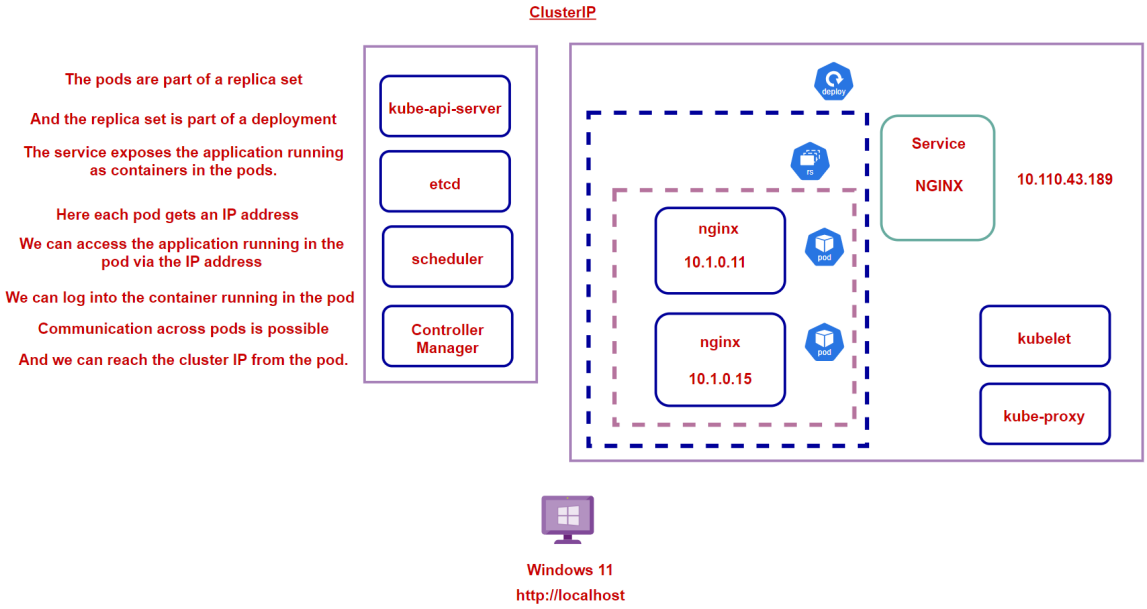
We will use kubectl to deploy the Service Object.

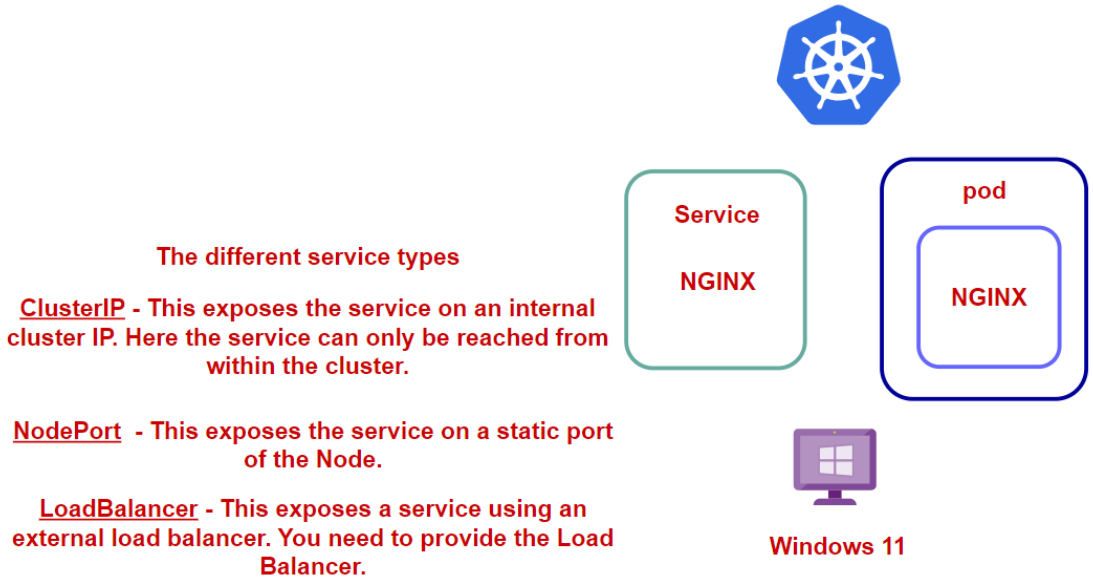
The Service will be of the type LoadBalancer.

We can then access NGINX running in the Pod via the use of the service.



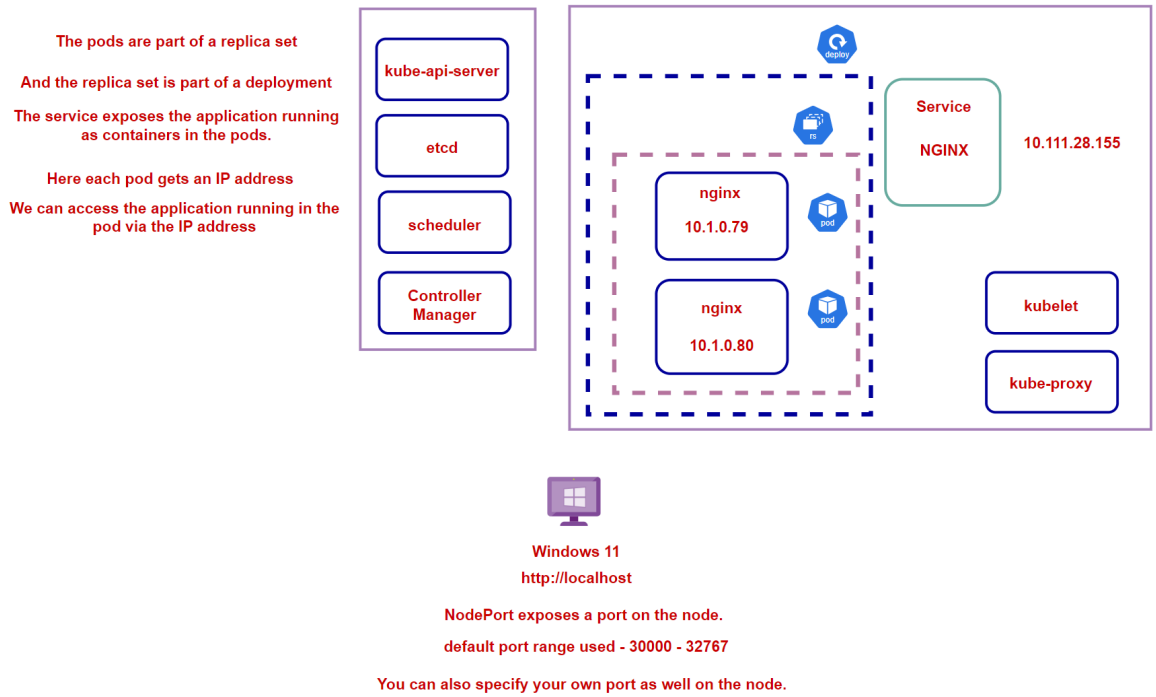
Understanding service types – ClusterIP





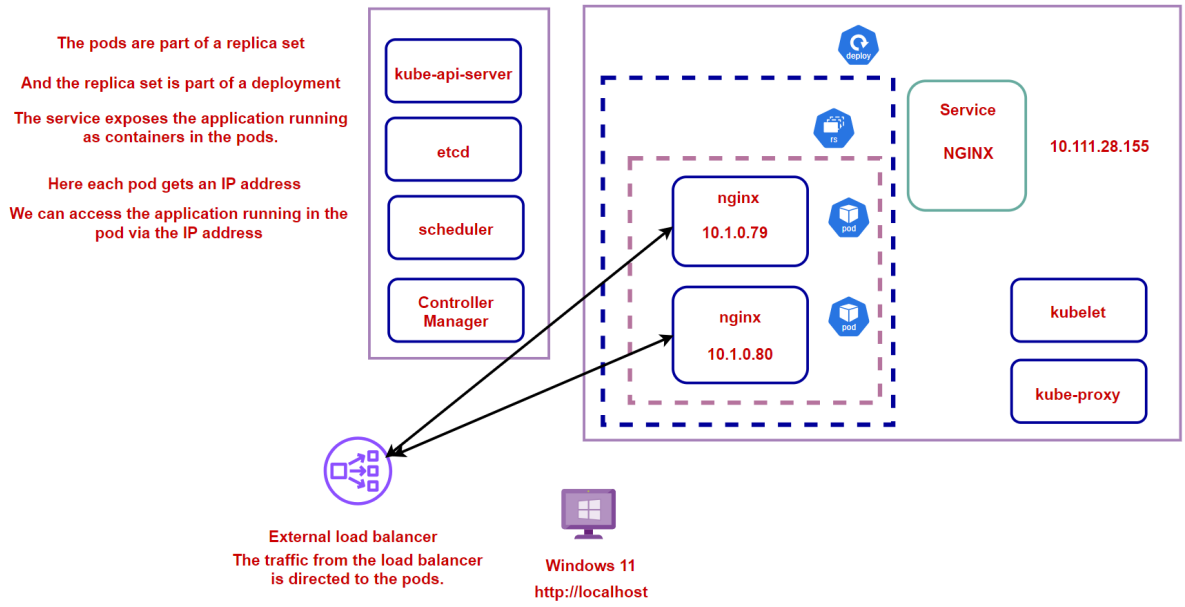
Understanding service types – NodePort

NodePort



Understanding service types- Load Balancer

Load Balancer



Deploying multiple containers onto Kubernetes

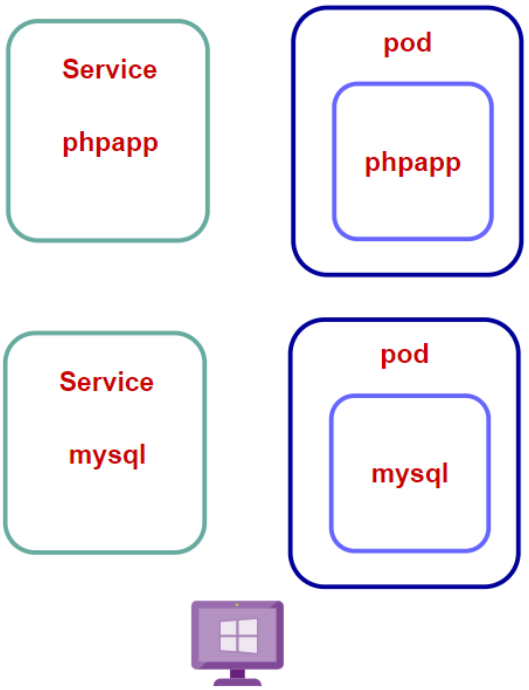
We are going to have different pods. One pod for the application and one pod for the database.

We can deploy the application and database in the same pod.

But then the database operation is different from an application.

For example, we can scale the instances of a pod. And the scaling works different for a database and an application.

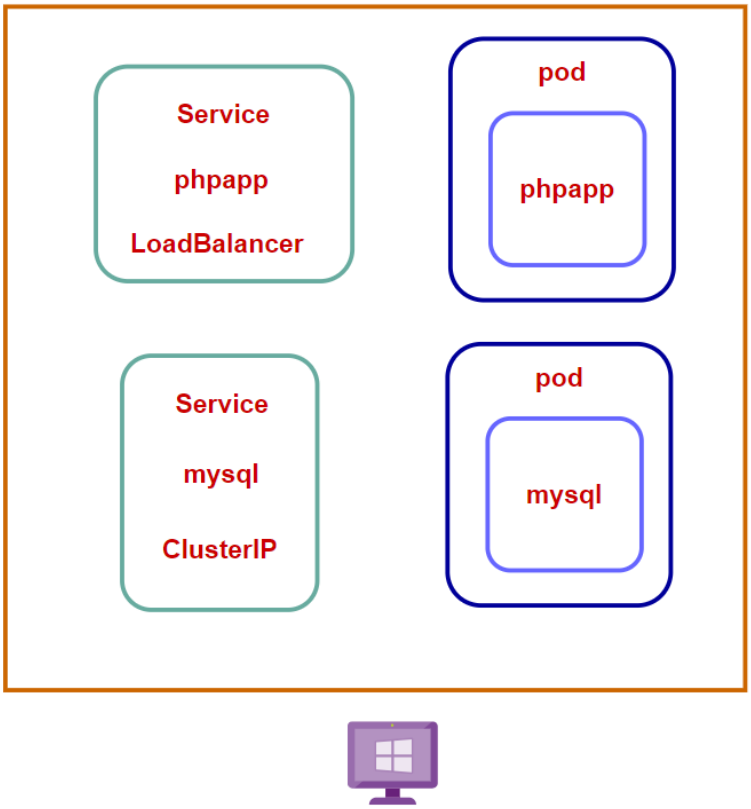
Also we might want to have storage attached seperately just for the database.



Windows 11

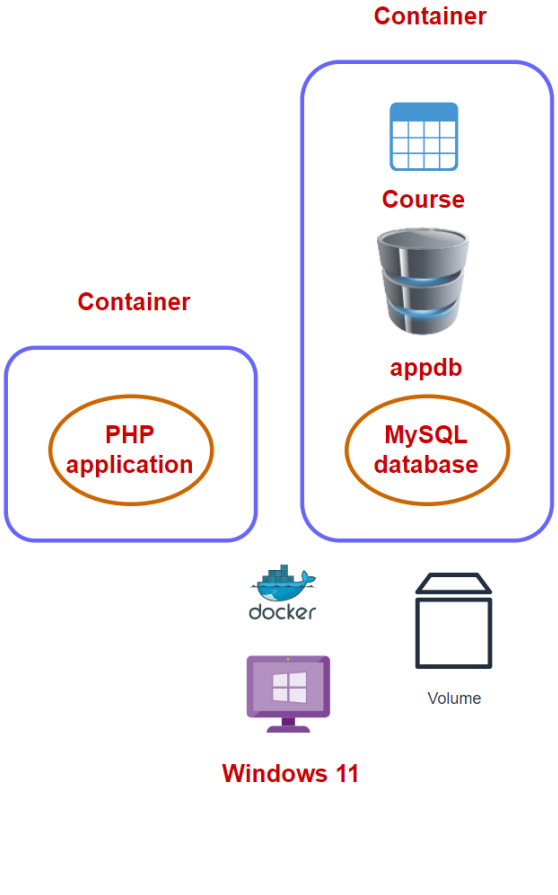
This time we will use ClusterIP as the type for the service.

The Load Balancer helps in a way to reach the service from the outside world



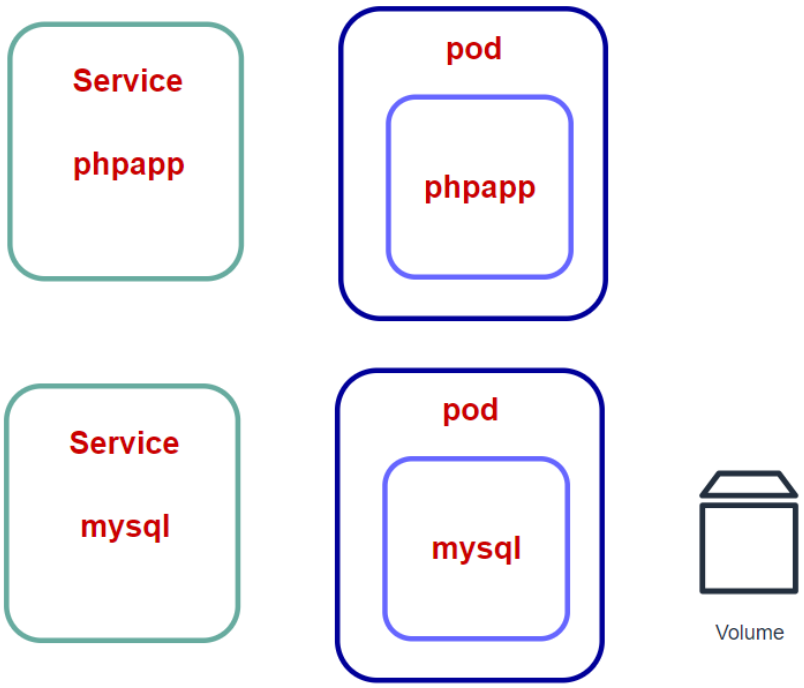
Kubernetes – Storage

Usage of volumes



We have already seen the use of volumes with Docker.

We use volumes when it comes to persisting data.

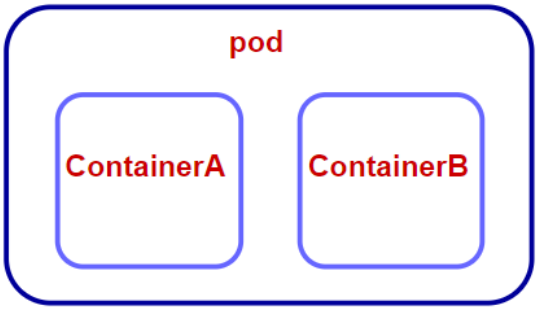


Windows 11

We can also use volumes with Kubernetes.

The way we do this is different when compared with Docker.

There are different types of volumes



Volume

You could have a requirement for multiple containers in a Pod to share data. This can be done via the use of a volume.

For this maybe you can use the emptyDir volume.

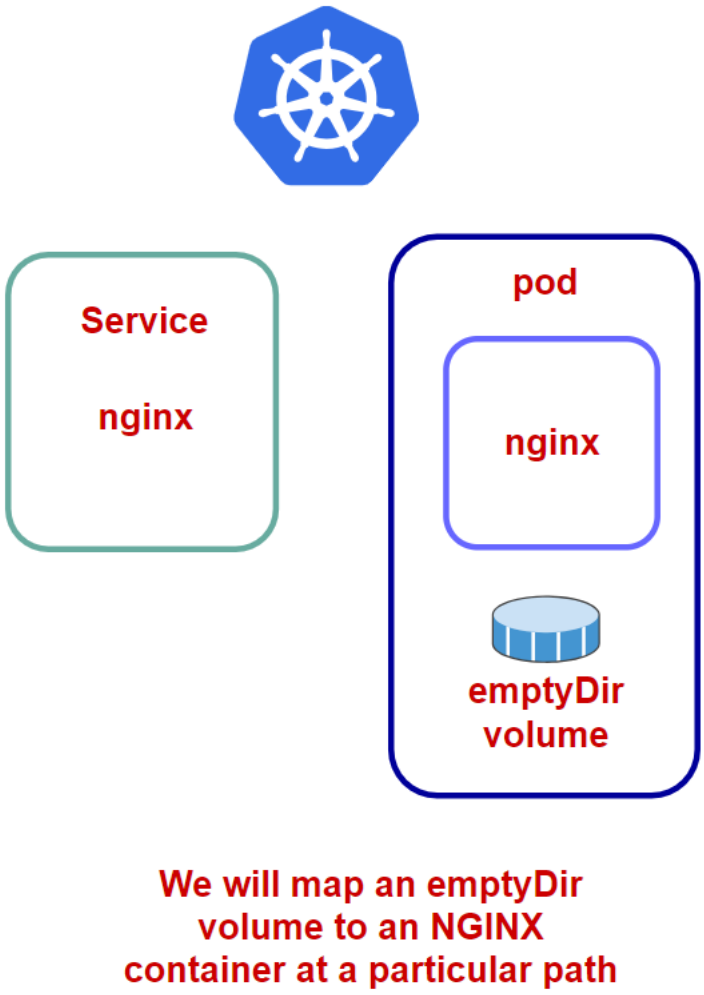
This is a directory that initialy has no data.

This directory can be mapped to different paths within different containers in the pod.

If a container crashes in a pod and needs to be restarted, the data in the directory will be still be present.

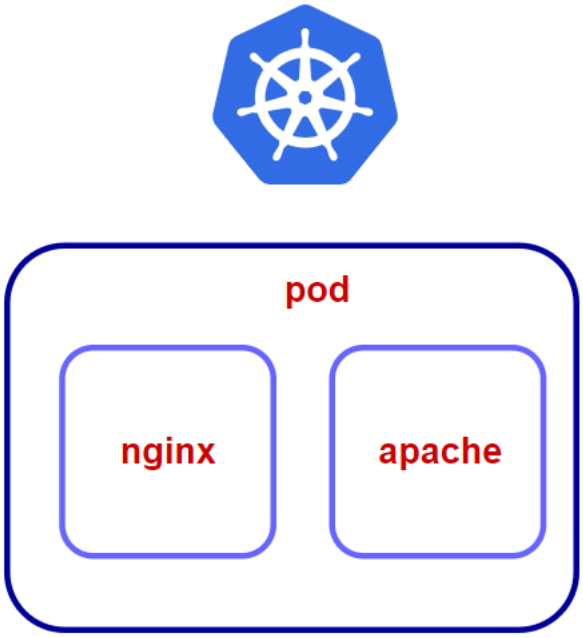
But if the pod is terminated, the directory will also be deleted and you will loose the contents of the directory.

Lab- Using the emptyDir volume type



We will map an emptyDir volume to an NGINX container at a particular path

What we need to do to solve our issue



I want to run two containers in a pod.

Both are web servers.

Both web servers listen on port 80.

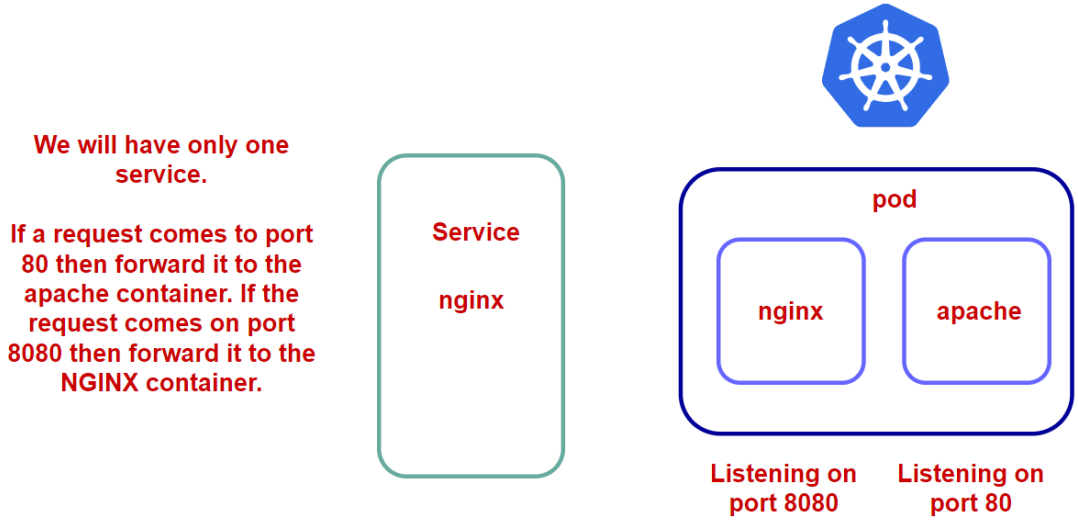
The problem is both containers run on the same pod. You cannot have two web servers listening on the same port.

So I want to make one web server listen on another port number.

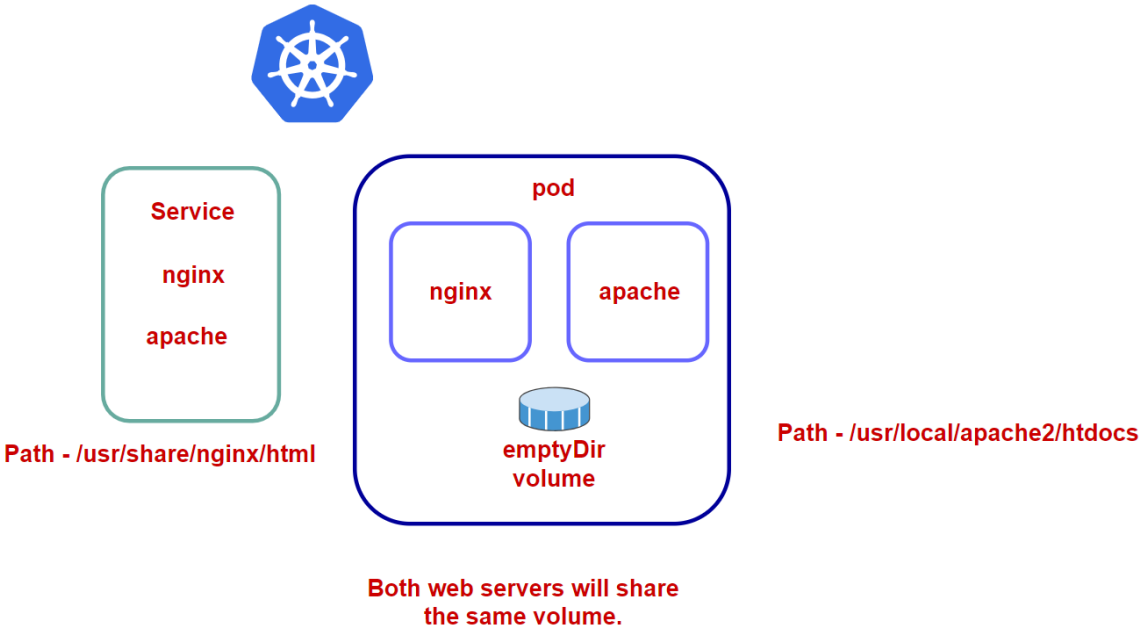
For this I need to change the nginx.conf file that is present in NGINX to listen on another port number.

For this I am going to make use of ConfigMaps - This can be used to store data in the form of key-value pairs. This is good for environment variables, command line arguements or configuration files.

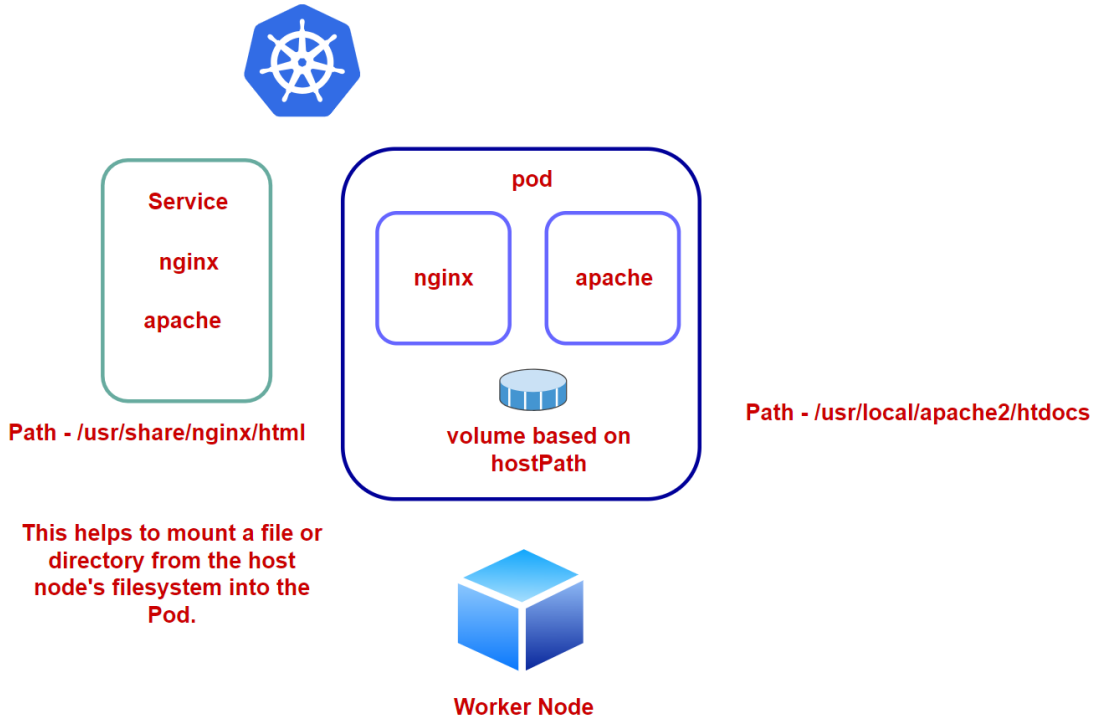
There are different ways of passing in ConfigMaps to Pods. Here I want to replace the nginx.conf file. Hence we will make use of volumes to pass in the file data.



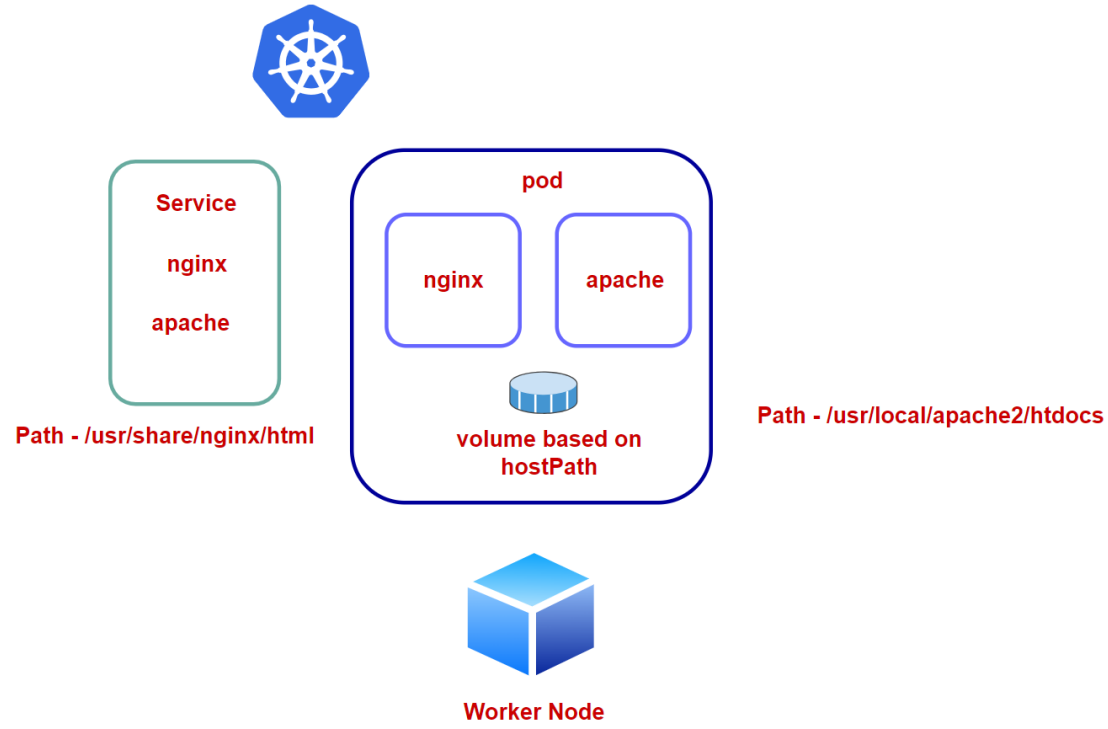
Lab- Sharing a volume across multiple containers



Lab- Using a volume based on hostPath



Using Persistent Volumes



Using hostpath volumes is great when you want data to persist even if the Pod were to fail.

But what happens if the entire node fails.

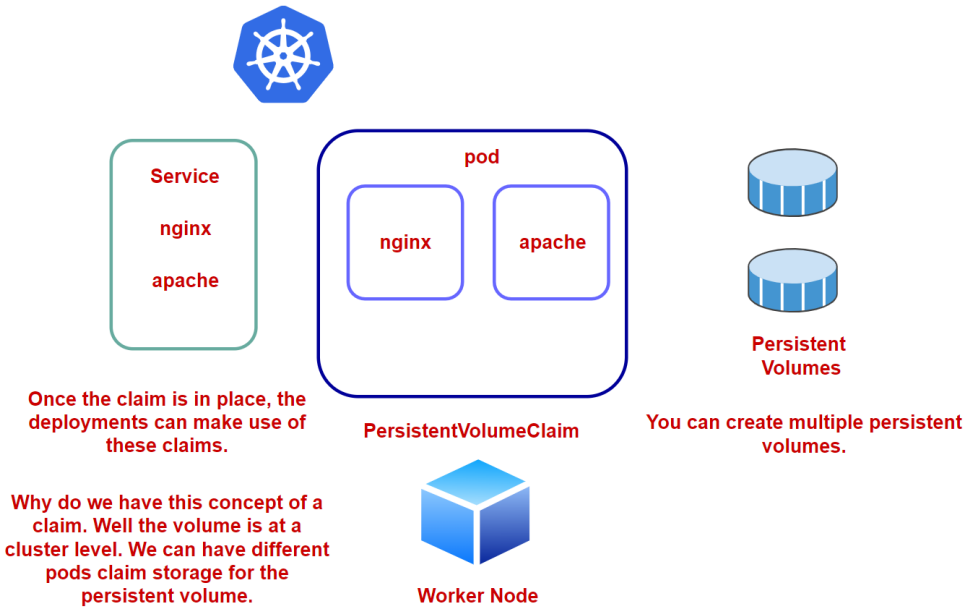
In our case, all is well and fine because we only have one machine as the node. But in a production environment, you will have multiple nodes in a cluster.

To make sure that data is persistent even if a Node fails, we can make use of Persistent Volumes.

You define a Persistent volume as the cluster level.

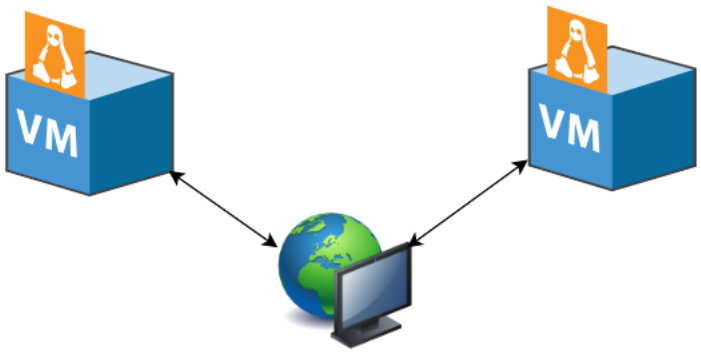
You don't need to define this for each deployment of your Pod. Instead you can just define the volume once.

Your Pods can then make use of these persistent volumes.



Kubernetes- Using cloud platforms

Going through the networking part



By having a network we have the ability to connect machines together.

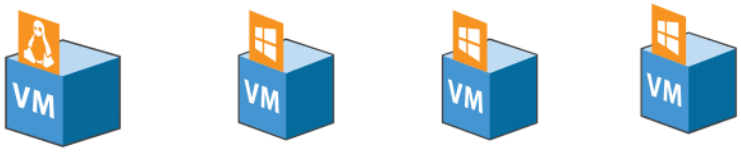
Machines can communicate with each other.

With the help of IP addresses we can identify machines on a network.

We can make use of hardware devices like routers and switches to build networks.

We also now have software aspects that allow us to define networks , especially when it comes to cloud platforms.

The virtualization software can help to leverage the hardware network aspects to build a software-enabled network.



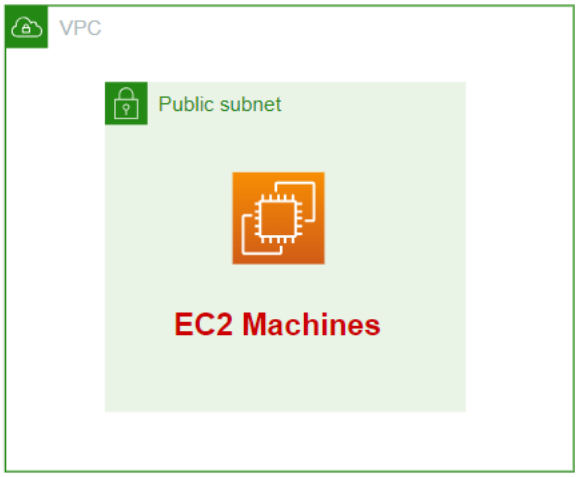
Virtualization Technology / Software

Physical server

Physical server

Hardware enabled networking

Amazon Web Services



Physical server

Physical server

Hardware enabled networking

Datacenter

Review of some core AWS aspects

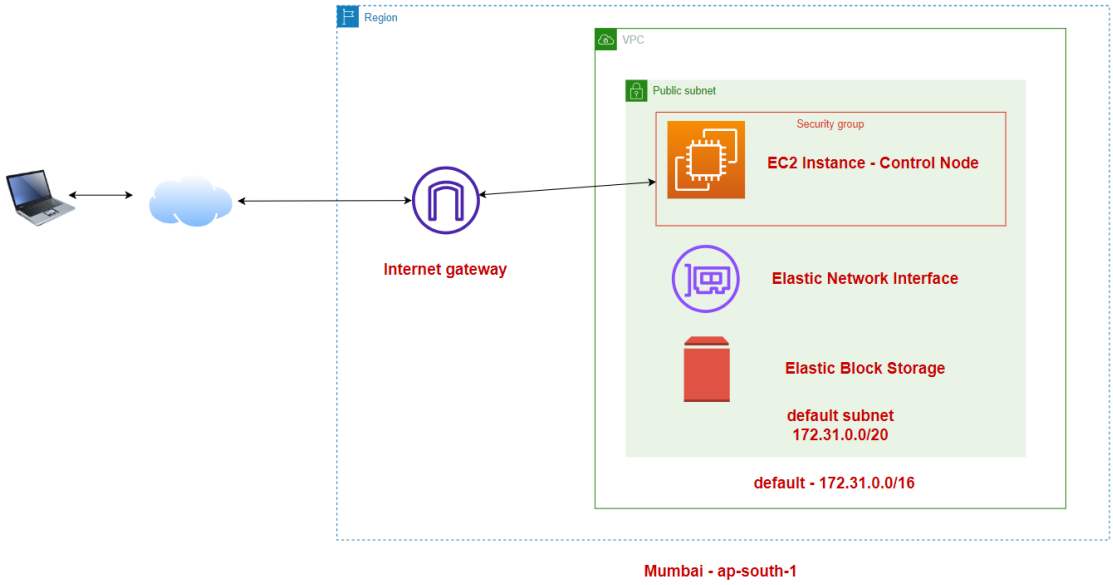
VPC - Virtual Private Cloud - This is an isolated network on the cloud.

Subnets - This is used to segregate the different parts of your network.

EC2 - Elastic Compute Cloud - This helps provsion compute capacity on the cloud.

Internet gateway - This allows resources in the VPC to connect to the Internet.

Elastic Network Interface - This is a virtual network interface for your EC2 Instance.



The Elastic Network interface gets a Private and Public IP address - The Private IP address is used for internal communication in the VPC. The public IP address is used for communication onto the Internet.

EBS volumes - Elastic Block Storage

This is durable, block-level storage devices that can be attached to instances.

The EBS volumes can be mounted as devices on the instances.

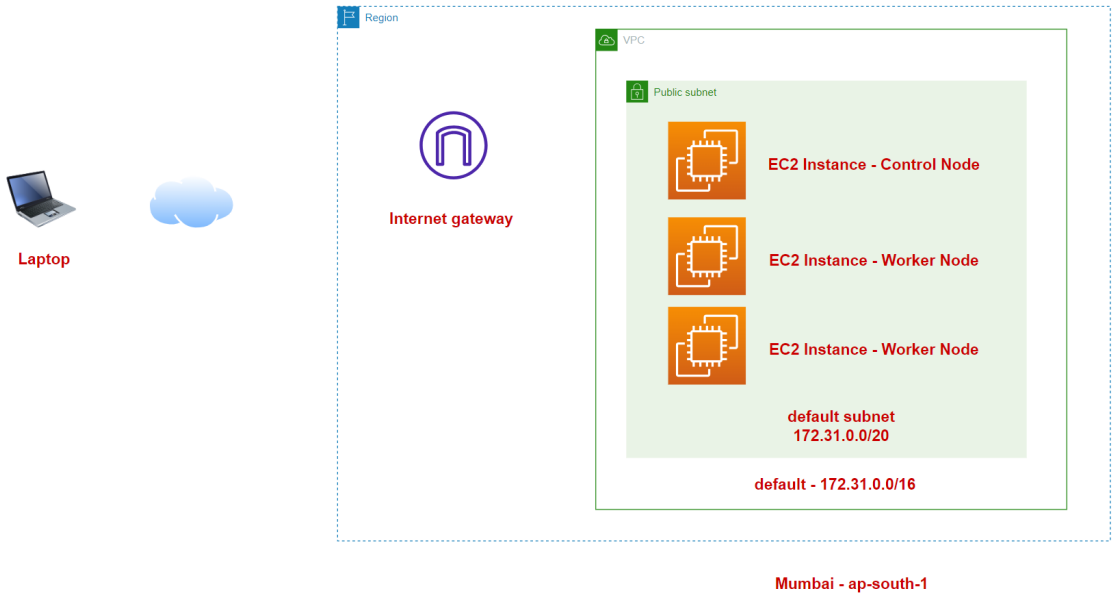
You can then create a file system on the volumes.

Security groups

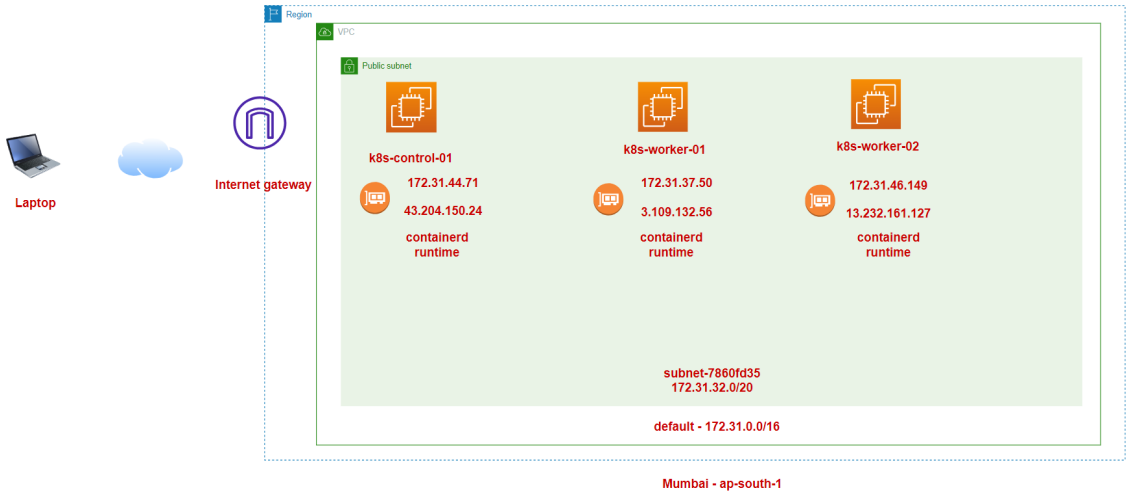
This acts as a firewall. This is used to allow traffic to and from the resources in the VPC.

You can install and host applications on your EC2 Instance.

Quick overview of our implementation



Lab- Cluster Setup- Installing the container runtime



containerd runtime

This is an industry standard container runtime.

So we know that Kubernetes is a tool that can be used to run and manage container-based applications.

We are all familiar with Docker. Before Kunernetes used to be able to run Docker containers by default.

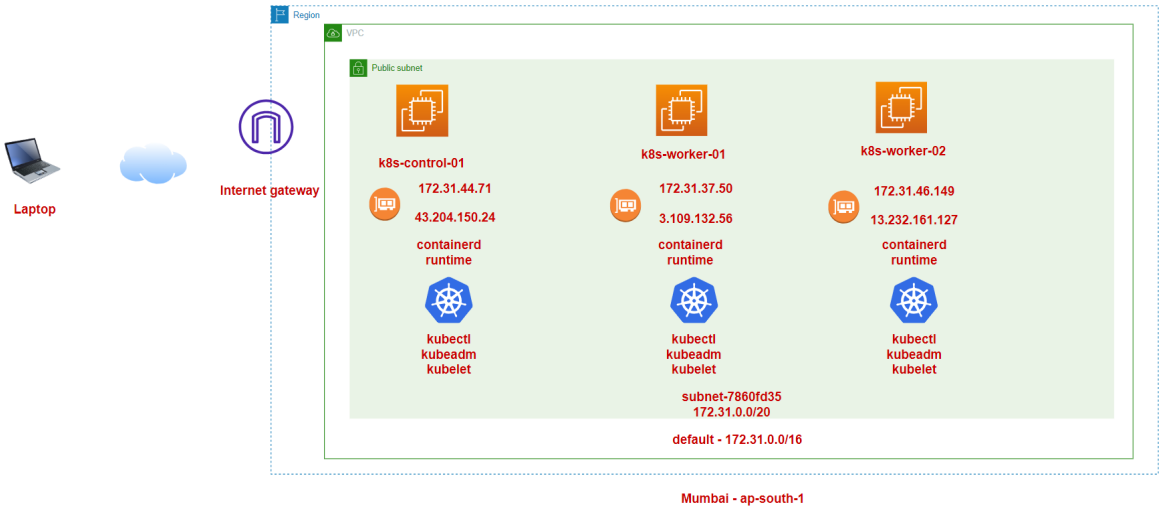
But then we also have other container tools as well - RKT

Kubernetes wanted to have support for other container-based tools as well.

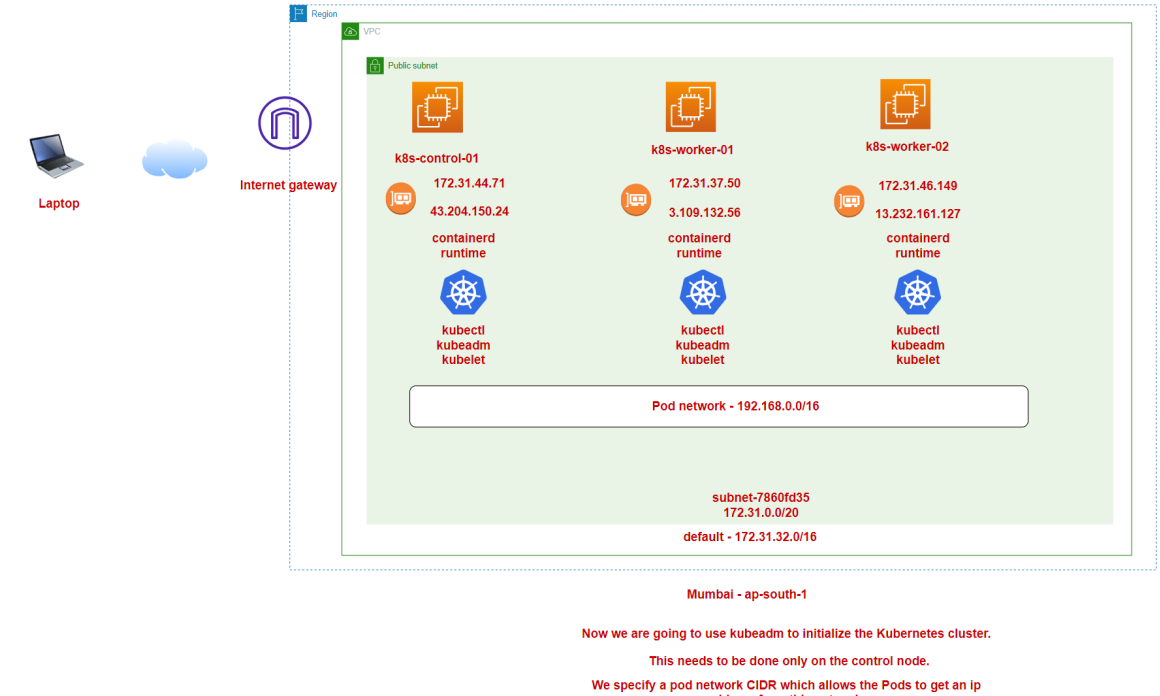
Kubernetes then decided to remove Docker support and instead run containers using the standard runtimes.

And your Docker containers can run by default using these standard runtimes.

Lab- Cluster Setup- Installing kubeadm



Lab- Cluster Setup- Using kubeadm

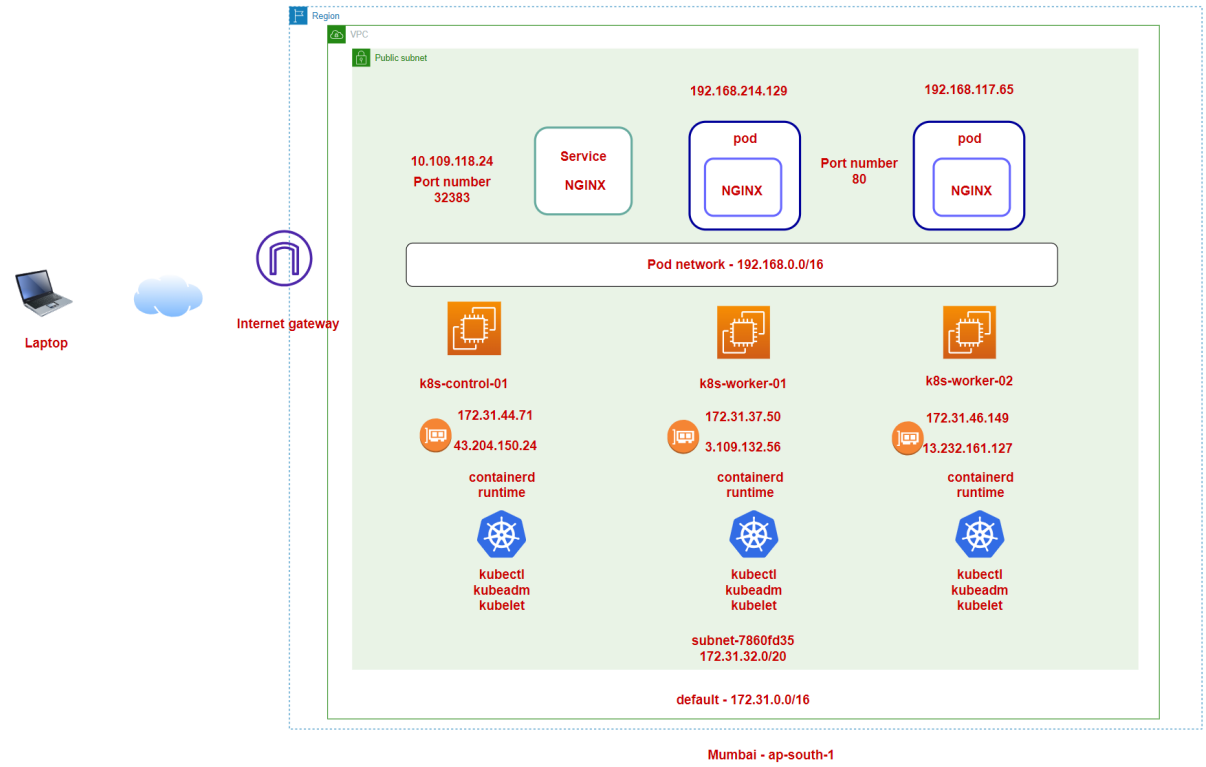


Now we are going to use kubeadm to initialize the Kubernetes cluster.

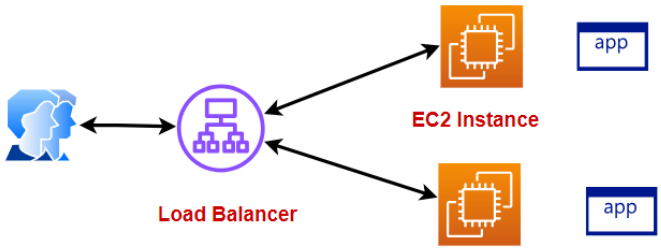
This needs to be done only on the control node.

We specify a pod network CIDR which allows the Pods to get an ip address from this network.

Understanding the IP addresses



Using a Load Balancer

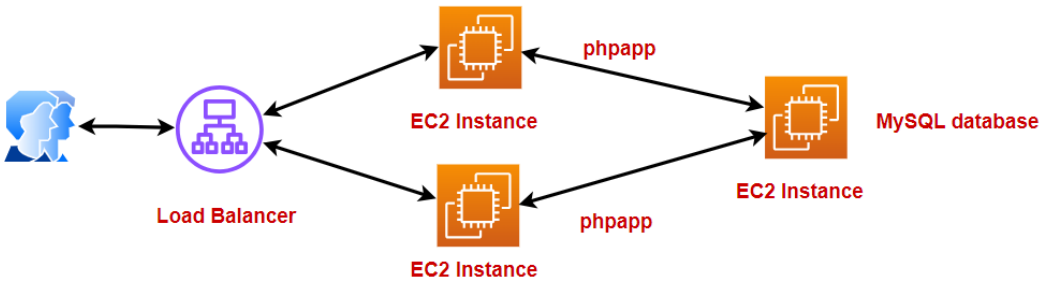


The primary purpose of a Load balancer is to distribute traffic across machines hosting your application.

Cloud platforms normally have an implementation of a Load balancer.

So normally companies deploy multiple instances of their application onto EC2 Instances.

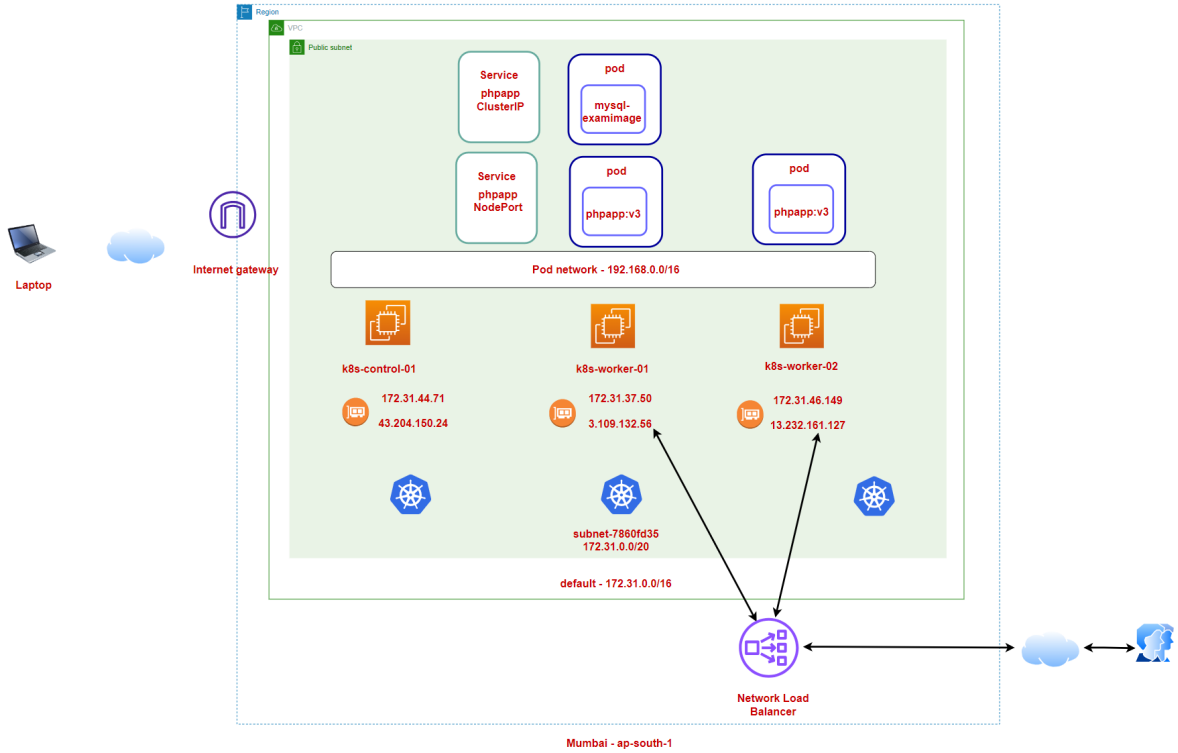
If you are considering our phpapp and mysql scenario



But now our application is defined as pods in a Kubernetes cluster

Listener - This checks for connection requests from clients and forwards it to the target group.

Target Group - This has the targets that are process the requests, normally its the EC2 Instances.



Amazon Elastic Kubernetes service

Amazon EKS

This is a managed service wherein you can use Kubernetes to run your container-based applications.

Here you don't need to install or configure Kubernetes, this is all managed by the service.

You can create a cluster with this service.

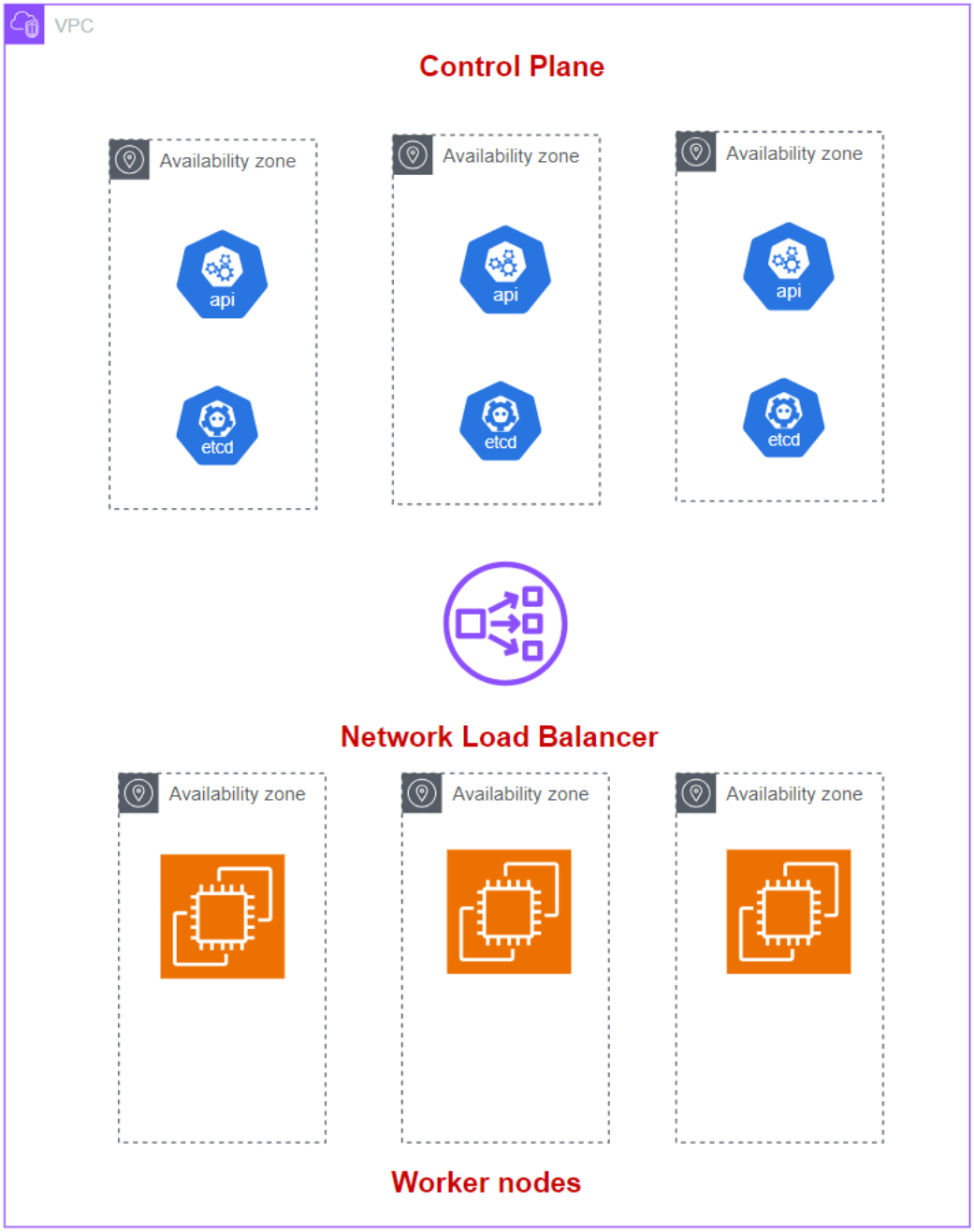
Your nodes can be Amazon EC2 Instances. Or you can make use of serverless deployments via the use of AWS Fargate.

Whenever there is an upgrade in the Kubernetes version, this is made available to you via the service.

Benefits of using Amazon EKS

You need to manage the underlying infrastructure and software.

You can integrate with other AWS services - Amazon Elastic Container Registry for storing the container images.

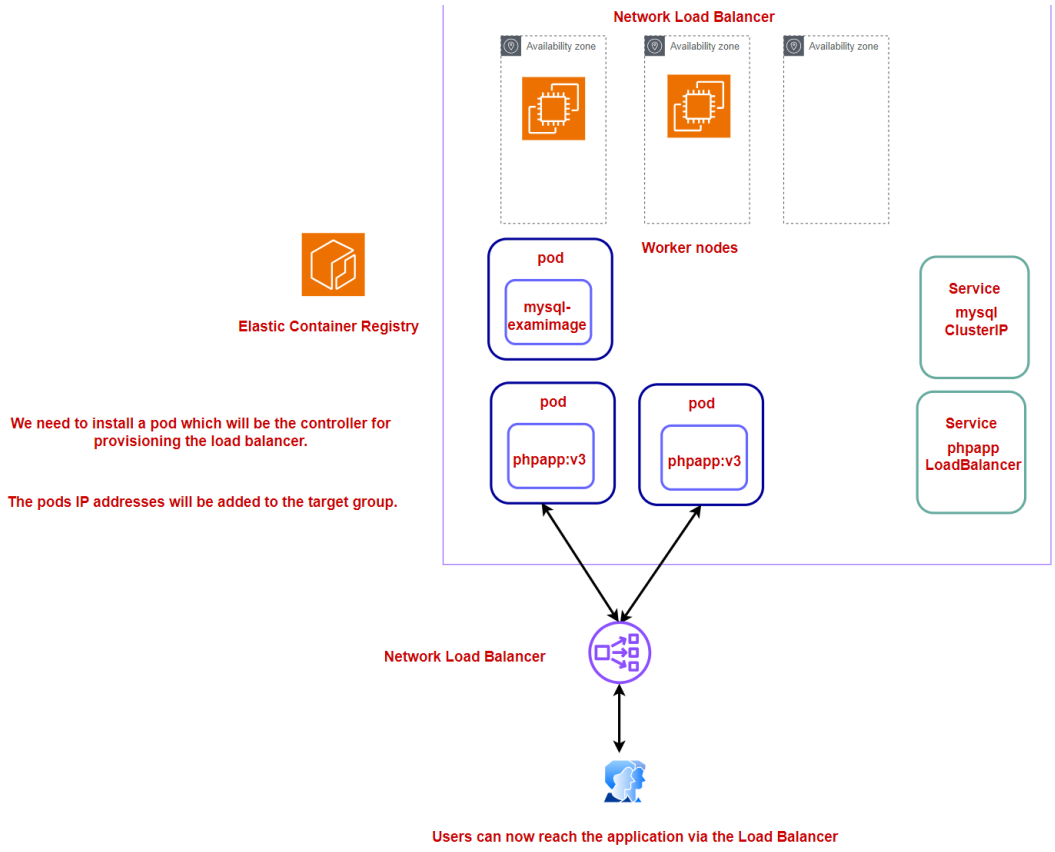


The control plane has at least two API server instances and three etcd instances that are located across AWS Availability zones.

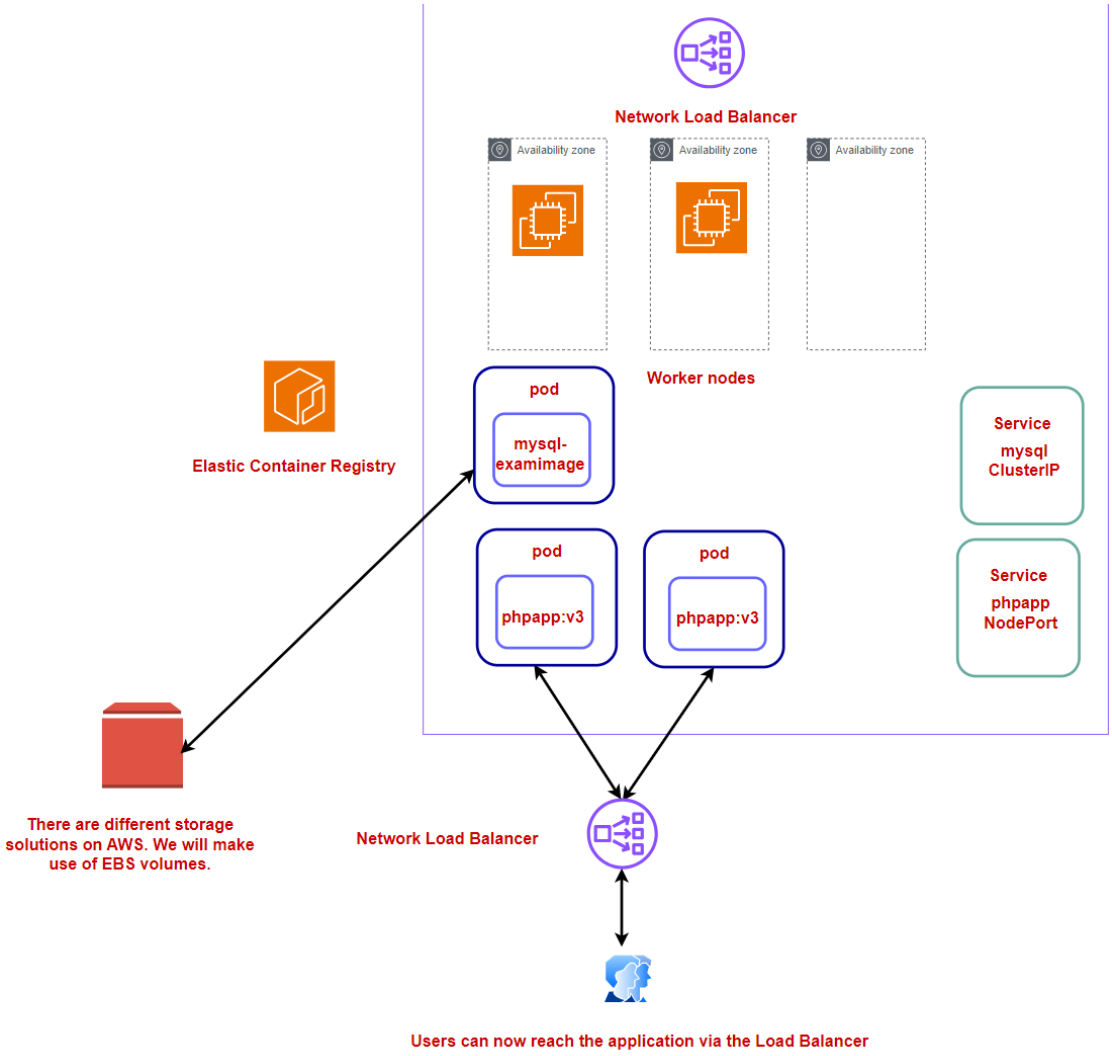
If any instance in the control plane fails, the Amazon EKS service will replace the instance.

Your worker nodes are also distributed across different Availability zones.

Amazon EKS- Lab- Using a Load Balancer



Lab- Using EBS volumes – Implementation



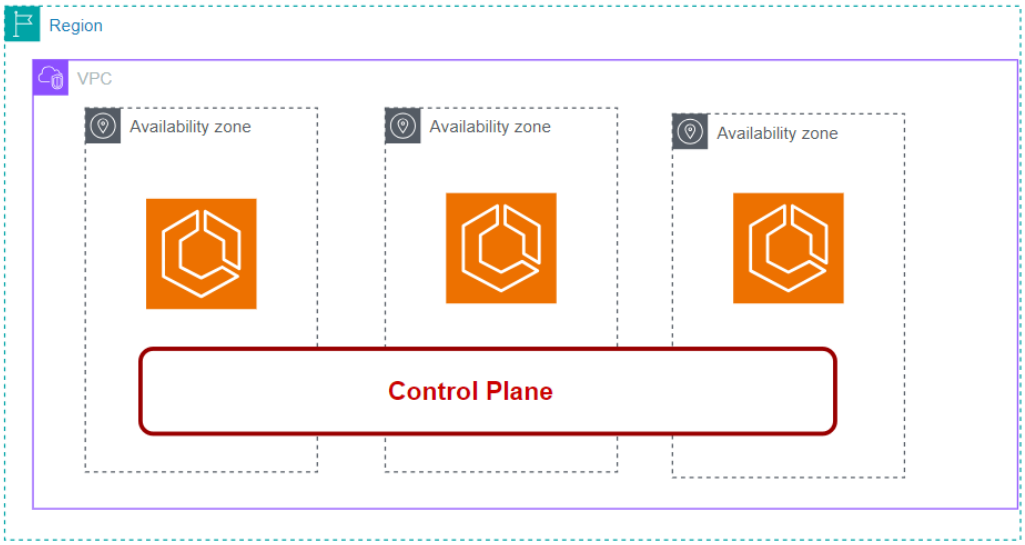
Amazon Elastic Container Service

Amazon Elastic Container Service

This is a fully managed container orchestration service.

Here you can deploy and manage your container-based applications.

Amazon ECS has its own control plane that manages the scheduling and deployment of the workloads.



The actual infrastructure for the deployment of the containers can be Amazon EC2 Instances, AWS Fargate, On-premises infrastructure.

AWS Fargate is serverless compute. Here you don't need to manage the servers.

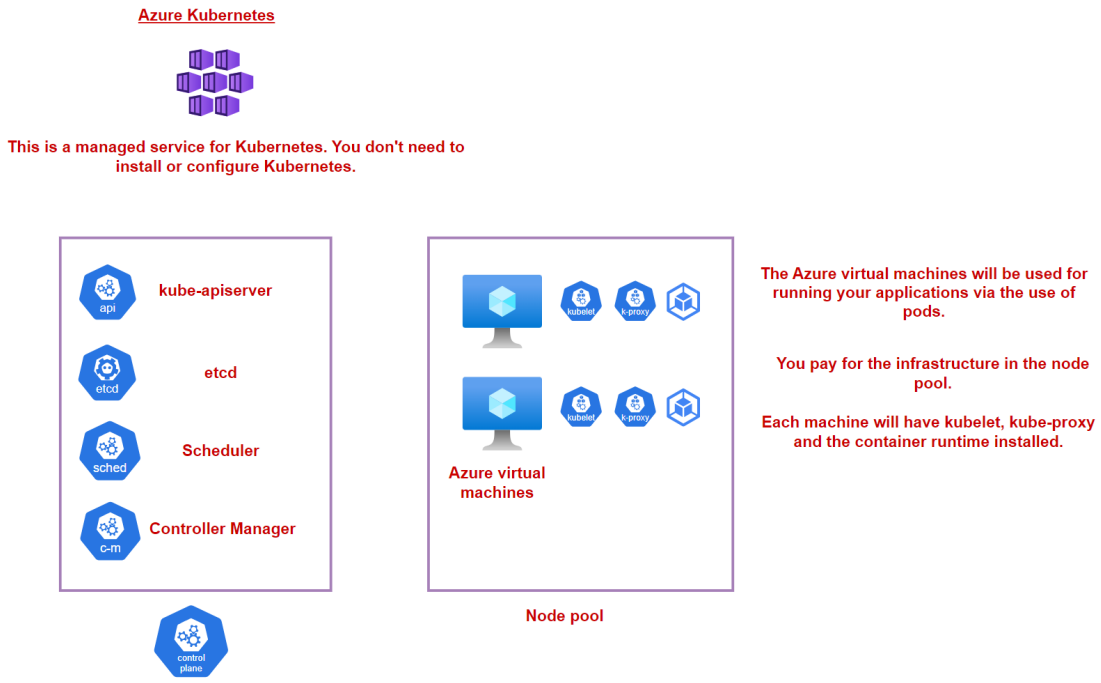
Task Definition

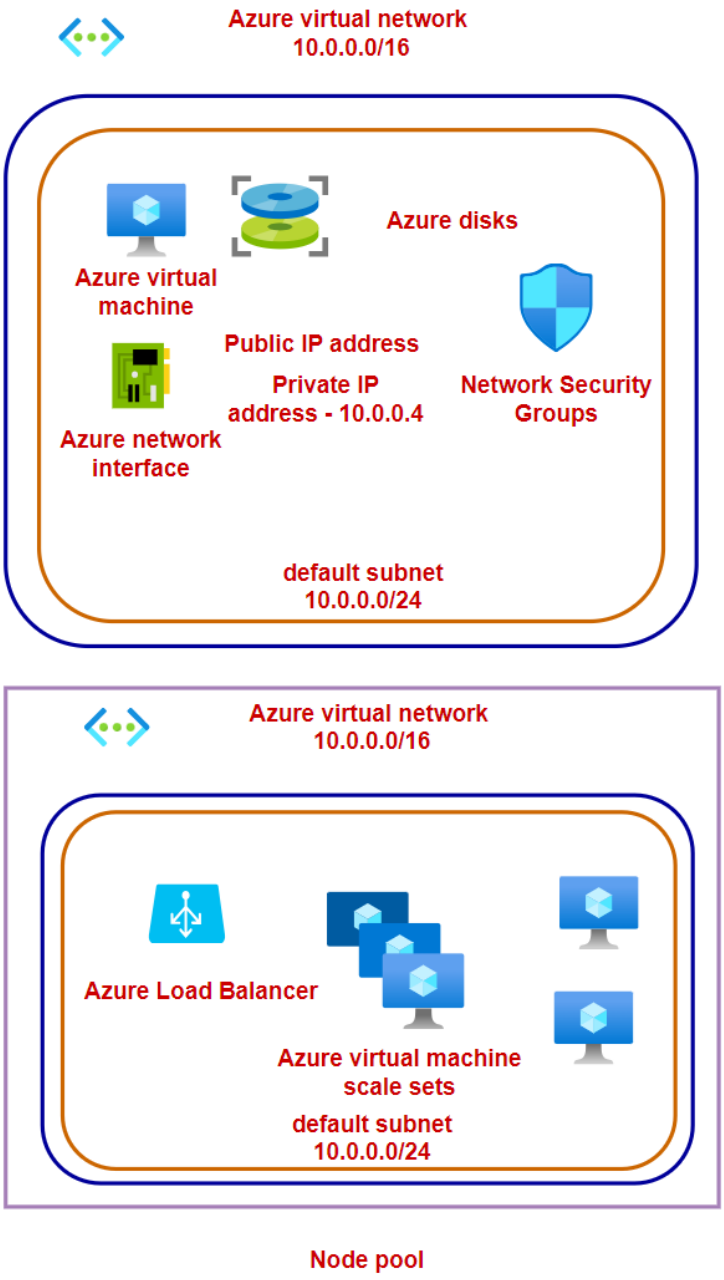
This is a JSON file that specifies the containers that need to run. And also specify the required parameters for the containers to run.

Service

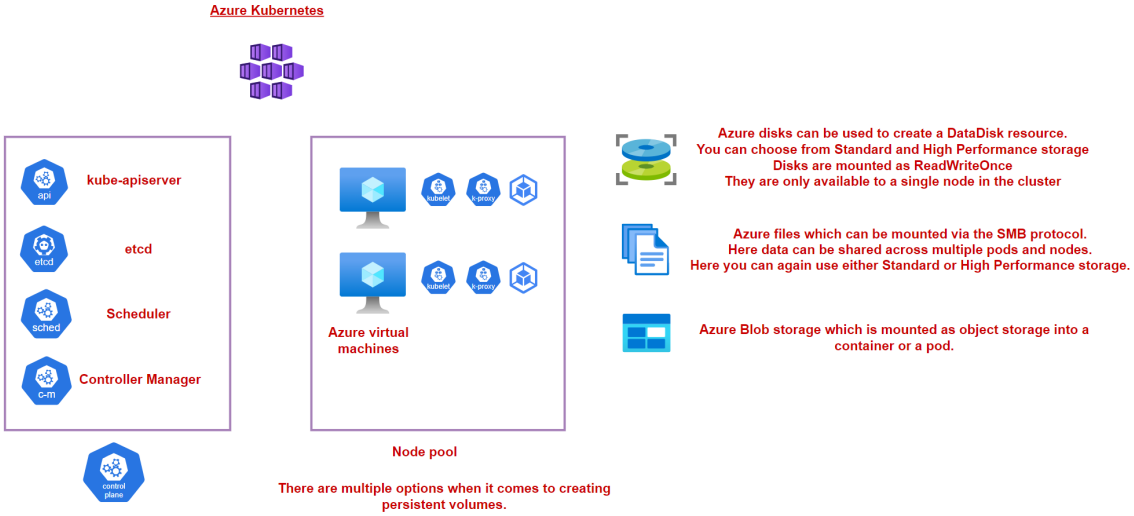
The service is used to specify the desired number of tasks that need to run in the Amazon ECS cluster.

Azure Kubernetes Service



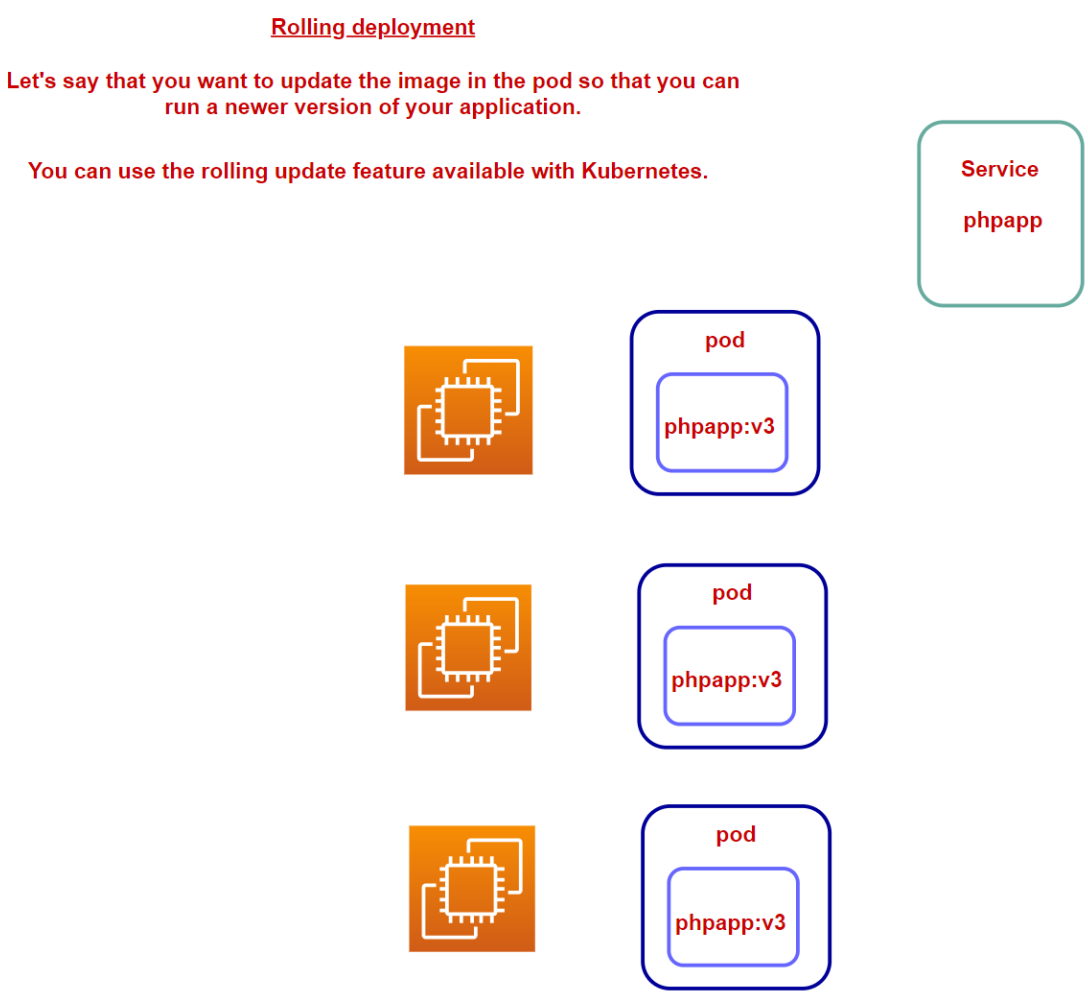


AKS- Lab- Using Disks for storage



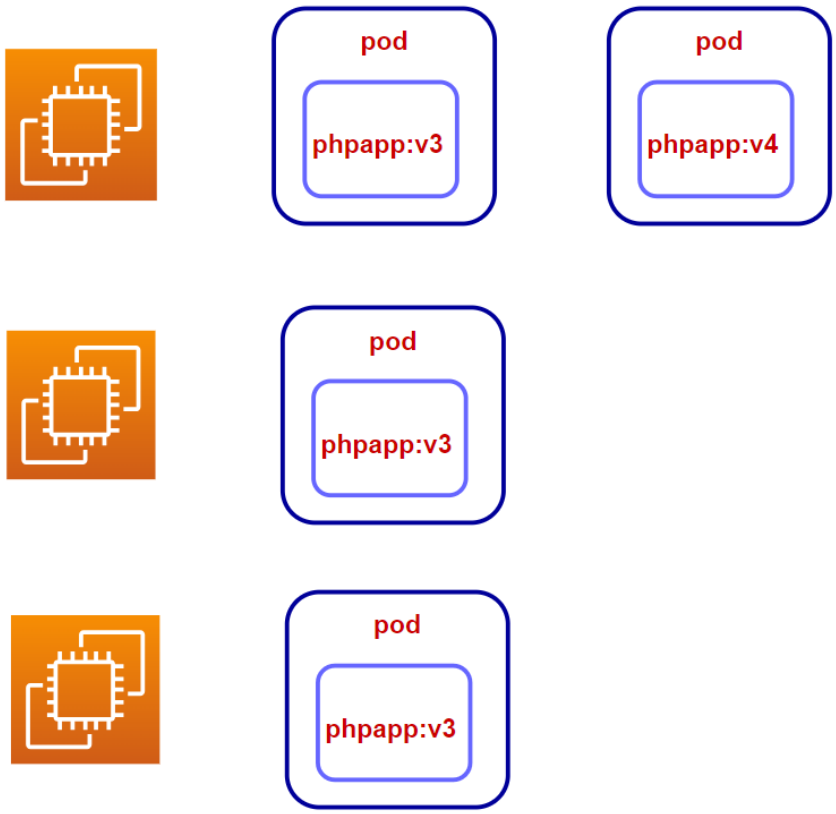
Kubernetes – Learning further aspects

Lab- Performing a rolling update

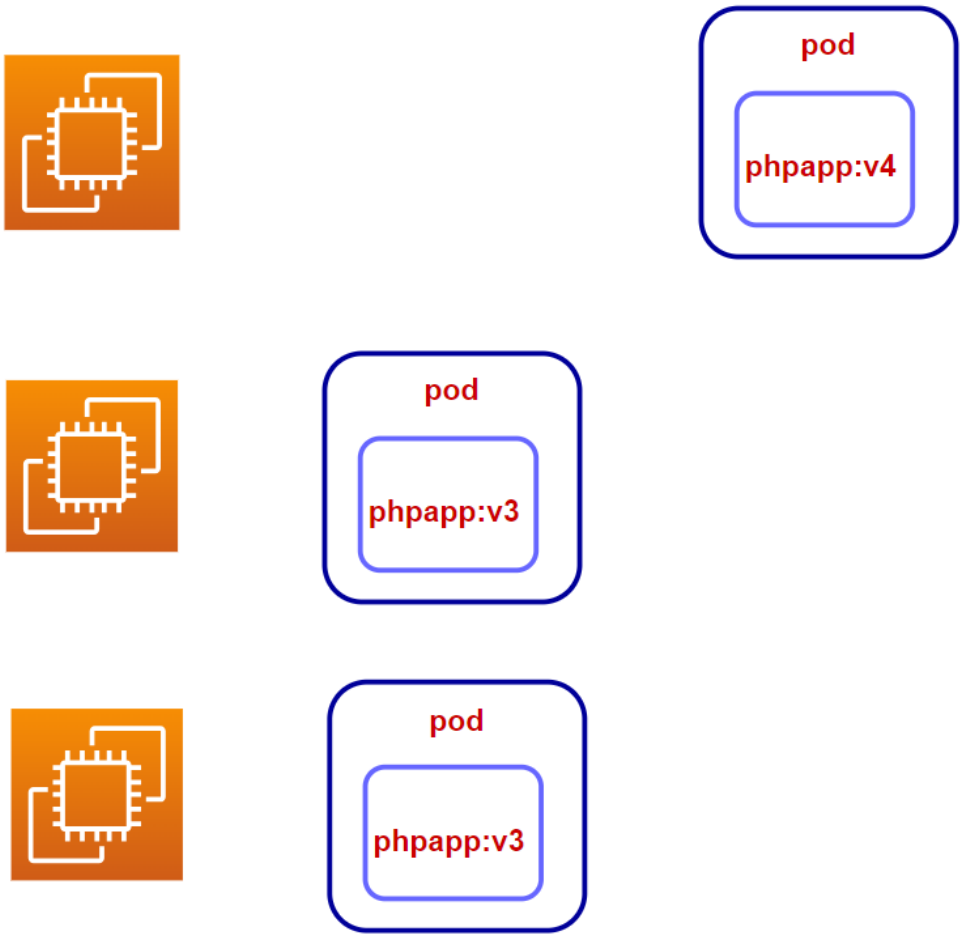


Here Kubernetes will gradually ensure the new deployments are carried out.

First Kubernetes will deploy a pod with the newer image version



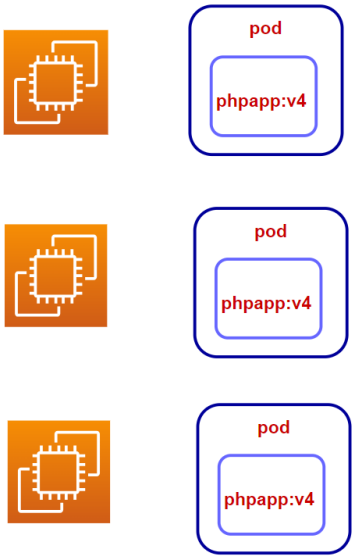
Then it will delete the pod with the older image version



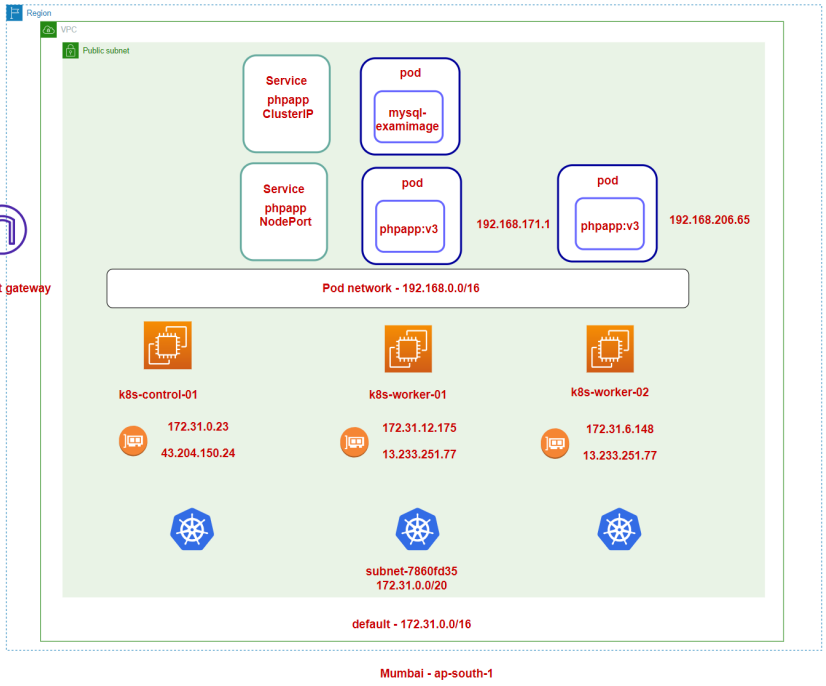
It will gradually ensure that all nodes run new pods with the newer image version

This helps to ensure there is no downtime for your application when updates are made.

It also allows you to roolback easily to the prior version if the newer version fails for any reason.

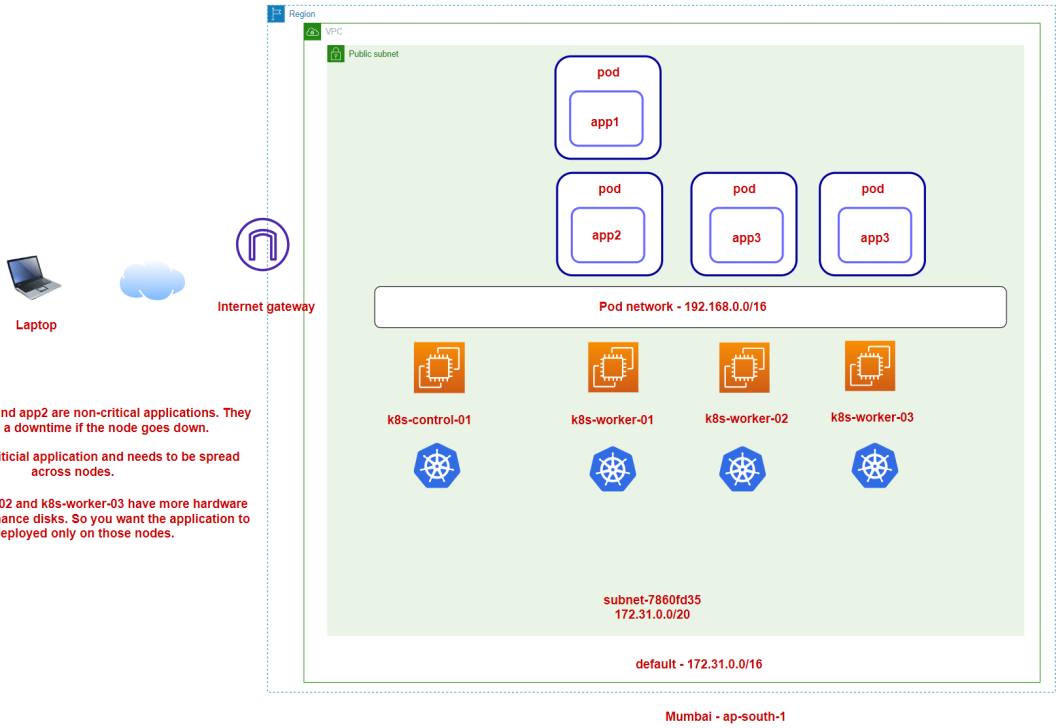


Lab- Using a node selector



Currently Kubernetes will assign pods to nodes based on resource availability. In our case it was making sure the pods are placed on different nodes.

But you could also make sure that pods are only deployed on certain nodes in the cluster.



Scenario - app1 and app2 are non-critical applications. They can afford a downtime if the node goes down.

But app3 is a criticial application and needs to be spread across nodes.

And k8s-worker-02 and k8s-worker-03 have more hardware and high performance disks. So you want the application to be deployed only on those nodes.

Lab- init containers

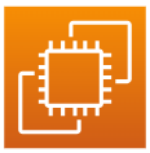
Init containers

These are containers that run before the application containers in the pod.

You can have definitions for multiple init containers in the pod.

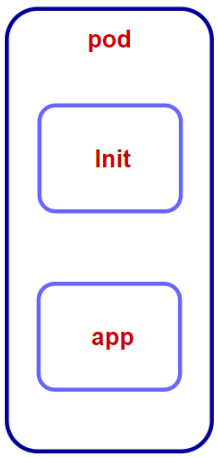
The init container always runs to completion.

If the init container fails and the restartPolicy is Never then the pod is deemed as failed.



Control Node

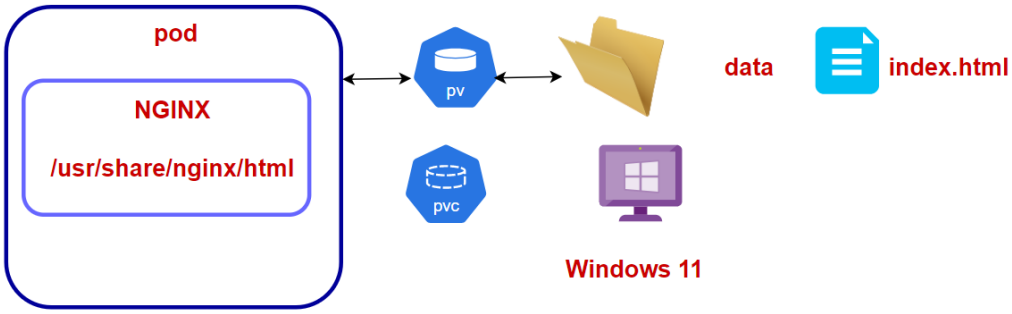
Worker nodes



You could have the use case of an init container carrying out some startup tasks that may be needed for the application container.

You could have the init container fetch secrets that would be required by the application.

What are we going to implement



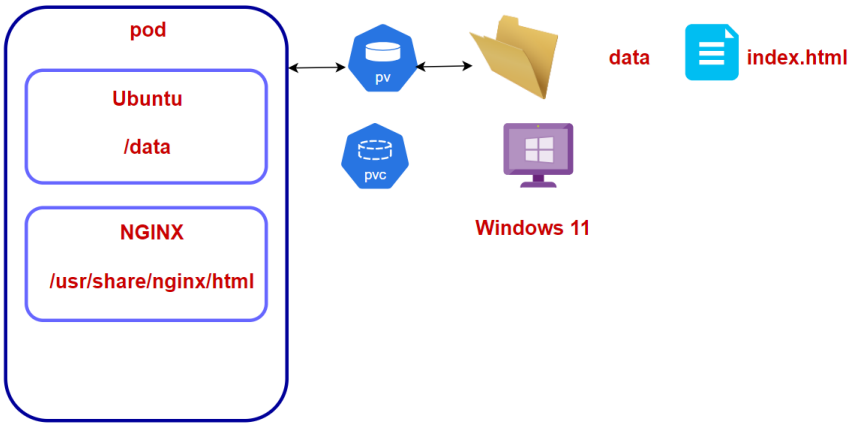
We had a Pod running the NGINX container.

The Pod had a persistent volume that was mapped to a folder on my local machine. This was mapped to the html folder for NGINX.

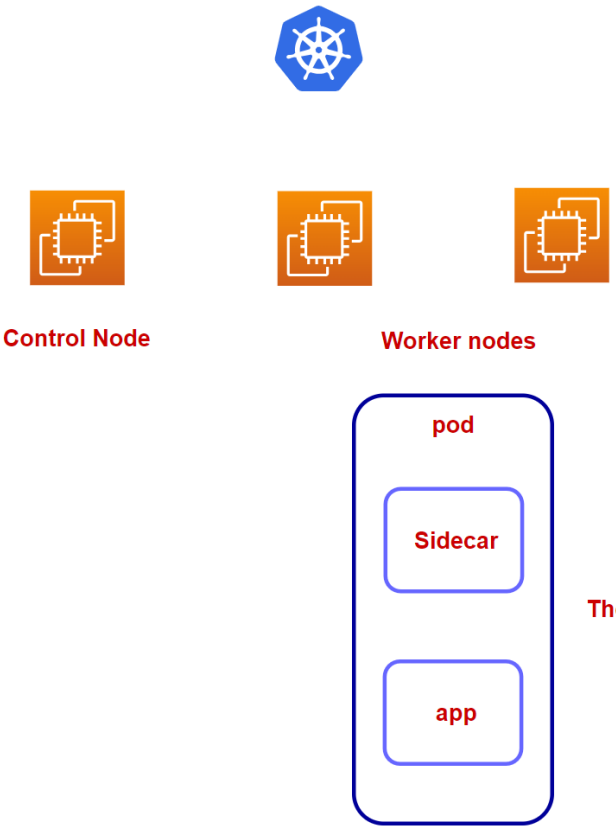
We then manually in some form or the other copied the file.

Init container

The init container will create the index.html file for us.



Lab- sidecar containers



Probes in Kubernetes

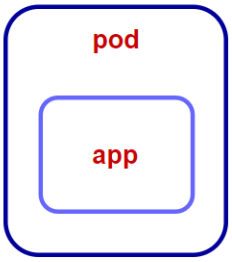
Control Node

Worker nodes

There are times wherein your application stops working.

Let's say that the application is connecting to a database and there is a network issue.

The application might crash and require a restart.



A liveness probe can continuously carry out a check for the application running in the container.

If the check fails then Kubernetes will restart the failed container.

For the liveness probe, you can define different types of checks.

command - Here you can execute a command on the underlying container. If the command succeeds then it means the container is running as expected.

HTTP request - If you are running a web server, you can issue HTTP requests to ensure the container is running as expected.

TCP request - You can configure a TCP check on a particular port to ensure that the container is running as expected.

gRPC probe to check the running container.

Settings

initialDelaySeconds - This tells kubelet to wait until it performs the first check.

periodSeconds - This specifies the interval after which kubelet will perform the check.

timeoutSeconds - The time to wait for a reply.

successThreshold - The number of successful probe requests that would deem the container as healthy.

failureThreshold - The number of failed probe requests that would deem the container as unhealthy.