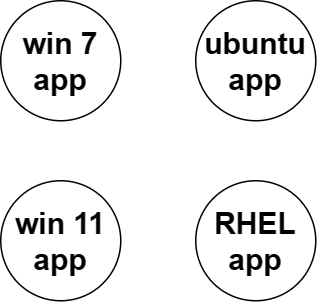
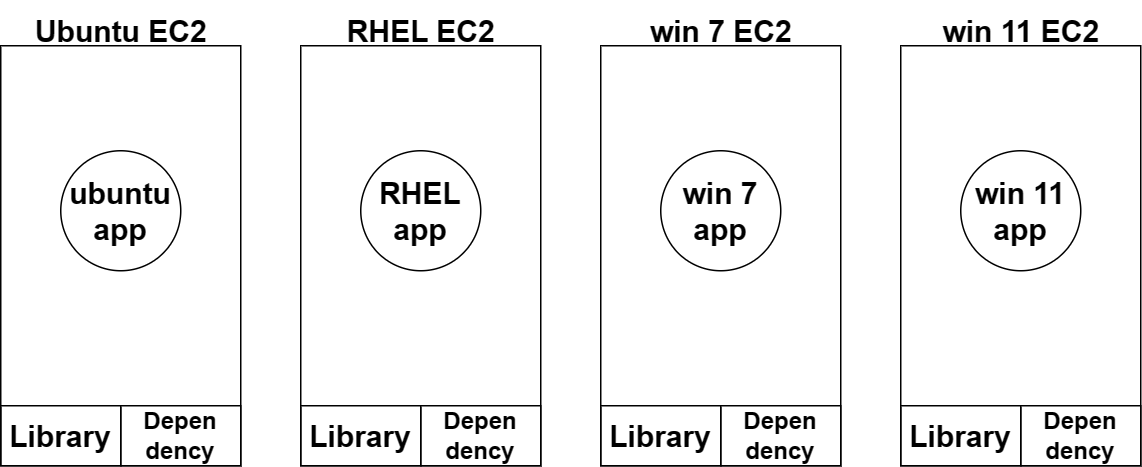
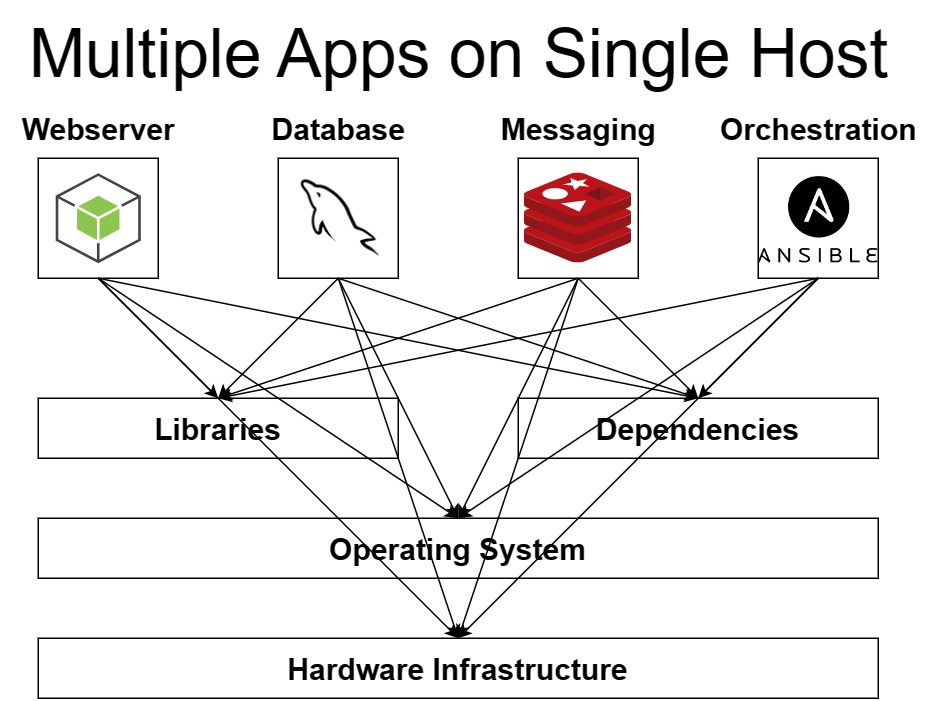
Problem Statement

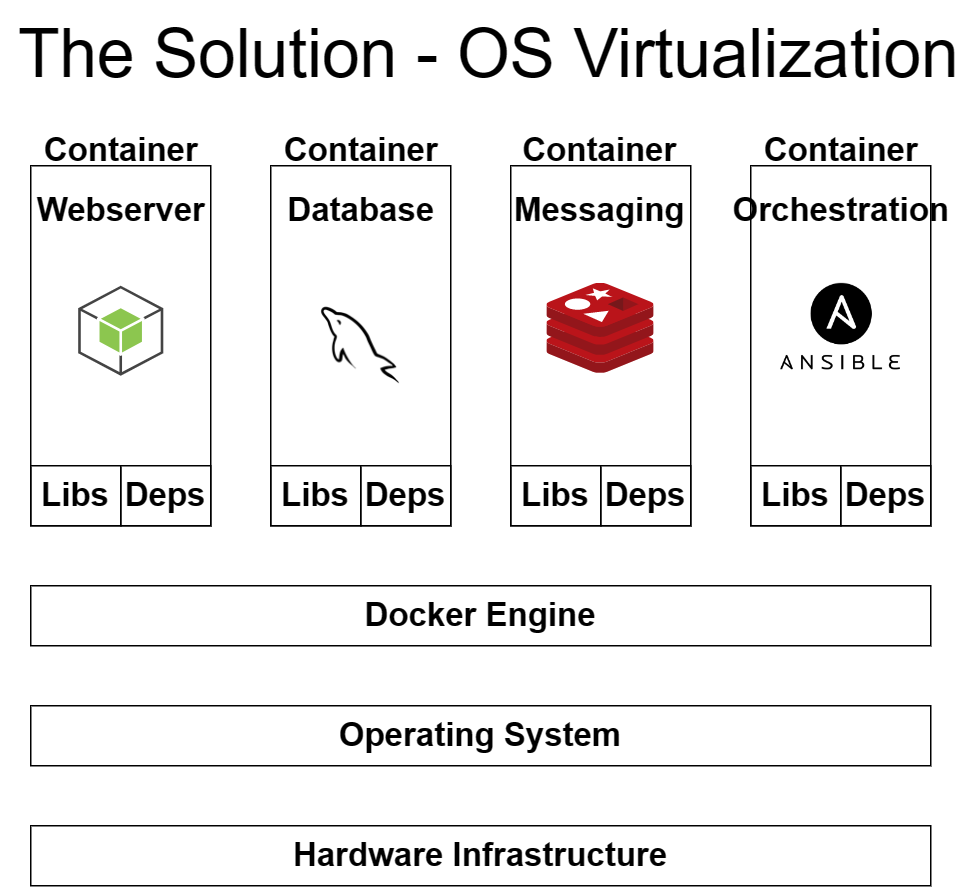


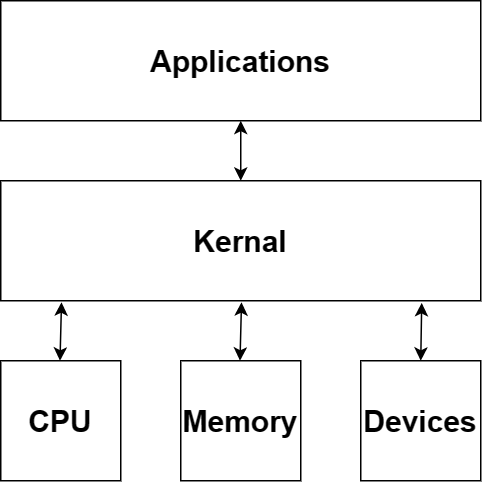
# Solution with VM

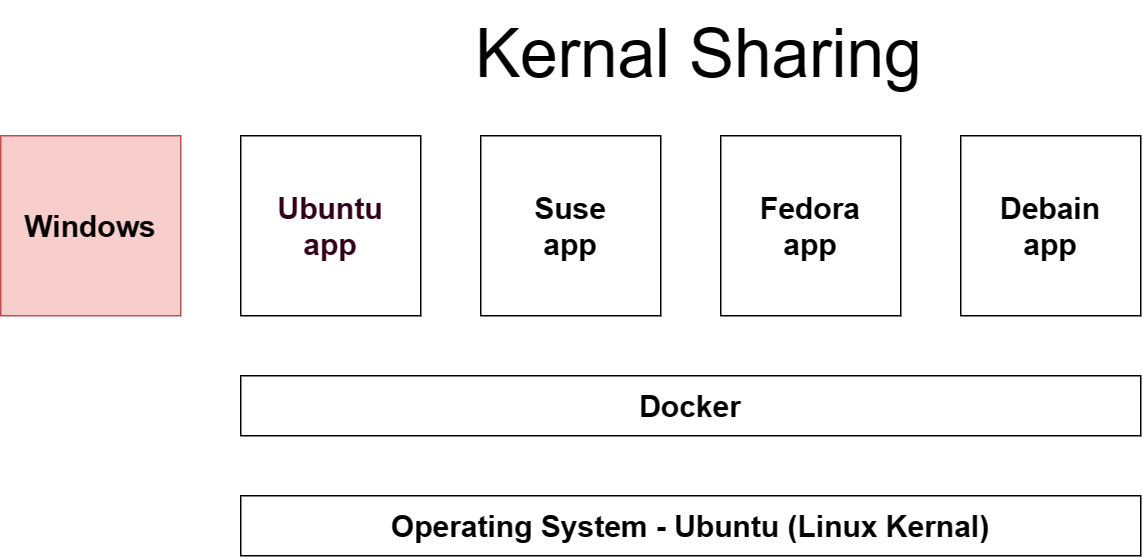


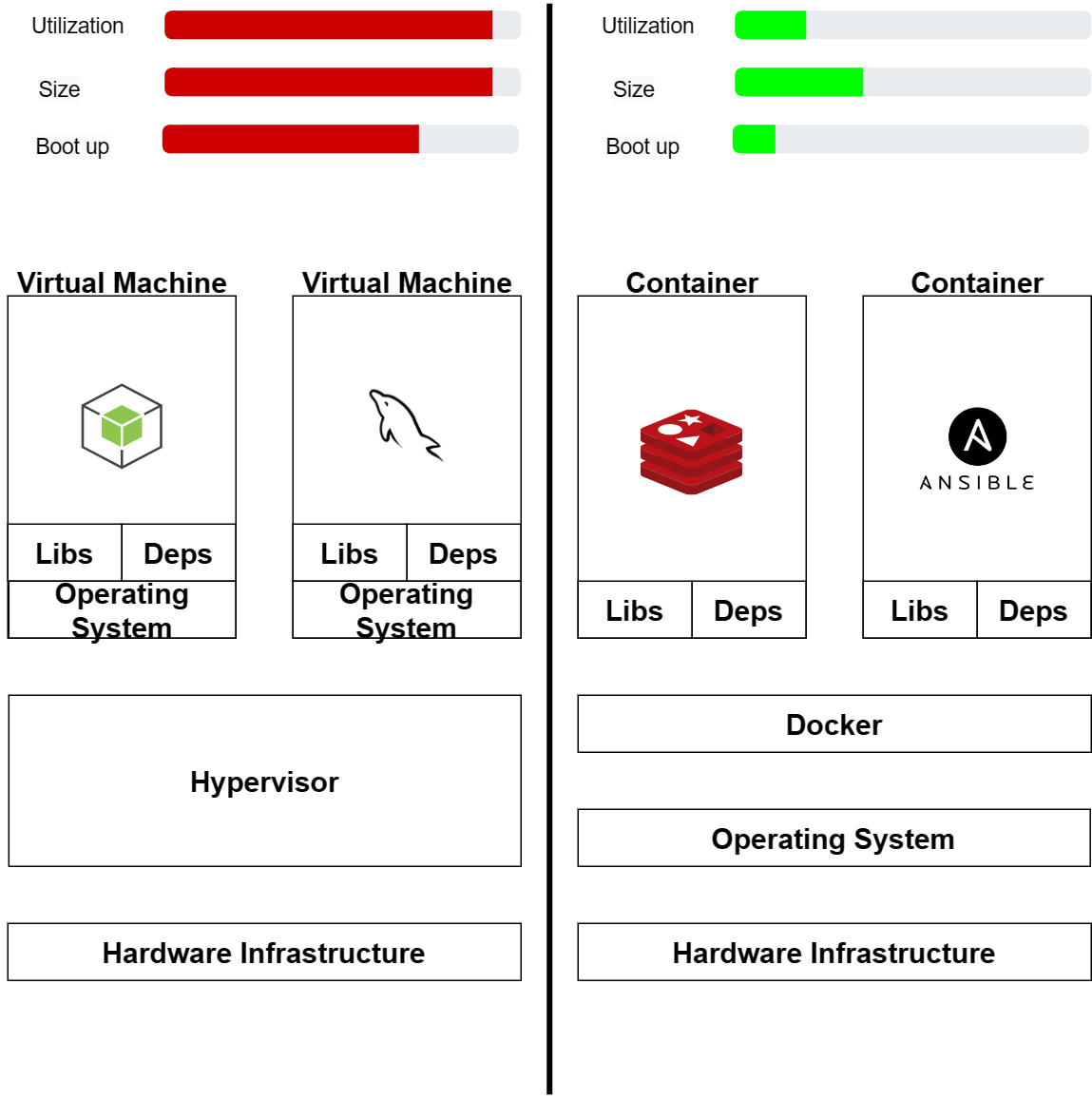
Dependency Issue

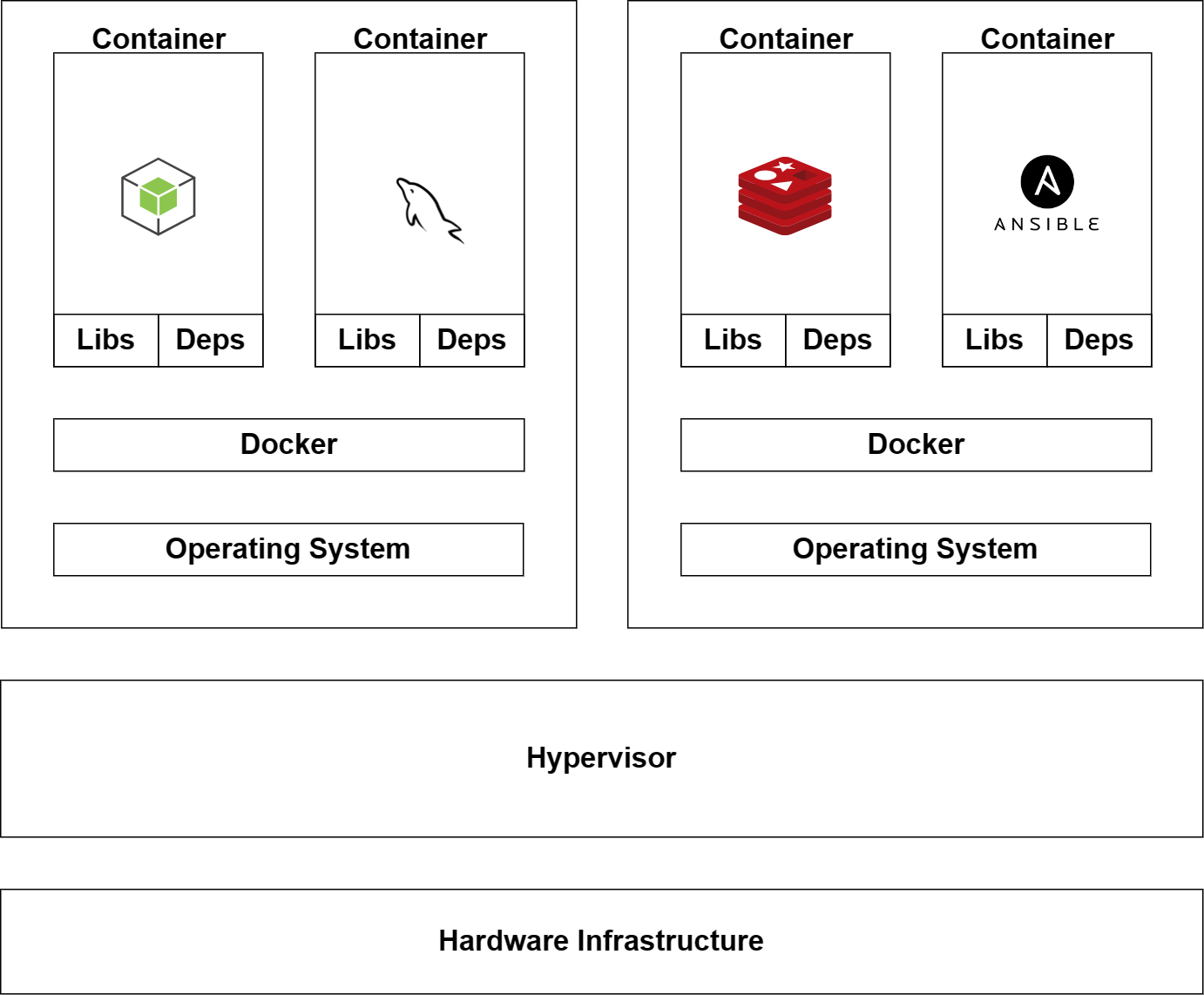
Whats a Container ? 

Whats a Container ? 

Whats a Kernal ? 

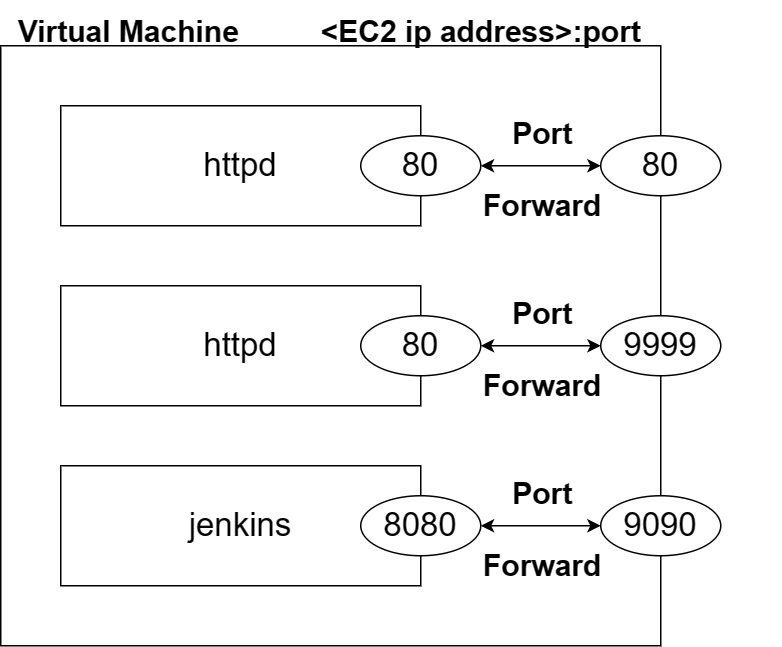
Kernal Sharing ! 

VM vs Docker

Actually its both Virtualization & Containerization

Docker

1. Create a Ubuntu EC2 in Amazon Console.
2. Install docker
   1. ssh to the Server using putty
   2. sudo su - root
   3. apt-get update
   4. apt install docker.io
3. Docker commands to interact with docker daemon
   1. docker images
   2. docker ps -a
   3. docker pull httpd
   4. docker images
   5. docker run -itd -p "80:80" httpd (“hostport:containerport”)
   6. hit the ec2 public ip address in browser
4. How Port forwarding works



1. How to get inside the container
   1. docker exec -it <container id> /bin/bash
   2. ls -l
   3. cd htdocs
   4. cat index.html
   5. echo "Hello Srm 1st container" > index.html
   6. exit
   7. hit the ec2 public ip address in browser
2. Create more containers on port forwarding
   1. docker run -itd -p "9999:80" httpd
   2. docker ps -a
   3. Hit the <ec2 public ip>:9090 address in browser
   4. Check the Security group and add the 9090 port in inbound rule or open all port.
   5. docker run -itd -p "9090:8080" jenkins/jenkins:lts
   6. docker stop <container id>
   7. docker rm <container id>
3. Creating Customized Image
   1. vi index.html > add some content & ESC :wq
   2. create a Dockerfile

FROM httpd

COPY ./index.html /usr/local/apache2/htdocs

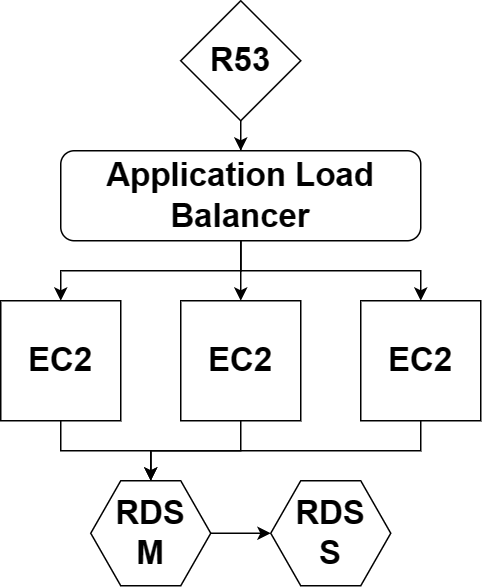
* 1. docker build -t myownimage:v1 /root/
  2. docker run -itd -p "80:80" myownimage:v1

1. Uploading to Dockerhub
   1. Create a Dockerhub account to store the your images
   2. docker login > give your Dockerhub Username & Password
   3. docker tag myownimage:v1 iammithran/myownimage:v1 ( “iammithran” is Dockerhub id )
   4. docker images
   5. docker push iammithran/myownimage:v1

Why do we need Kubernetes

Docker commands got Executed in a single server.  
If the Server Crashes All the containers inside will be Terminated.  
We reply on Multiple Server and deploy our Containers on top of them.  
And to Mange these Server we bring in Kubernetes as a Tool aka Container Orchestration Tool

Monolithic Architecuture

In a Nutshell: If Code is deployed directly to the Server we call it as Monolithic Architecture. 

MicroService Architecture

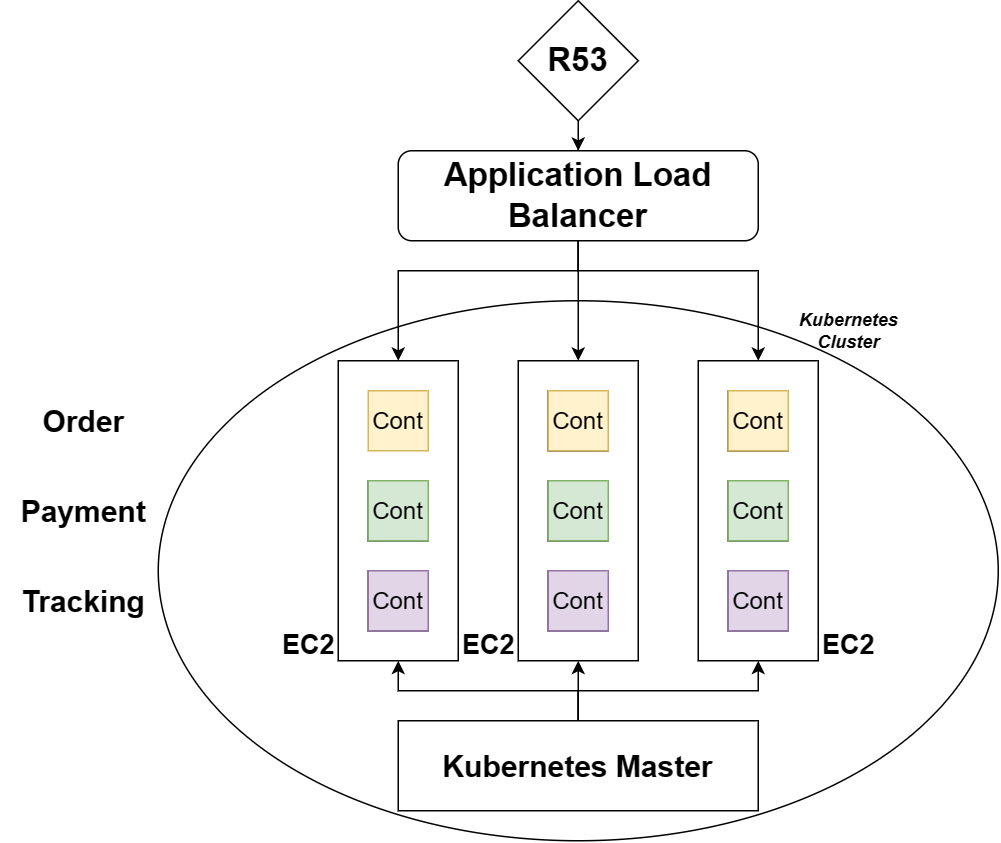
When the Application is Huge it takes multple Development Teams to work and bring in a single application. For example sbi.com  
Lets assume In Zomato application with Monolithic Architecture

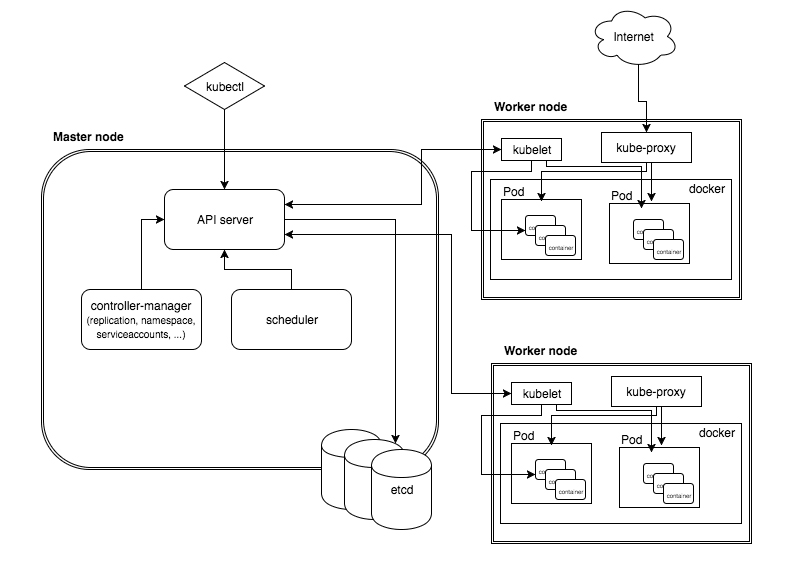
2014 -> 100,000 lines of code -> 1gb .war file  
2016 -> 200,000 lines of code -> 2gb .war file  
2018 -> 500,000 lines of code -> 5gb .war file  
2020 -> 800,000 lines of code -> 8gb .war file  
2022 -> 1,000,000 lines of code -> 10gb .war file

With the Introduction to Docker and Kubernetes we break down the Application into Smaller MicroServices.

2022 -> 1,000,000 lines of code -> 10gb .war file

* Order 400,000 -> 4gb .war file -> Docker Image -> Docker Container
* Payment 300,000 -> 3gb .war file -> Docker Image -> Docker Container
* Tracking 300,000 -> 3gb .war file -> Docker Image -> Docker Container



Kubernetes Architecture

Kubernetes Installation

# Installation Methods

1. Single-node installation: minikube
2. Manual cluster installation: kubeadm
3. Automatic cluster installation: kops
4. Managed clusters: EKS, AKS, GKE

# KOPS PRE-REQUIREMENTS

1. linux machine (ubuntu)
2. AWS account
3. kops binary (kubernetes cluster initiate)

$ curl -Lo kops https://github.com/kubernetes/kops/releases/download/$(curl -s https://api.github.com/repos/kubernetes/kops/releases/latest | grep tag\_name | cut -d '"' -f 4)/kops-linux-amd64

$ chmod +x ./kops

$ sudo mv ./kops /usr/local/bin/

1. kubectl binary (kubernetes deployments)

$ curl -Lo kubectl https://storage.googleapis.com/kubernetes-release/release/$(curl -s <https://storage.googleapis.com/kubernetes-release/release/stable.txt)/bin/linux/amd64/kubectl>

$ chmod +x ./kubectl

$ sudo mv ./kubectl /usr/local/bin/kubectl

1. IAM User for Cloud Authorization Kubernetes

Create IAM User from the Console with Security Credentials on Admin Access

aws configure

export AWS\_ACCESS\_KEY\_ID=<insert key here>

export AWS\_SECRET\_ACCESS\_KEY=<insert key here>

1. S3 Bucket to Store the State File from the Console with bucket versioning turned ON
2. Public & Private Key Pair for Node Authentication

ssh-keygen

1. Cluster Name and S3 Bucket Name as Env Variables

export NAME=<insert cluster name>.k8s.local

export KOPS\_STATE\_STORE=s3://<insert bucket name>

1. Create a Cluster.

kops create cluster --zones ap-south-1a ${NAME}

kops edit ig --name=<insert cluster name>.k8s.local nodes-ap-south-1a (Optional)

kops update cluster --name <insert cluster name>.k8s.local --yes –admin

Kubernetes

kubectl get pods

vi pod.yaml

apiVersion: v1

kind: Pod

metadata:

  name: httpd-pod

spec:

  containers:

  - name: cont1

    image: httpd

    ports:

      - containerPort: 80

kubectl apply -f pod.yaml

kubectl get pods

kubectl delete pod httpd-pod

vi deployment.yaml

apiVersion: apps/v1

kind: Deployment

metadata:

  name: my-deployment

spec:

  replicas: 3

  selector:

    matchLabels:

      app: webserver

  template:

    metadata:

      labels:

        app: webserver

    spec:

      containers:

      - name: cont1

        image: httpd

        ports:

          - containerPort: 80

kubectl apply -f deployment.yaml

kubectl set image deployment/my-deployment cont1=nginx

vi service.yaml

apiVersion: v1

kind: Service

metadata:

  name: loadbalancer-svc

spec:

  type: LoadBalancer

  selector:

    app: webserver

  ports:

    - name: http

      protocol: TCP

      port: 80

kubectl apply -f service.yaml