



KUBESIMPLIFY
CLOUDNATIVE FOR EVERYONE

VISUAL GUIDES COLLECTION



@kubesimplify



kubesimplify.com



@kubesimplify

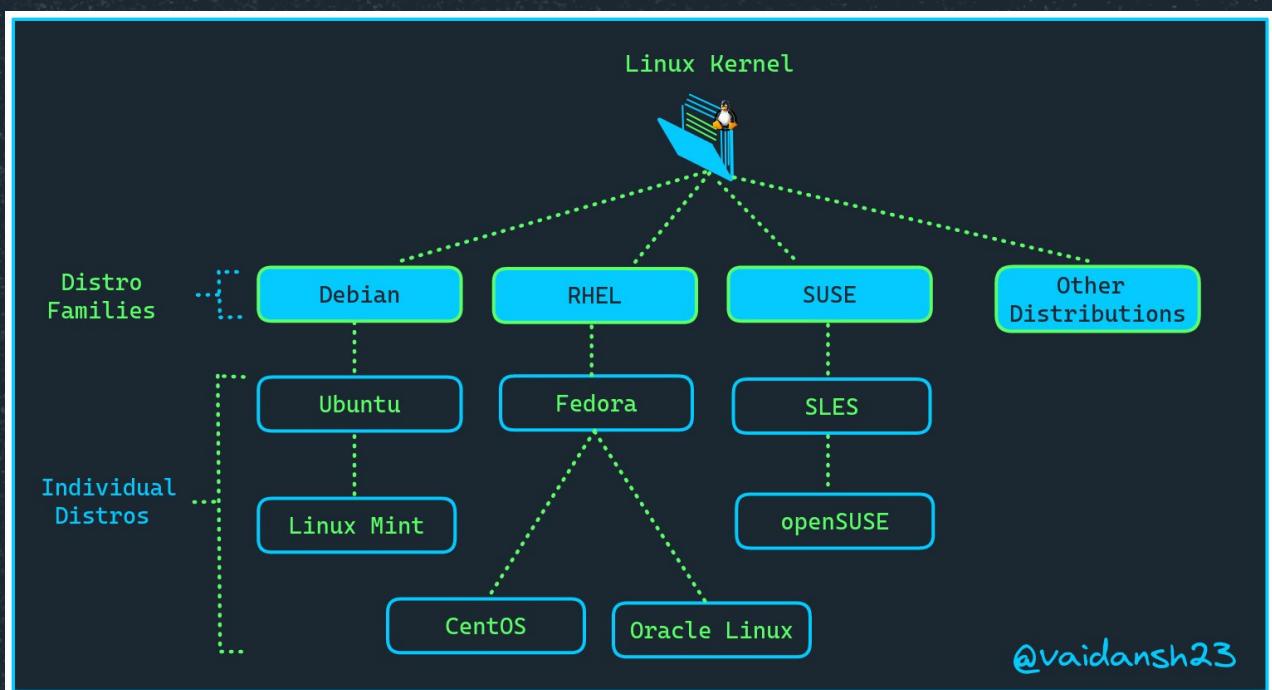
Table Of Contents

- 01 Linux
- 02 Kubernetes
 - 2.1 Kubernetes Series
- 03 CI/CD
- 04 Containerization & VMs
- 05 DevOps - Overview
- 06 Cloud Service Models
- 07 Hypervisor
- 08 Networking
- 09 Prometheus
- 10 Service Meshes
- 11 Monolithic V/s Microservices

Linux



Linux Distributions

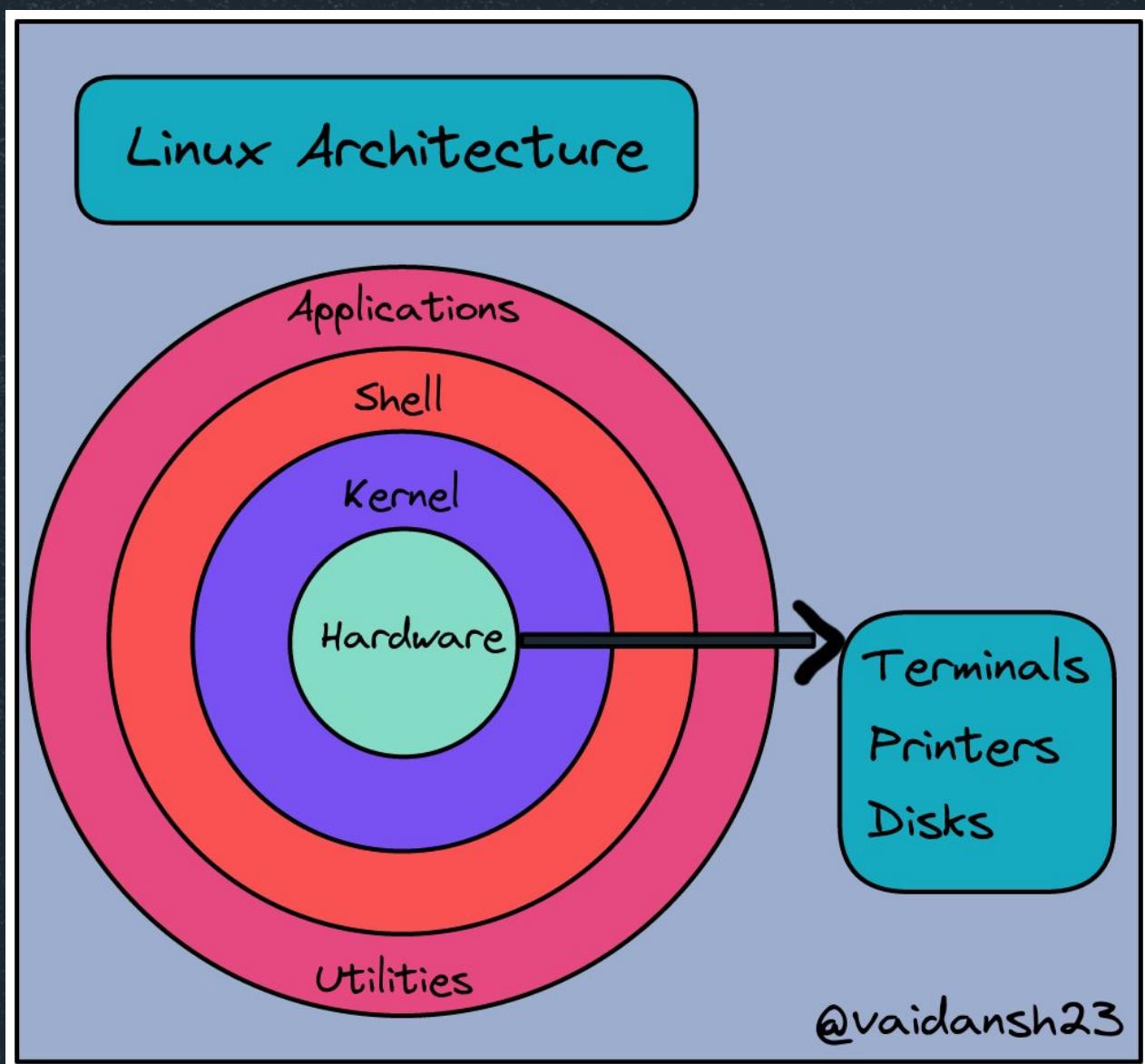


~Vaidansh Bhardwaj

Linux



Linux Architecture



~Vaidansh Bhardwaj

Linux



Linux Commands

● Basic Linux Commands

- sudo - superuser do, used for running as root.
- mkdir - used to create a folder in our system.
- cd - allows us to move into the directory.
- rmdir - used to remove the directory.
- pwd - print working directory, displays your current location.
- touch - used to create a new file.
- mv - allows to move the files and also used for renaming.
- cp - used to copy files from one folder to another.
- cat - short for concatenation, used to see the content inside the file.
- ls - Lists all files and directories in the present working directory.

Linux Commands

- clear - It will clear the screen of all previous commands.
- man - short for manual, used to understand any specific command.
- grep - allow us to search in the file for a specific word.
- history - Gives a list of all past commands typed in the current terminal session.
- cd .. - Go up a directory.

● File Permission Commands

0 = None ---
1 = Execute only --x
2 = Write only -w-
3 = Write & Execute -wx
4 = Read Only r--
5 = Read & Execute r-x
6 = Read & Write rw-
7 = Read, Write & Execute rwx

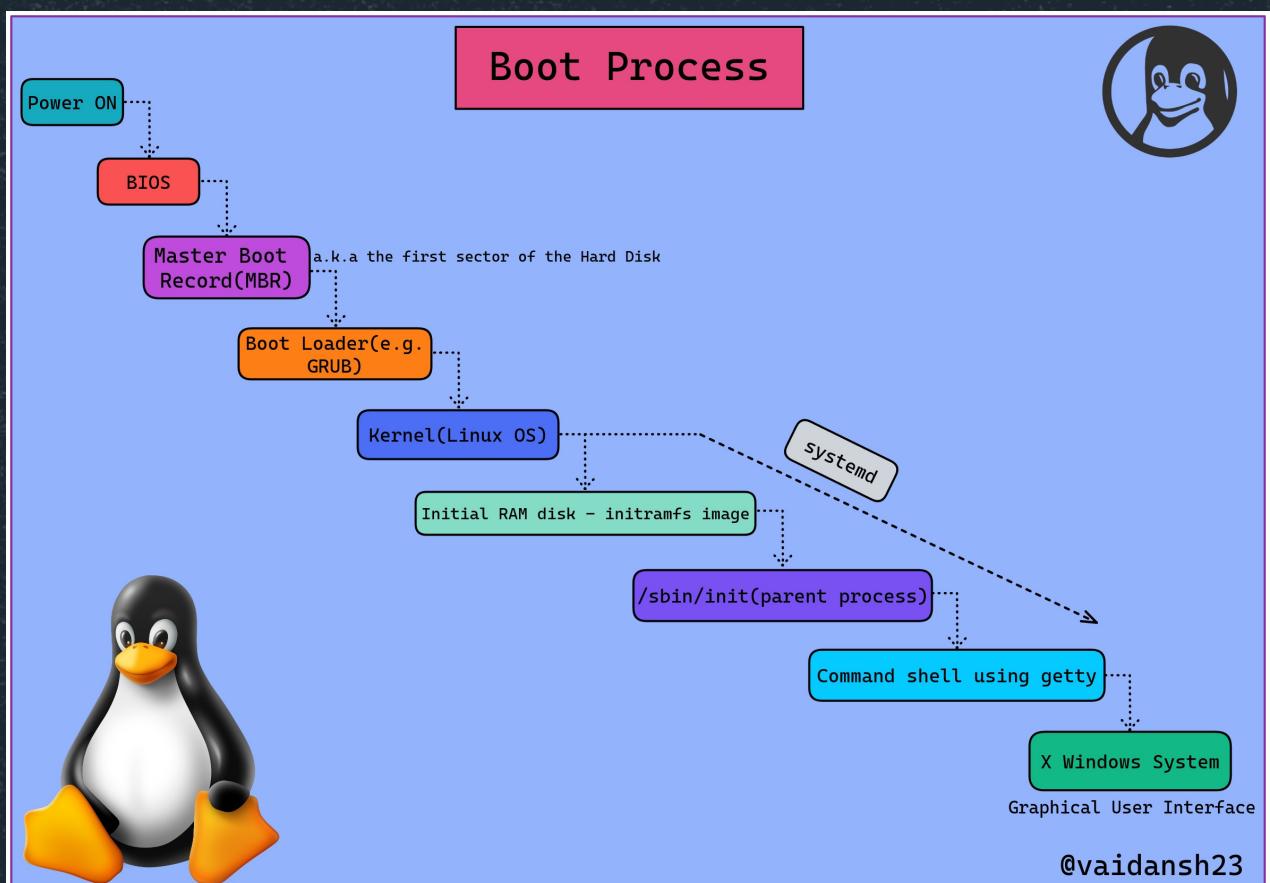
- ls -l - shows file type and access permission.
- chmod u=rwx,g=rw,o=r file
 - setting explicit permissions for different groups.
- chown user - changing the ownership of a file/directory.
- Environment Variables Commands
- echo \$VARIABLE - used to display the value of a variable.
- env - Displays all environment variables.
- VARIABLE_NAME= variable_value
 - Creates a new variable.
- Unset - Remove a variable.
- export Variable=value - To set value of an environment variable.

~Harshita Rao

Linux



Linux Boot Process



~Vaidansh Bhardwaj

Kubernetes



Kubernetes Series

A Series of diagrams from the [Kubernetes 101 Workshop](#) by [Saiyam Pathak!](#)

Here's the link to the workshop:

<https://www.youtube.com/watch?v=PN3VqbZqmD8>

Kubernetes



Kubernetes Series: #1 Intro

What is Kubernetes ?

Kubernetes, also known as K8s, is an open-source system for automating deployment, scaling, and management of containerized applications.

- CNCF Graduated
- Born out of Borg and Omega
- Launched 2014
- Designed for Scale
- Run anywhere

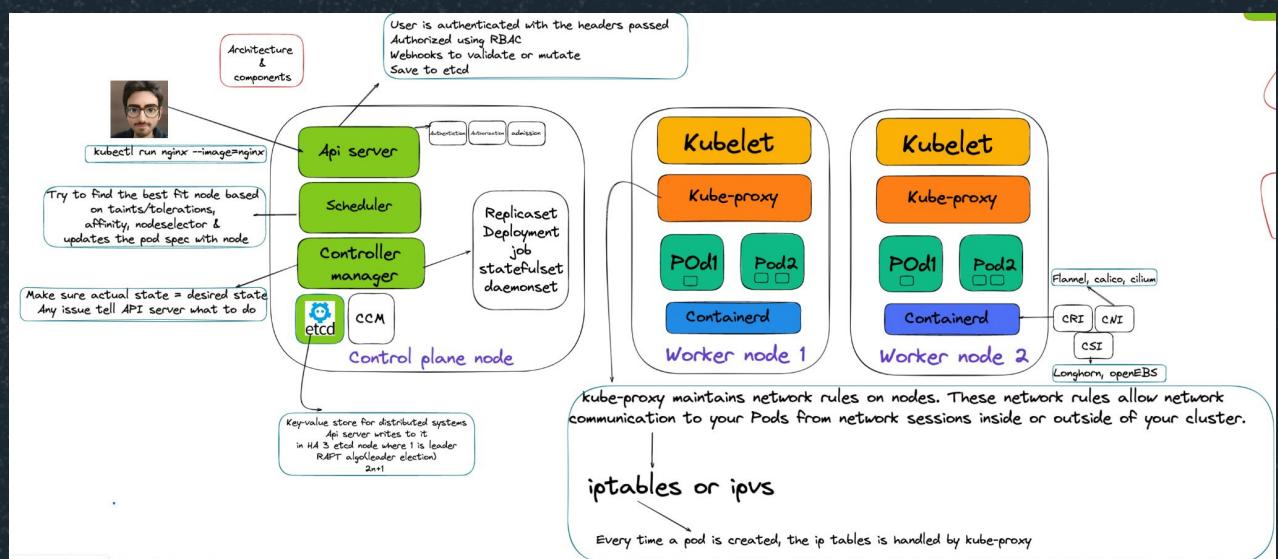
Why Kubernetes ?

- Few containers are fine but what about scale ?
- How do monitor them ?
- What about running pods on multiple nodes?
- What about flexibility ?
- Autoscaling?
- Scheduling
- Self healing capabilities.

Kubernetes



Kubernetes Series: #2 Architecture & Components



Kubernetes



Kubernetes Series: #3 YAML File

The diagram illustrates the structure of a YAML file with various annotations:

- YAML**: A box containing a Pod example.
- Pod example**:
 - Key: value**: Annotations for the Pod.
 - Object**: The Pod itself.
 - attributes of the objects**: Annotations for the containers.
 - List**: Annotations for the resources.
 - List items**: Annotations for the dnsPolicy and restartPolicy.
- Placeholder**: Annotations for environment variables.
- Separate different objects using this**: Annotations for deployment and service objects.
- Code Examples**:
 - A terminal session showing a certificate chain for a client.
 - A terminal session showing a multi-line string using the pipe symbol (|).

```
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    run: nginx
    name: nginx
  spec:
    containers:
      - image: nginx
        name: nginx
        resources: {}
    dnsPolicy: ClusterFirst
    restartPolicy: Always
    status: {}
```

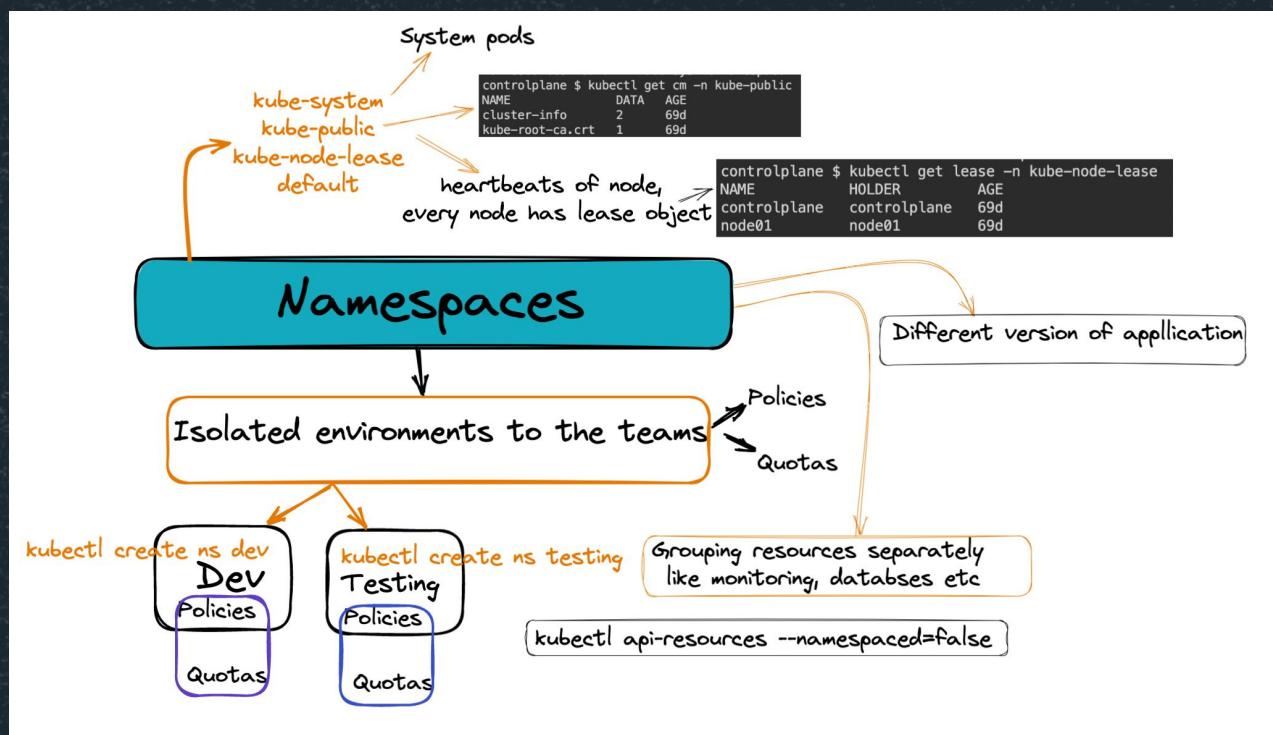
```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: helloservice
spec:
  selector:
    matchLabels:
      app: helloservice
  template:
    metadata:
      labels:
        app: helloservice
    spec:
      terminationGracePeriodSeconds: 5
      containers:
        - name: server
          image: sayiam911/argocd-demo:{{ image_deploy_tag }}
          imagePullPolicy: Always
          ports:
            - containerPort: 8080
          env:
            - name: PORT
              value: "8080"
```

```
apiVersion: v1
kind: Service
metadata:
  name: helloservice
spec:
  selector:
    app: helloservice
  ports:
    - name: http
      port: 80
      protocol: TCP
      targetPort: 8080
  type: NodePort
```

Kubernetes



Kubernetes Series: #4 Namespaces



Kubernetes



Kubernetes Series: #5 Labels & Selectors



Kubernetes



Kubernetes Series: #6 PODS

The diagram illustrates a Pod structure within a Node. The Node contains a Pod IP address and a Pod named 'nginx'. The Pod contains a single container named 'nginx' with a volume labeled 'c1'. A red box highlights the 'nginx' name in both the container and the Pod specification. Red arrows point from these highlighted names to the corresponding 'name: nginx' entries in the provided YAML manifest.

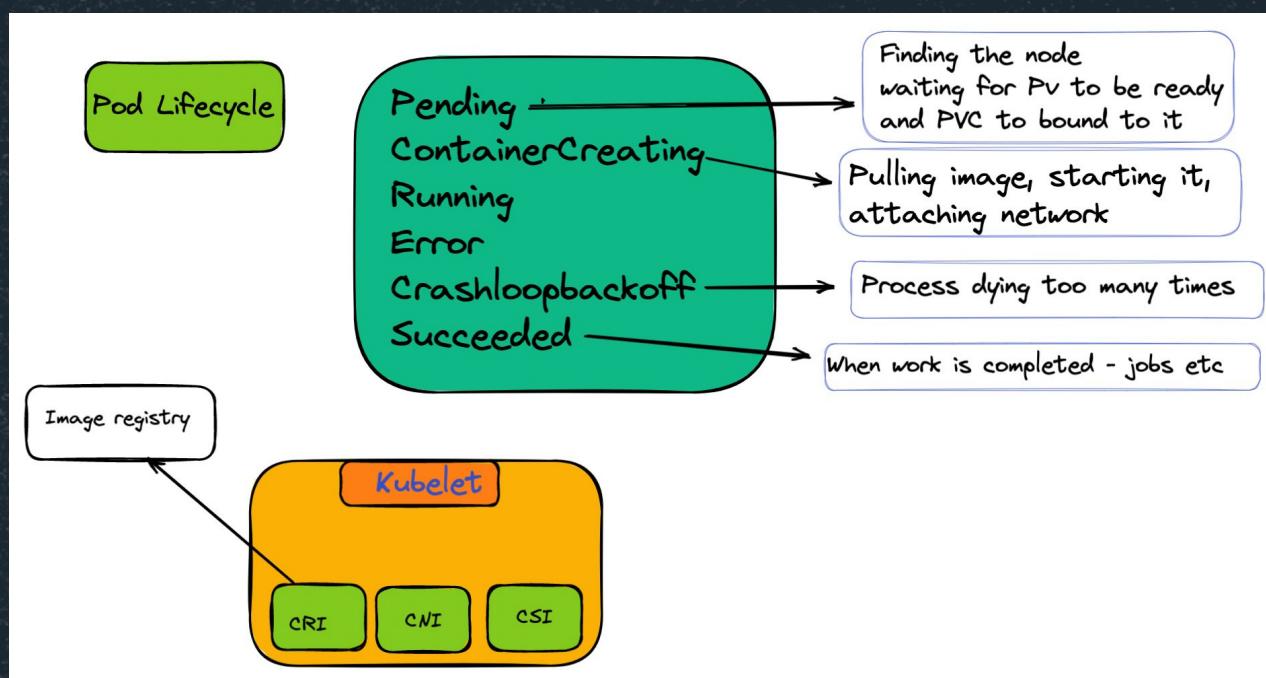
```
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    run: nginx
  name: nginx
spec:
  containers:
  - image: nginx
    name: nginx
  ports:
  - containerPort: 80
  resources: {}
  dnsPolicy: ClusterFirst
  restartPolicy: Always
status: {}
```

create, describe, logs, delete, exec, wide

Kubernetes



Kubernetes Series: #7 PODS Lifecycle



Kubernetes



Kubernetes Series: #8 INIT Container

init container →

Runs before the main container

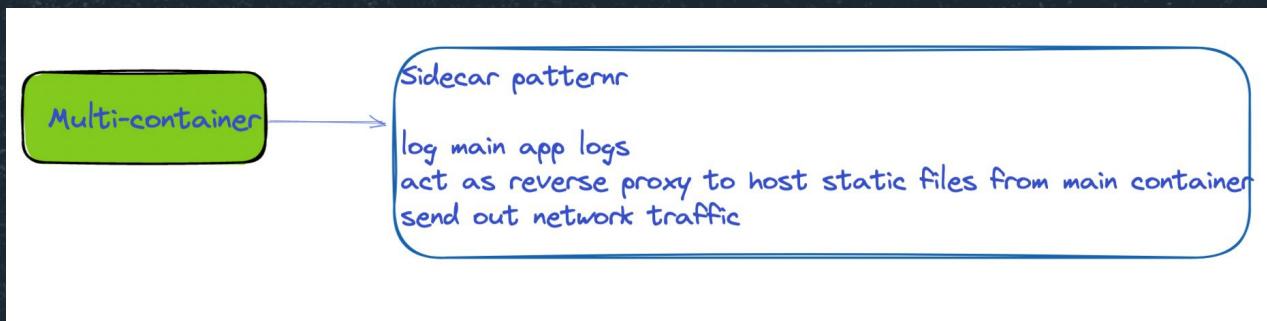
- can contain the custom code that is not present in the app
- change filesystem based on certain login before the main container starts
- pre condition checks
- run in sequential order

Events:				
Type	Reason	Age	From	Message
Normal	Scheduled	87s	default-scheduler	Successfully assigned default/init-demo1 to node01
Normal	Pulling	86s	kubelet	Pulling image "busybox"
Normal	Pulled	86s	kubelet	Successfully pulled image "busybox" in 462.914086ms
Normal	Created	85s	kubelet	Created container install
Normal	Started	85s	kubelet	Started container install
Normal	Pulling	85s	kubelet	Pulling image "nginx"
Normal	Pulled	80s	kubelet	Successfully pulled image "nginx" in 4.683285911s
Normal	Created	80s	kubelet	Created container nginx
Normal	Started	80s	kubelet	Started container nginx

Kubernetes



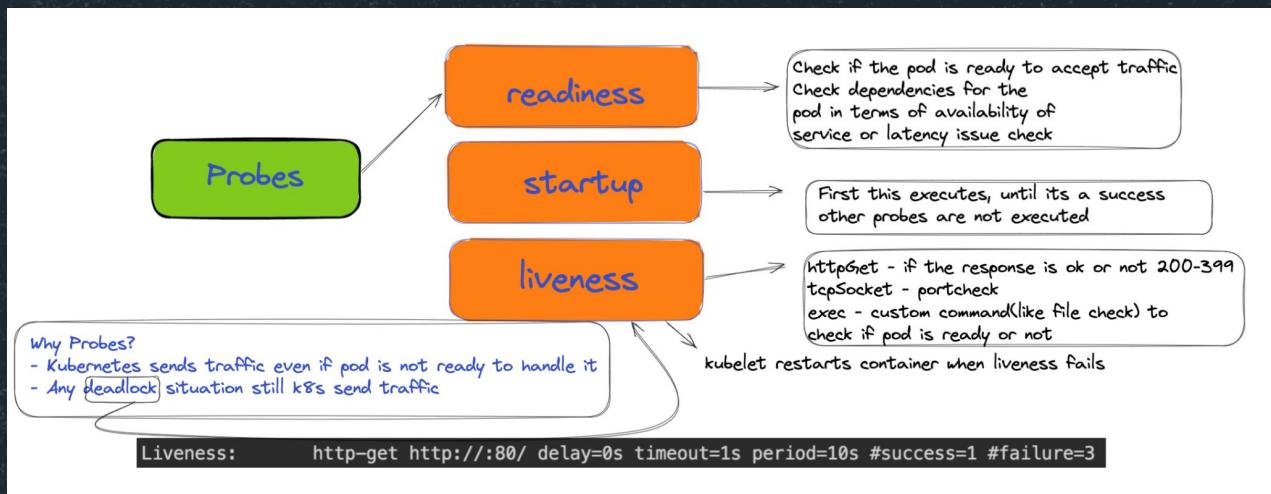
Kubernetes Series: #9 Multi-Container



Kubernetes



Kubernetes Series: #10 Probes



Kubernetes



Kubernetes Series: #11 Deployment

The diagram illustrates the creation of a deployment. A central box labeled "Deployment" has two arrows pointing to it: one from the text "Also create RS" and another from the text "create 2 pods". Below the deployment box are two smaller boxes labeled "1" and "2", representing the two pods it creates. The text "kubectl rollout status deployment name" is at the top right.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  creationTimestamp: null
  labels:
    app: nginx
    name: nginx
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  strategy: {}
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: nginx
    spec:
      containers:
        - image: nginx
          name: nginx
          resources: {}
status: {}
```

controlplane \$ kubectl get pods

NAME	READY	STATUS	RESTARTS	AGE
nginx-8f458dc5b-8fcfkf	1/1	Running	0	8s
nginx-8f458dc5b-dzphw	1/1	Running	0	8s

StrategyType: RollingUpdate
MinReadySeconds: 0
RollingUpdateStrategy: 25% max unavailable, 25% max surge

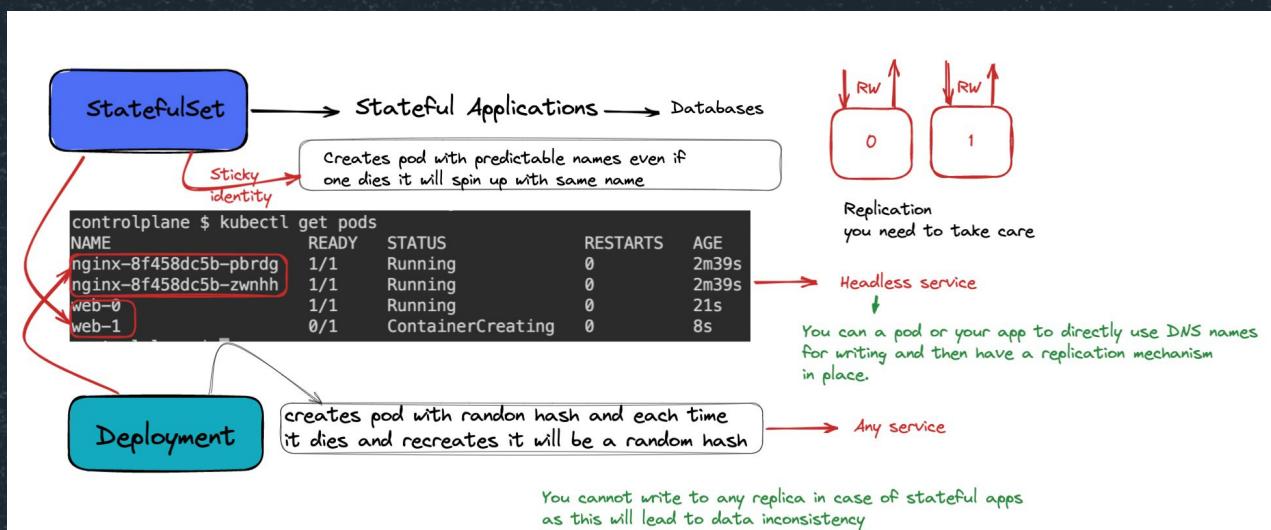
When you update the deploy image

- New pod gets created, old pod serves traffic until it finishes its terminationgraceperiod (30 by default)

Kubernetes



Kubernetes Series: #12 StatefulSet



Kubernetes



Kubernetes Series: #13 Networking

What happens when you run the pod?

```

controlplane $ kubectl get nodes
NAME      STATUS    ROLES     AGE   VERSION
controlplane   Ready    control-plane   68d   v1.24.0
node01      Ready    <none>    68d   v1.24.0
controlplane $ 

controlplane $ cat pod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: shared-namespace
spec:
  containers:
    - name: p1
      image: busybox
      command: ['bin/sh', '-c', 'sleep 10000']
    - name: p2
      image: nginx
controlplane $ kubectl apply -f pod.yaml
pod/shared-namespace created

```

After pod creation CNI attach Ip address and attach it to network

`ip netns list`
List of network namespaces

```

eth0
+---+
|   |
|   veth Root Ns
|   |
|   +---+
|   |   eth0<--> ip addr
|   |   192.168.1.4
|   |   busybox pause
|   |   nginx
|   |
|   Node

```

```

controlplane $ kubectl exec -it shared-namespace -- sh
Defaulted container "p1" out of: p1, p2
/ # ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 brd 127.0.0.1 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 brd :: scope host
        valid_lft forever preferred_lft forever
3: eth0@if9: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1460 qdisc noqueue
    link/ether be:26:a7:95:24:17 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.4/32 brd 192.168.1.4 scope global eth0
        valid_lft forever preferred_lft forever
        inet6 fe80::bc26:a7ff:fe95:2417/64 scope link
            valid_lft forever preferred_lft forever
/ # route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref  Use Iface
default         169.254.1.1   0.0.0.0       UG    0      0      0 eth0
169.254.1.1   *             255.255.255.255  UH    0      0      0 eth0

```

```

node01 $ ip netns list
cni-56b3761-826e-66eb-d587-e917ee245752 (id: 2)
cni-60337942-7f48-e61d-65f-984f76de5fe6 (id: 1)
cni-ae432c80-c285-9338-a047-d3dbf4a9fbe (id: 0)

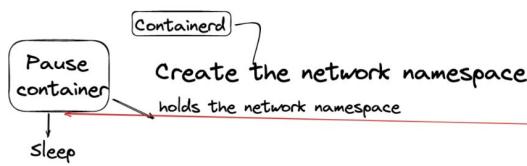
```

Kubernetes



Kubernetes Series: #14 Networking-2

There are veth pairs created or depending on CNI maybe different



To see that network namespace is held by pause container

```
node01 $ lsns | grep nginx
4026532587 mnt          2 32059 root      nginx: master process nginx -g daemon off;
4026532588 pid          2 32059 root      nginx: master process nginx -g daemon off;

node01 $ lsns -p 32059
          TYPE    NPROC   PID USER COMMAND
4026531835 cgroup   157    1 root /sbin/init
4026531837 cgroup   157    1 root /sbin/init
4026532518 net      3 31973 65535 /pause
4026532584 uts      3 31973 65535 /pause
4026532585 ipc      3 31973 65535 /pause
4026532587 mnt      2 32059 root nginx: master process nginx -g daemon off;
4026532588 pid      2 32059 root nginx: master process nginx -g daemon off;
```

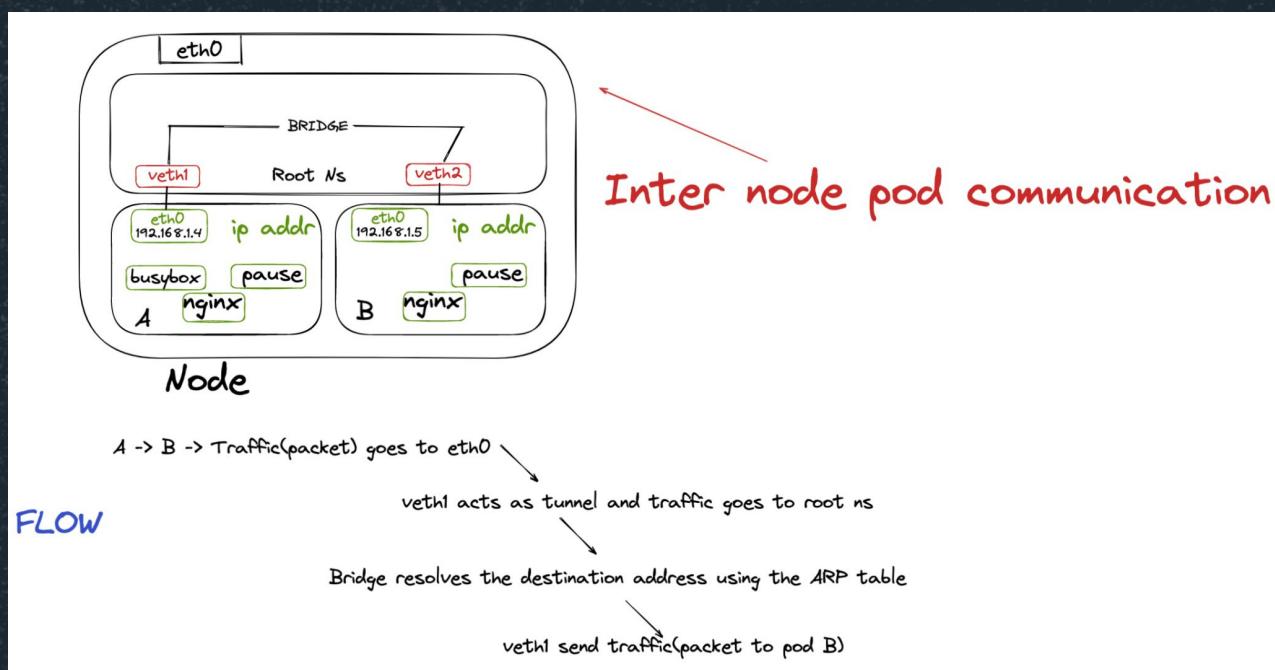
ls -lt /var/run/netns → List of namespaces created

```
node01 $ ls -lt /var/run/netns
total 0
-r--r--r-- 1 root root 0 Jul 16 14:25 cni-3cf655f5-9131-e23d-0756-b7ab6556ef8f
-r--r--r-- 1 root root 0 May  8 19:46 cni-06382f28-d5db-63f3-00ee-654e40ee904b
-r--r--r-- 1 root root 0 May  8 19:46 cni-592cc23b-1d2b-6054-bf32-437fd23a04df
node01 $ ip netns exec cni-3cf655f5-9131-e23d-0756-b7ab6556ef8f ip link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
3: eth0@if9: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1460 qdisc noqueue state UP mode DEFAULT group default
    link/ether 9e:bd:c7:d4:e5:99 brd ff:ff:ff:ff:ff:ff link-netnsid 0
node01 $ ip link | grep -A1 ^9
9: calic440f455693@if3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1460 qdisc noqueue state UP mode DEFAULT group default
    link/ether ee:ee:ee:ee:ee:ee brd ff:ff:ff:ff:ff:ff link-netns cni-3cf655f5-9131-e23d-0756-b7ab6556ef8f
node01 $
```

Kubernetes



Kubernetes Series: #15 Internode Pod Communication



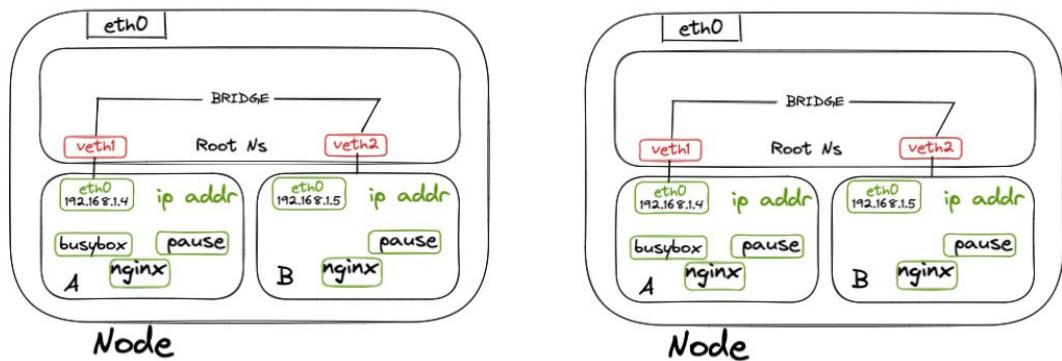
Kubernetes



Kubernetes Series: #16 Node-to-Node Communication

Node to node communication

Cannot use ARP as destination is not on same node



FLOW

To check if the destination is not on the same network bitwise or Adding operation is performed

Find the default gateway from ARP

Send to default gateway

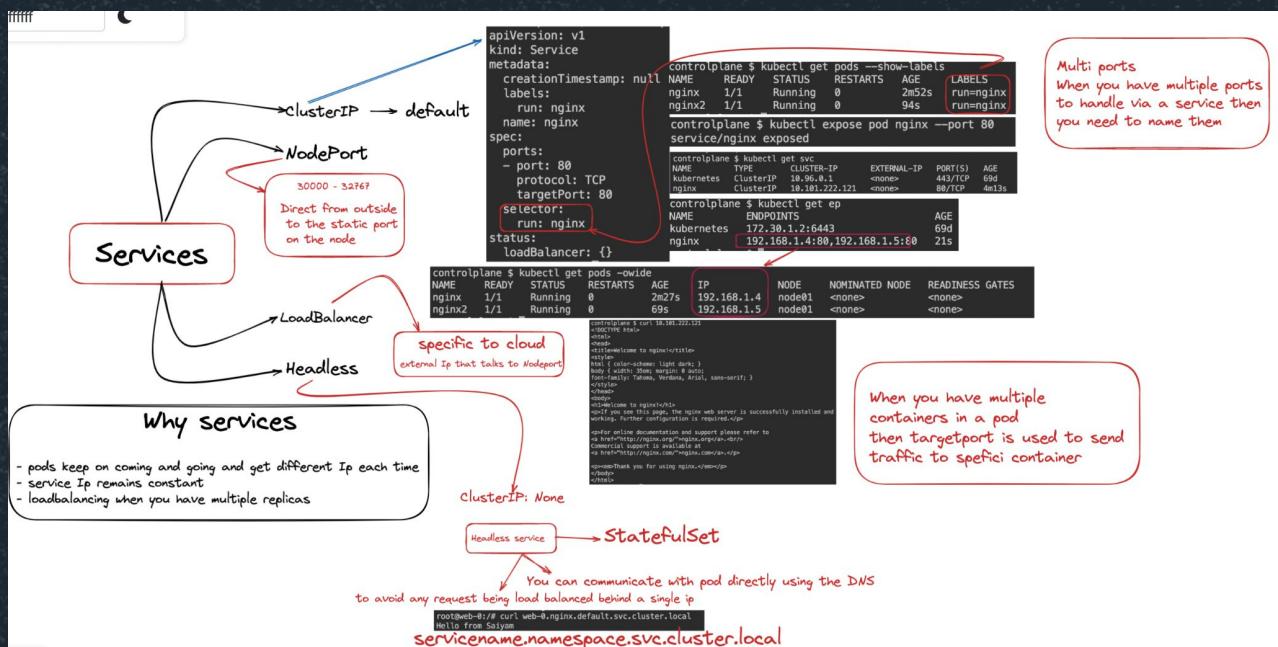
Default interface of other node

Services → iptables and netfilter

Kubernetes



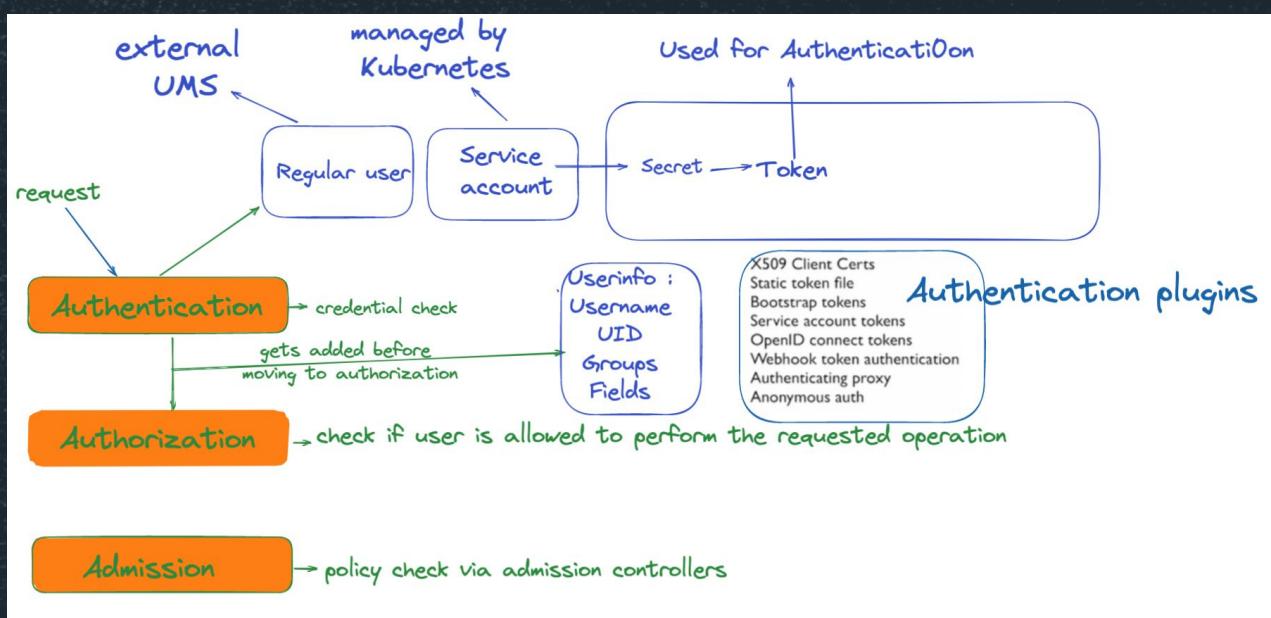
Kubernetes Series: #17 Services



Kubernetes



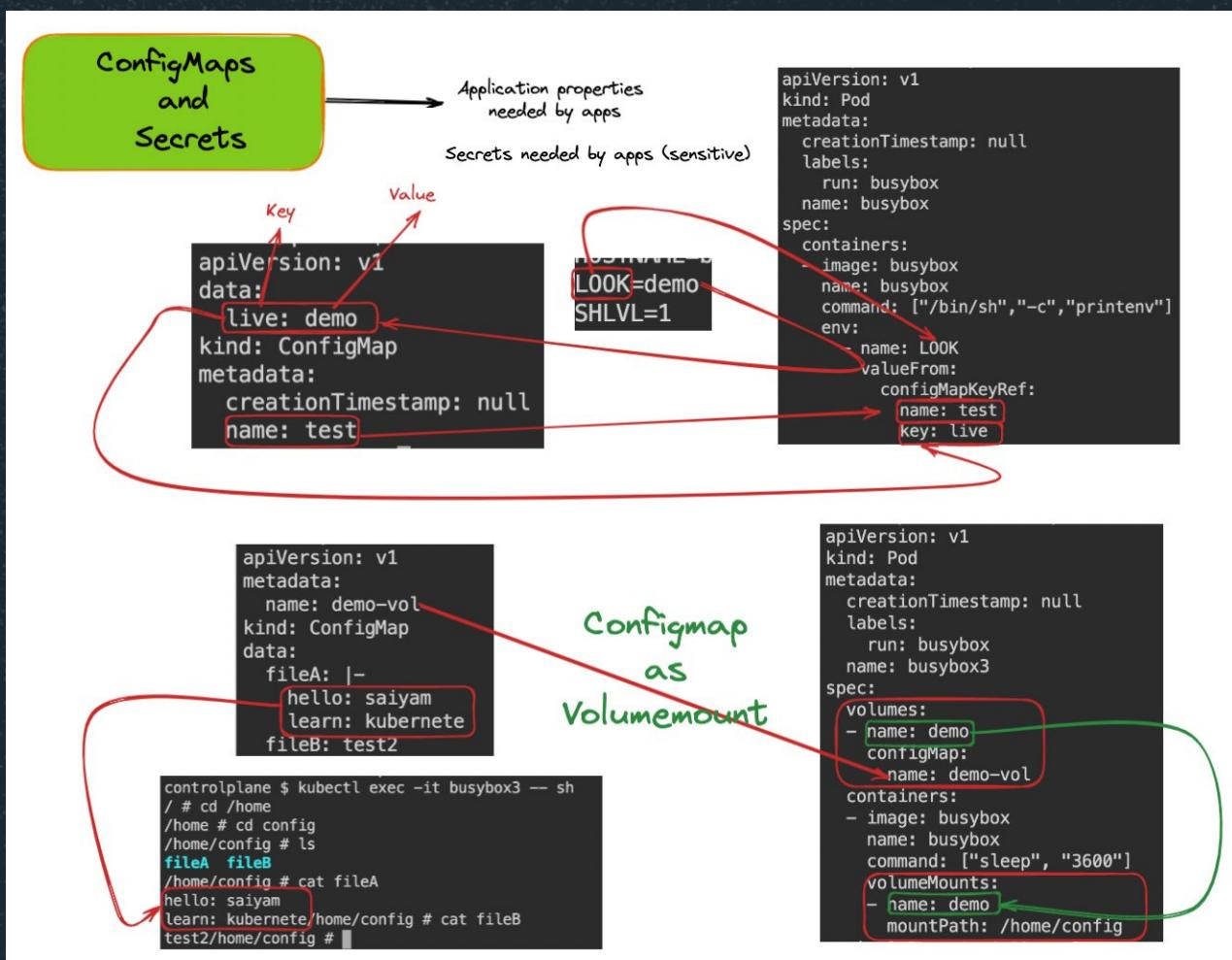
Kubernetes Series: #18 Authentication and Authorization



Kubernetes



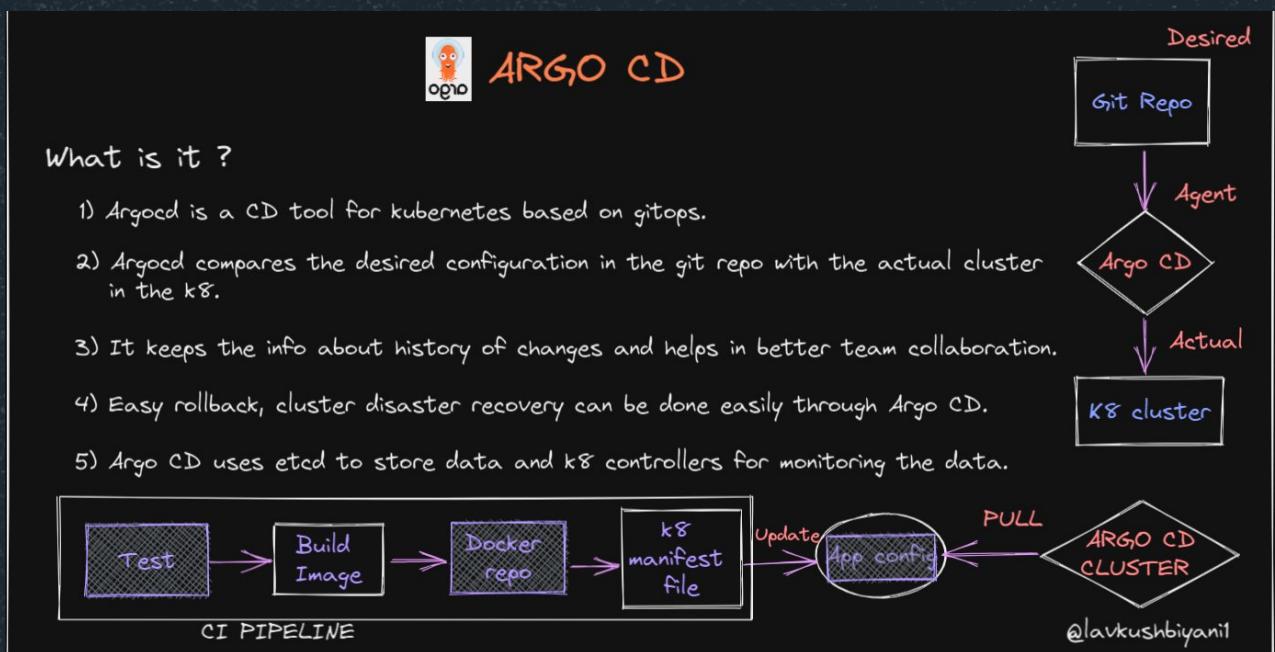
Kubernetes Series: #19 Configmaps & Secrets



Kubernetes



Argo CD



~Lavakush Biyani

Kubernetes



HELM



About

- Started as "Helm Classic" in 2015.
- CNCF Graduated and Open Source Software.
- Package manager for Kubernetes.
- Uses a collection of YAML files of an application called Helm Charts.
- Solves application distribution and management problems.

Features

- Install, upgrade and uninstall applications with a single command.
- All the Helm charts are in artifacthub.io to search and explore.
- Create new charts with Helm template.
- Manages application life cycle by itself.
- Installs software dependencies automatically.

@kubesimplify

~Pavan Gudiwada

Kubernetes



Kubecost



About

- Founded in 2019.
- CNCF member and OpenSource Software
- Provides real-time cost visibility and insights for teams.
- Improves upon standalone Grafana/Prometheus.
- Shows you where to cut down costs.

Features

- Better Cost Allocation
- Unified Cost Monitoring
- Alerts & Governance via email or slack
- Easy integration with cloud providers
- Export billing data for further analysis

@kubesimplify A small blue Twitter bird icon next to the handle.

~Pavan Gudiwada

| Page 28

Kubernetes



Kubernetes Image Aliases



Kubernetes

kubectl get pods	→	kubectl get po
kubectl get nodes	→	kubectl get no
kubectl get service	→	kubectl get svc
kubectl get deployments	→	kubectl get deploy
kubectl get namespaces	→	kubectl get ns
kubectl get pods --all-namespaces	→	kubectl get po -A
kubectl get pods -namespace name	→	kubectl get po -n name

@kubesimplify

~Pavan Gudiwada

Kubernetes



Terraform Images Commands



Terraform

Useful Commands

- * terraform init
- * terraform validate
- * terraform plan
- * terraform apply
- * terraform destroy
- * terraform fmt
- * terraform console
- * terraform state pull
- * terraform state push
- * terraform graph

@kubesimplify A small blue icon of a Twitter bird in flight.

~Pavan Gudiwada

Kubernetes



Name Labelling in Kubernetes

```
● ● ●

# Labelling in kubernetes <SOMELABEL=VALUE>

## Labelling a resource (e.g. Namespace)

$ kubectl label namespace default istio-injection=enabled

## Deleting a label

$ kubectl label namespace default istio-injection-

## Updating the value

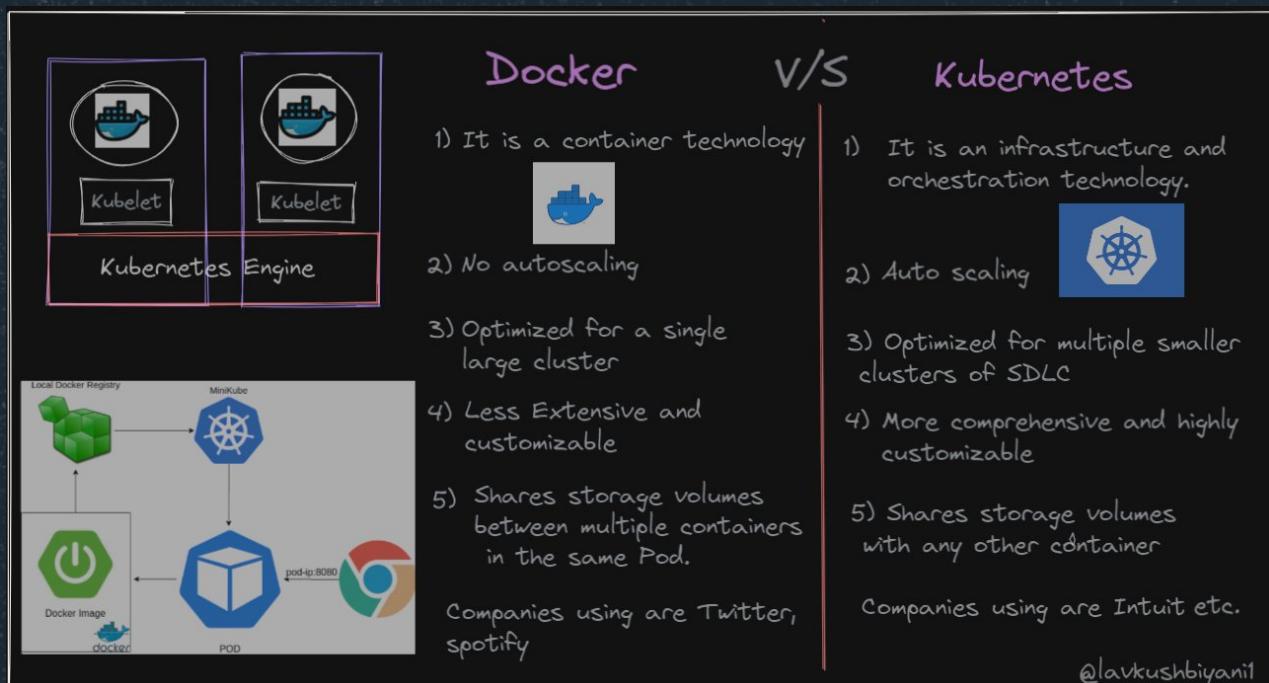
$ kubectl label namespace default istio-injection=enabled --overwrite
```

~Anurag Kumar

Kubernetes



Docker & Kubernetes

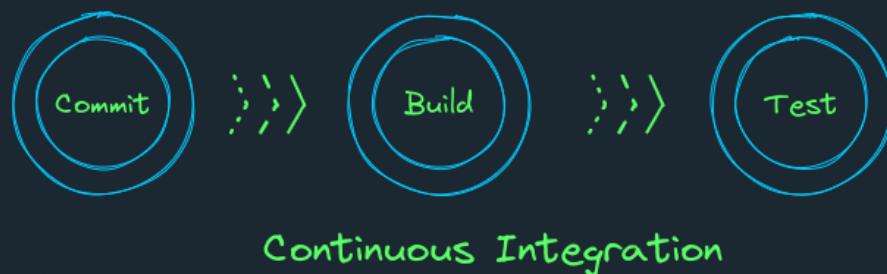


~Lavakush Biyani

CI/CD



Continuous Integration



@vaidansh23

~Vaidansh Bhardwaj

CI/CD



Continuous Delivery



@vaidansh23

~Vaidansh Bhardwaj

CI/CD



Continuous Deployment



Continuous Delivery



Continuous Deployment

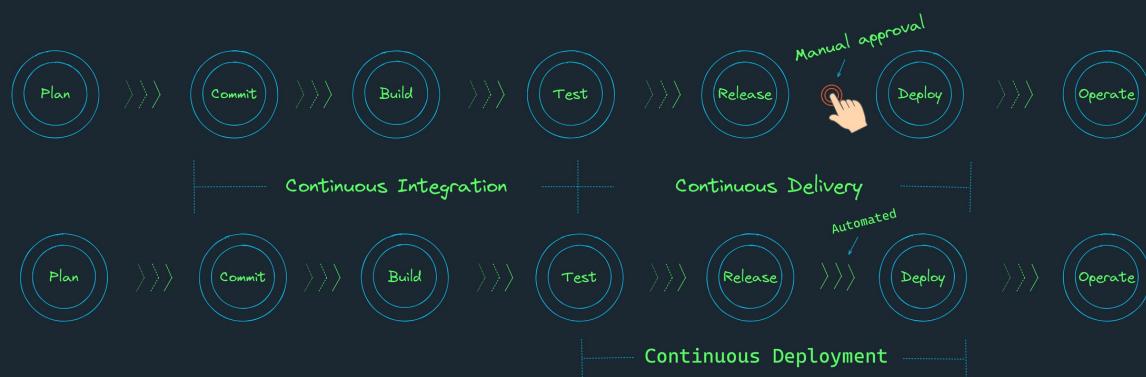
@vaidansh23

~Vaidansh Bhardwaj

CI/CD



CI/CD Pipeline

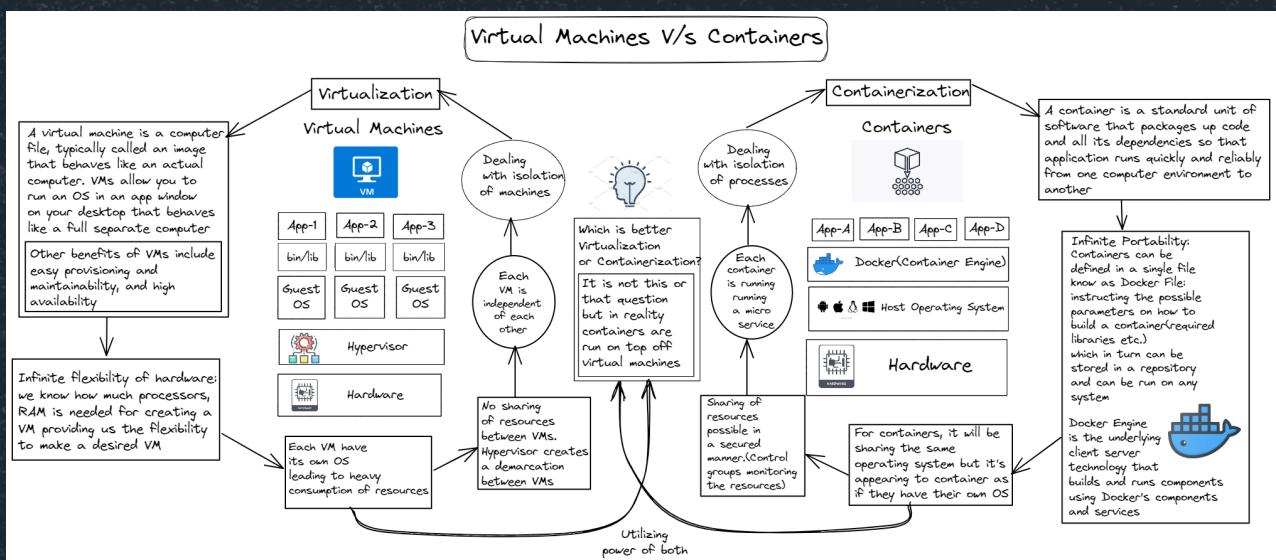


~Vaidansh Bhardwaj

Containerization & VMs



Simplifying Containers & VMs

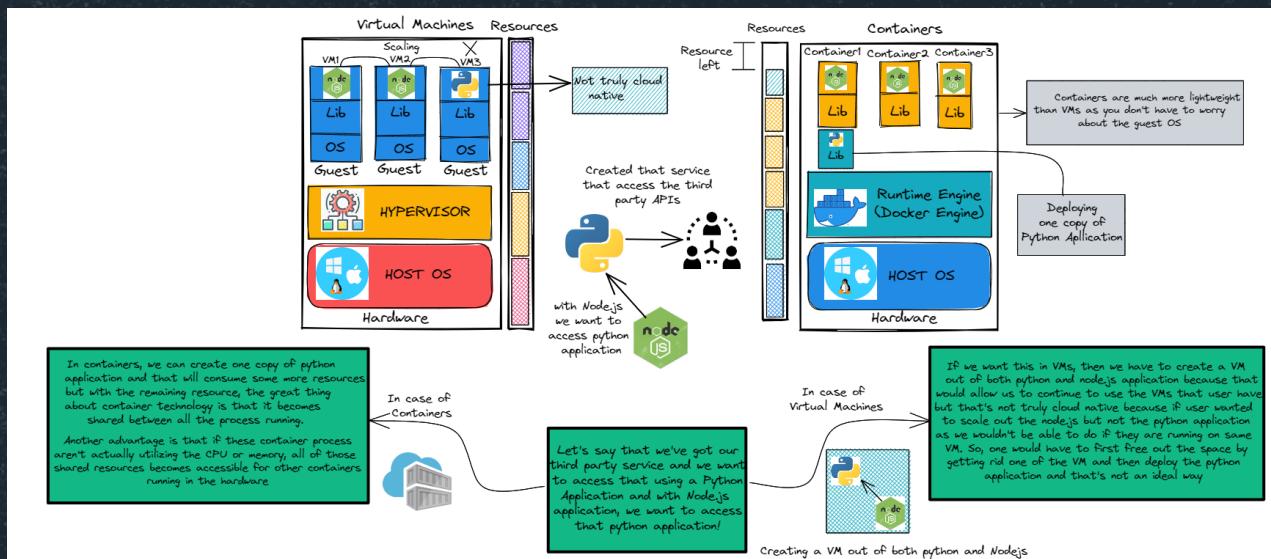


~Aviral Singh

Containerization & VMs



Resource Containers & VMs

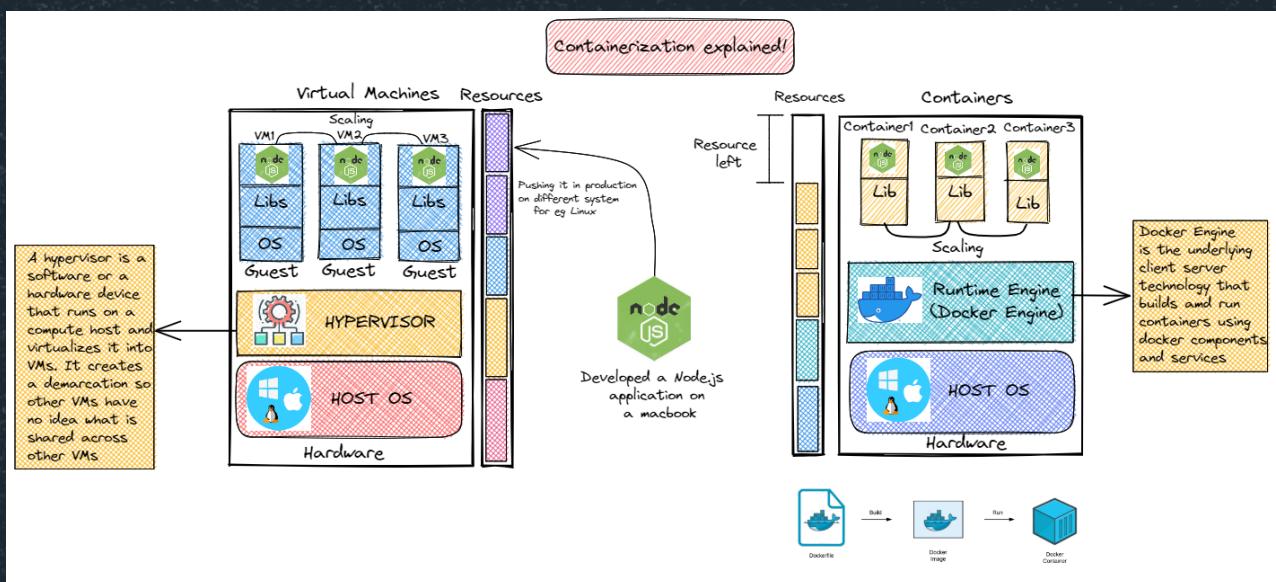


~Aviral Singh

Containerization & VMs



Resource Management

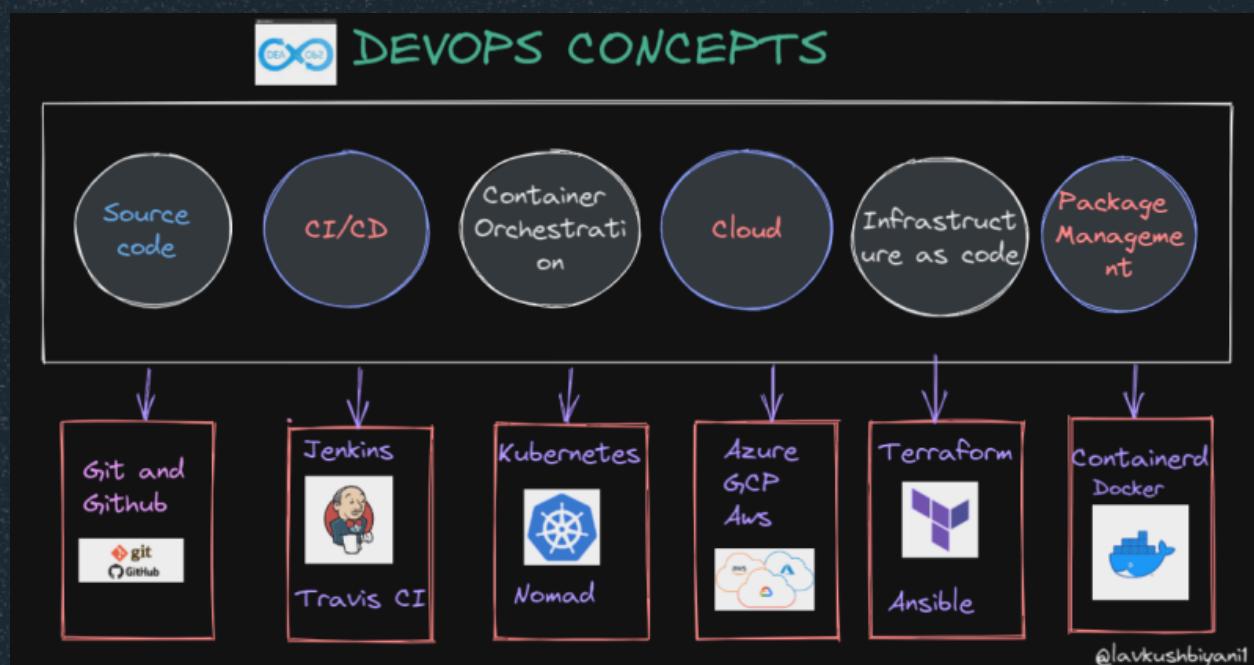


~Aviral Singh

DevOps



~An overview

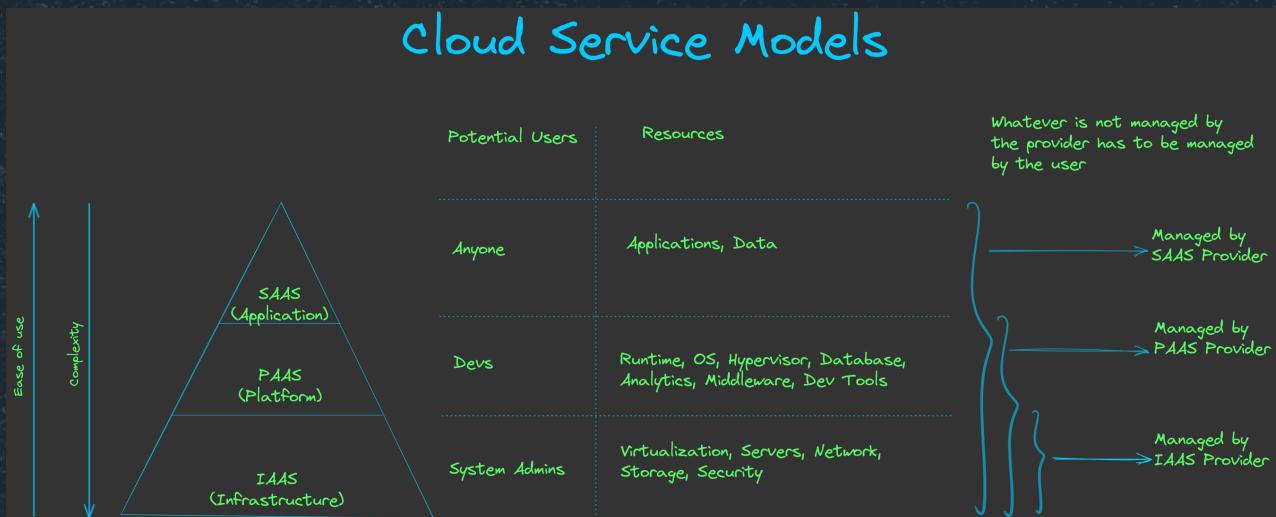


~Lavakush Biyani

Cloud Service Models



Cloud Service Models

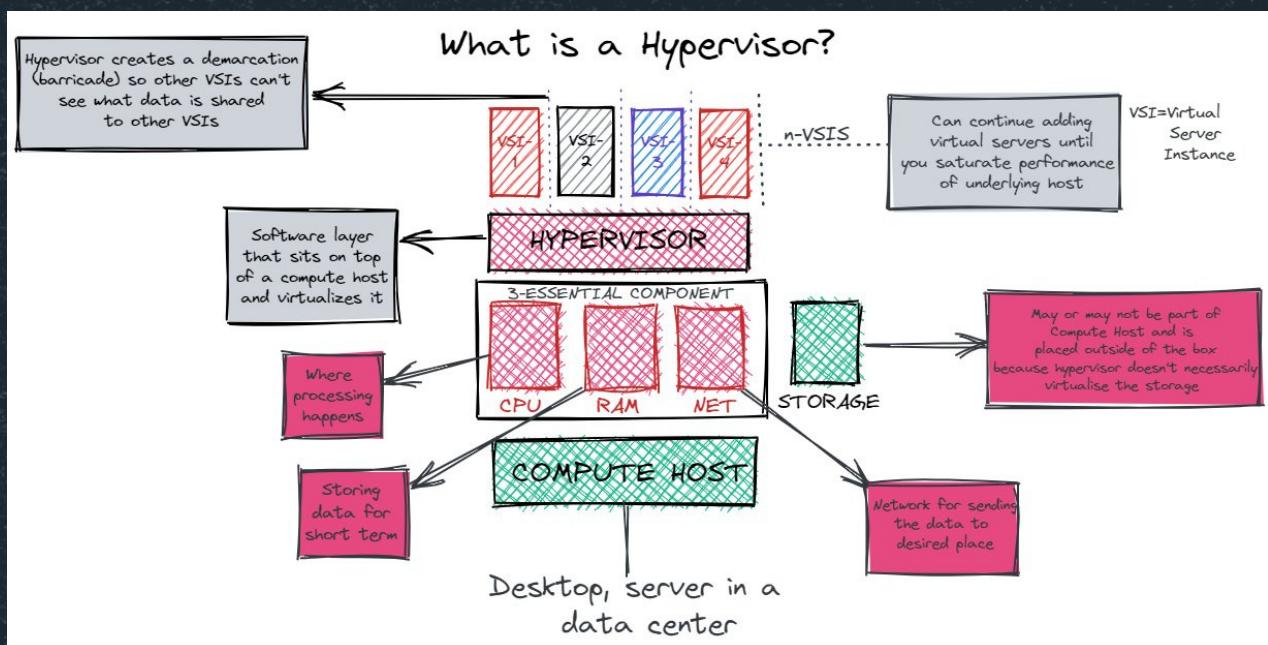


~Arnav Barman

Hypervisor



What is a Hypervisor

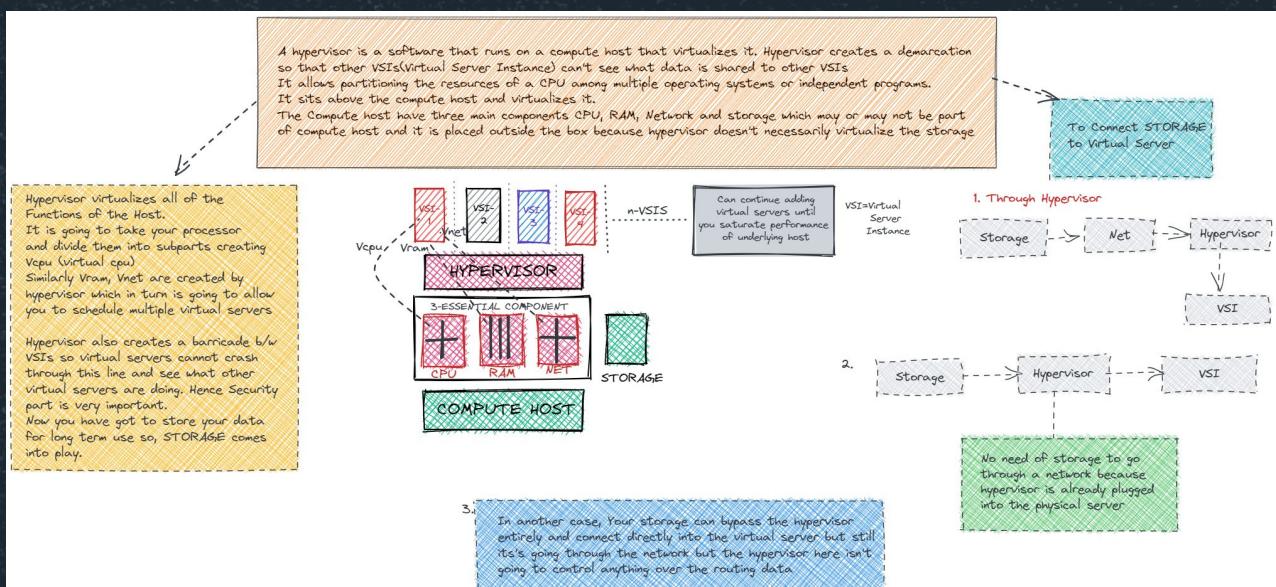


~Aviral Singh

Hypervisor



Hypervisor – VMs

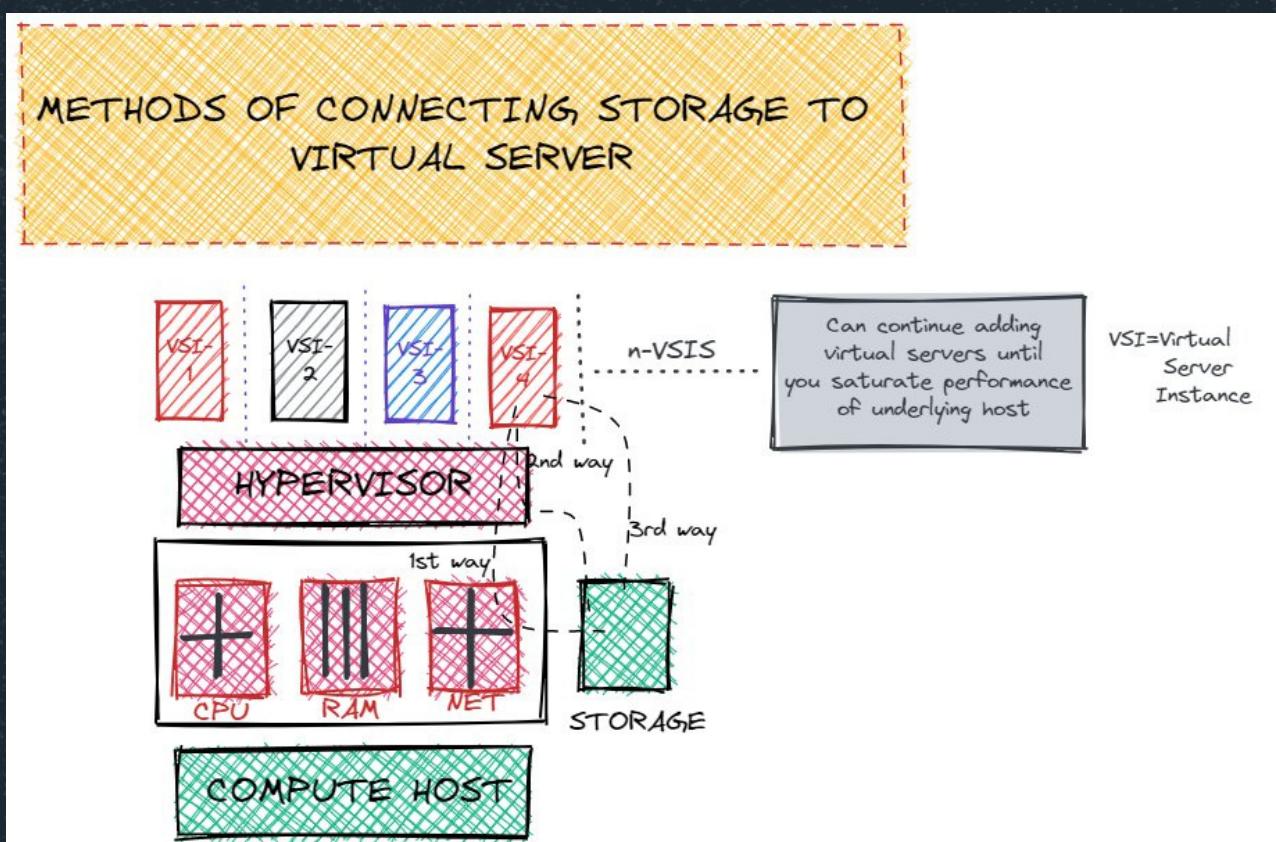


~Aviral Singh

Hypervisor



Connecting Storage to Virtual Server

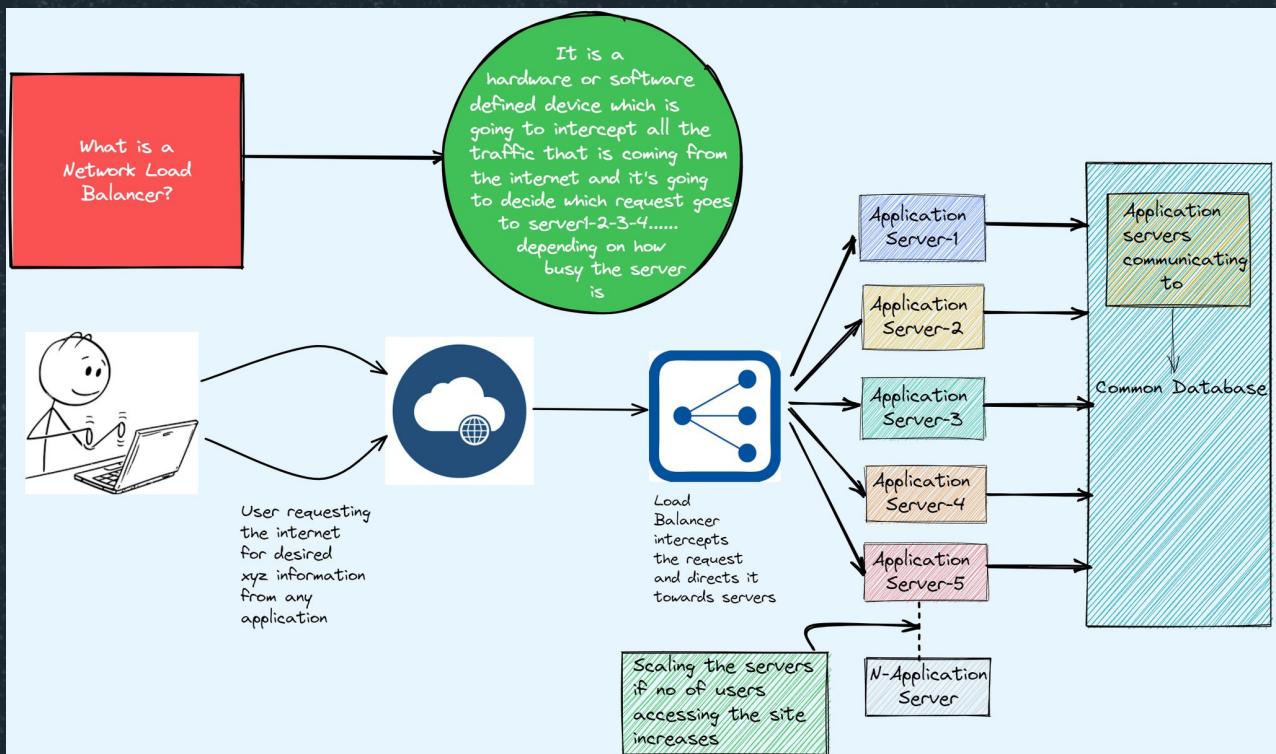


~Aviral Singh

Networking



Network Load Balancer #1

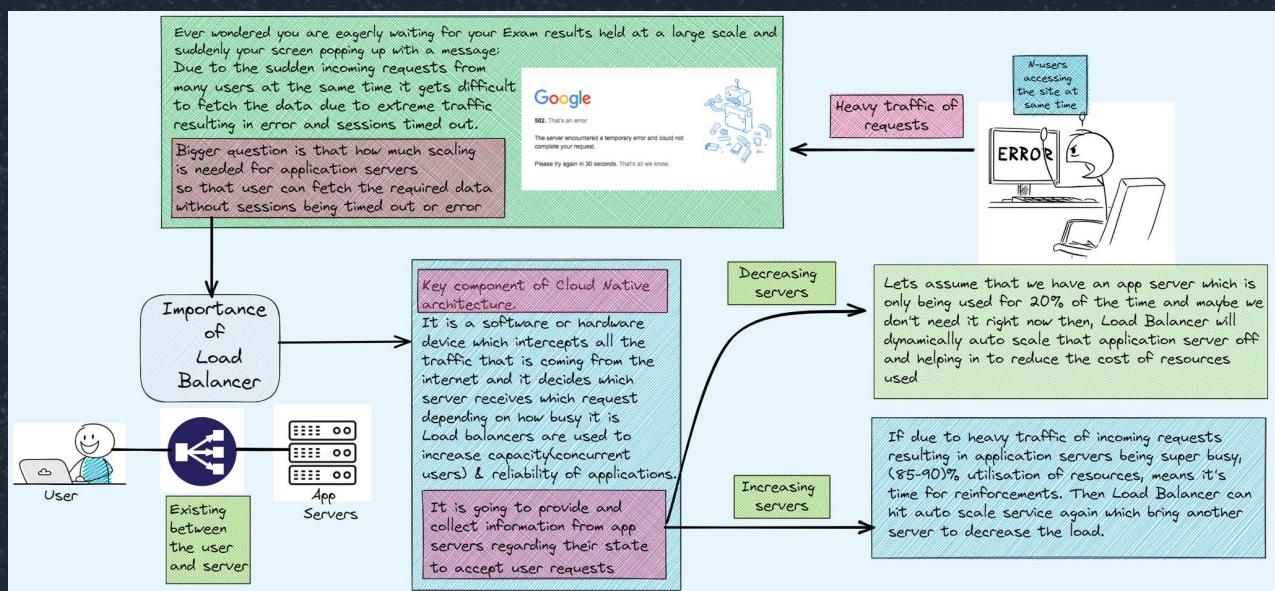


~Aviral Singh

Networking



Network Load Balancer #2

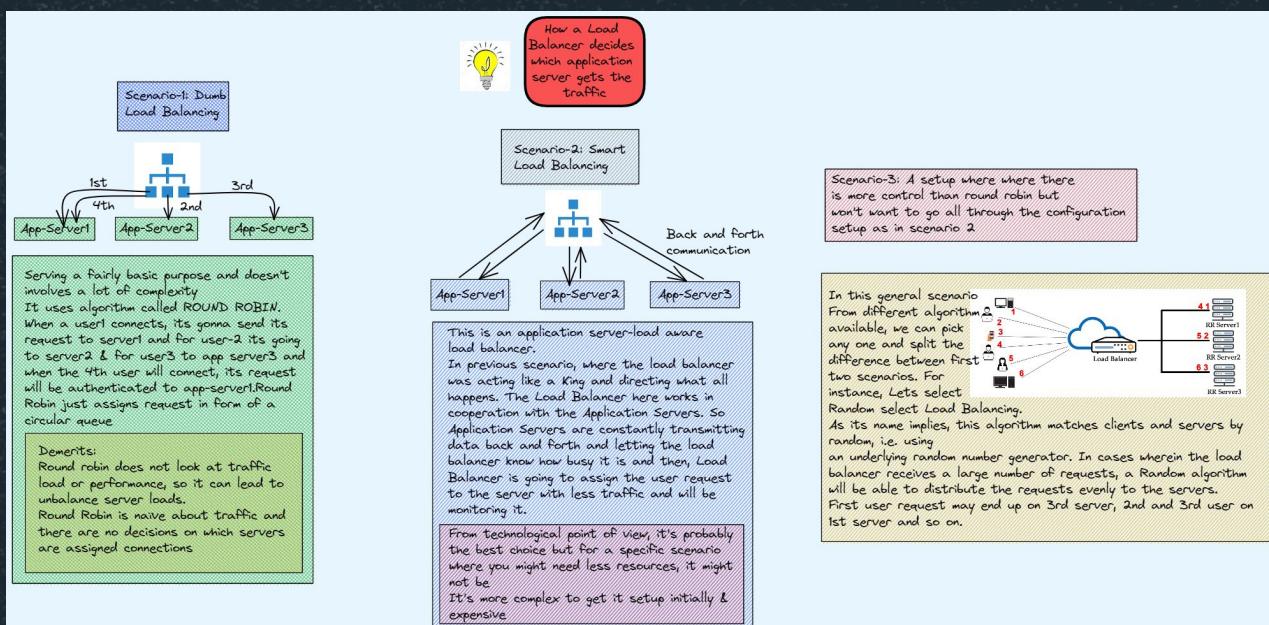


~Aviral Singh

Networking



Network Load Balancer #3

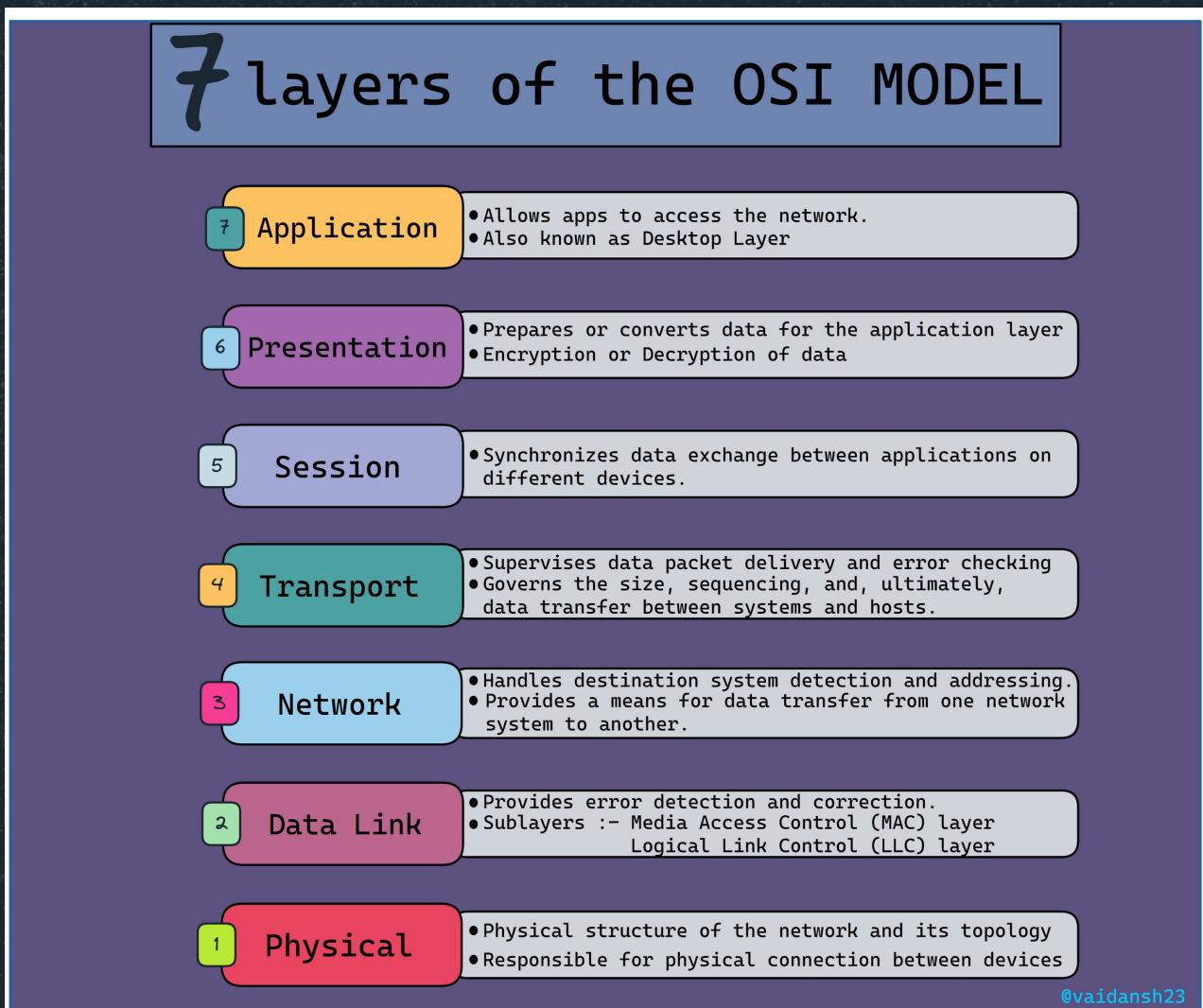


~Aviral Singh

Networking



OSI Model



@vaidansh23

~Vaidansh Bhardwaj

Prometheus



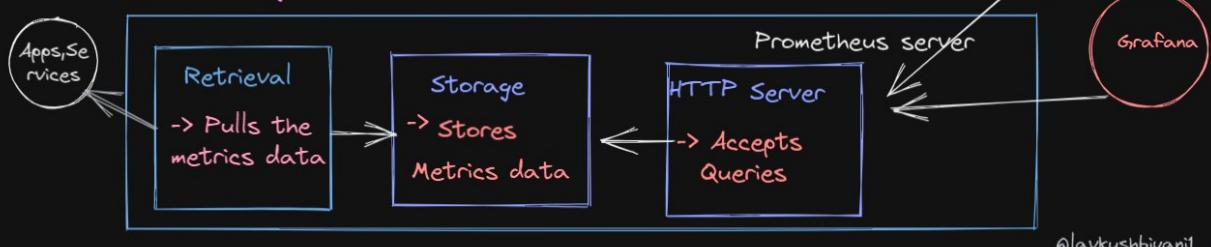
PROMETHEUS

-> It is a mainstream monitoring tool of choice in container and microservices.

USE CASES :

- 1) To constantly monitor all the microservices.
- 2) To get alert when something crashes in the server.
- 3) To check the memory usage and notify the administrator before hand.

How does Monitoring works :



@lavkushbiyani1

~Lavakush Biyani

Service Mesh

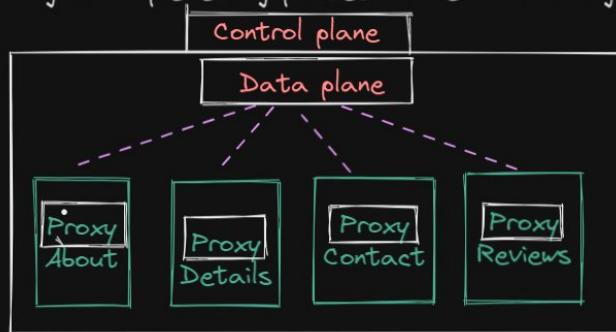


SERVICE MESHES

-> The main use of service meshes is to solve the challenges of microservices.

Definition:

- > It is a dedicated and configurable infrastructure layer that handles the communication between the services without having to change the code in a microservice architecture.
- > Service instances, sidecars and their interactions called the data plane in a service mesh. A different layer called the control plane manages tasks such as creating instances, monitoring and implementing policies for network management and security.



Responsibilities :

- 1) Traffic Management
- 2) Security
- 3) Observability
- 4) Service Discovery
 - > Health Checks
 - > Load balancing

@lavkushbiyani1

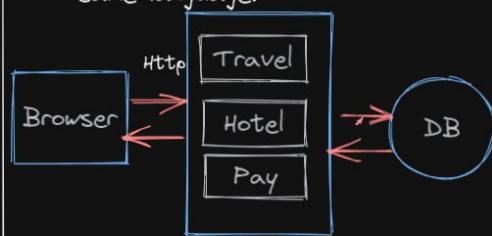
~Lavakush Biyani

Monolithic v/s Microservices

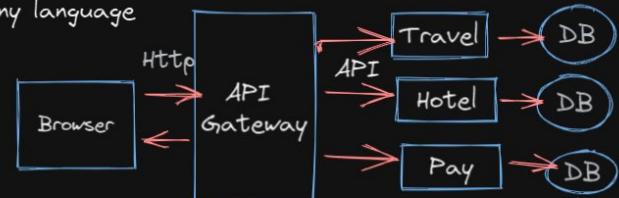


Monolithic V/s Microservices

- 1) All components are tightly packed
- 2) Minimal latency
- 3) Difficult in scaling
- 4) More time to deploy
- 5) All the code pieces are written in same language.



- 1) All components are separated.
- 2) Latency is more when compared the Monolithic.
- 3) Easy in scaling.
- 4) Can deploy individual components even on daily basis.
- 5) Flexibility to code each component in any language



@lavakushbiyani1

~Lavakush Biyani

Thanks for the read!

Hope you learned something new and valuable. Follow us for more such content!



<https://twitter.com/kubesimplify>



<https://kubesimplify.com>



<https://www.youtube.com/@kubesimplify>



<https://blog.kubesimplify.com>



<https://linkedin.com/company/kubesimplify>



<https://www.github.com/kubesimplify>



<https://discord.gg/26Z384WSPB>

Compiled and designed by:

Arnav Barman

Vaidansh Bhardwaj

