



KUBESIMPLIFY
CLOUDNATIVE FOR EVERYONE

VISUAL GUIDES COLLECTION



@kubesimplify



kubesimplify.com



@kubesimplify

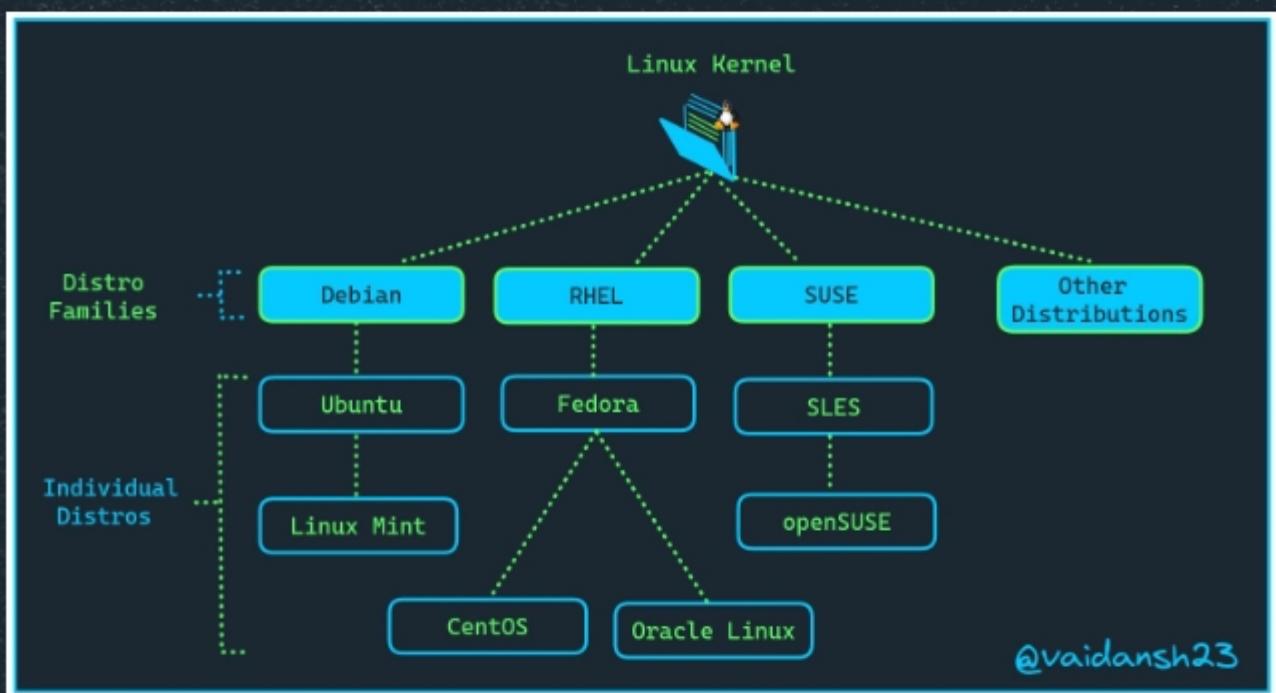
Table Of Contents

- 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

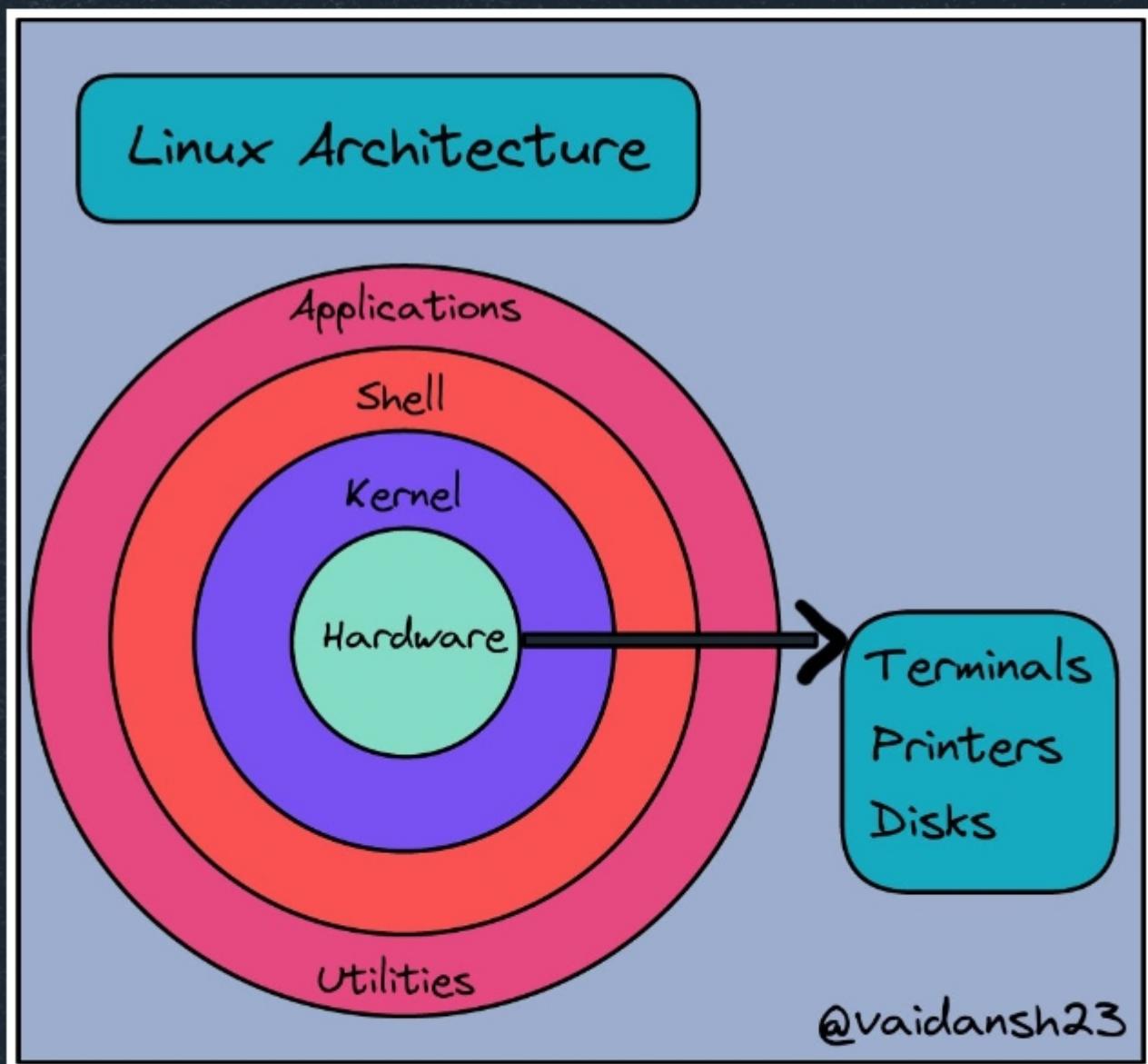


~Vaidansh Bhardwaj

Linux



Linux Architecture



~Vaidansh Bhardwaj

Linux



Linux Commands

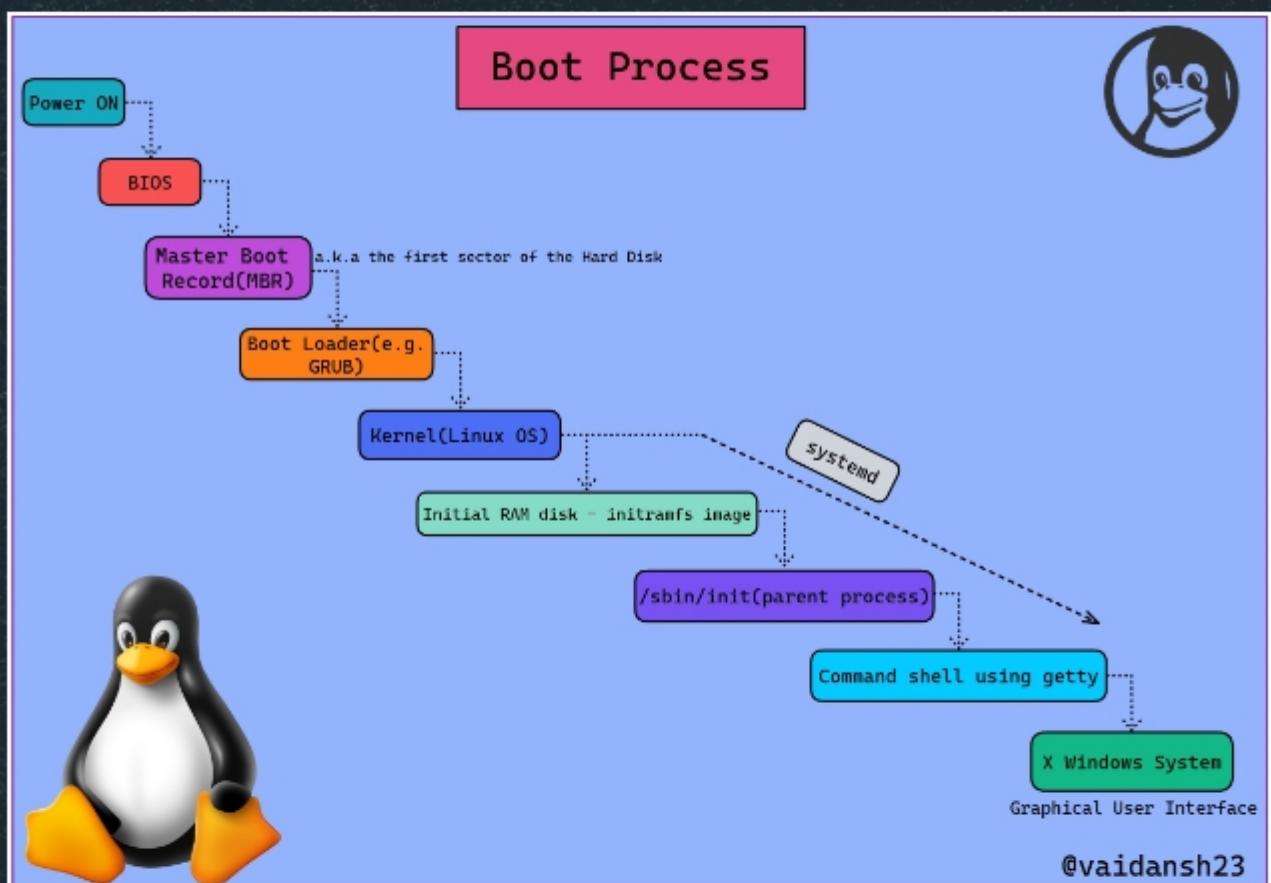
Linux Commands									
<ul style="list-style-type: none">● Basic Linux Commands<ul style="list-style-type: none">→ sudo - superuser do, used for running as root.→ mkdir - used to create a folder in our system.→ cd - allows us to move into the directory.→ rmdir - used to remove the directory.→ pwd - 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.<td><ul style="list-style-type: none">→ 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.<ul style="list-style-type: none">● File Permission Commands<table border="1"><tr><td>0 = None ---</td></tr><tr><td>1 = Execute only --x</td></tr><tr><td>2 = Write only ~w</td></tr><tr><td>3 = Write & Execute -wx</td></tr><tr><td>4 = Read Only r--</td></tr><tr><td>5 = Read & Execute r=x</td></tr><tr><td>6 = Read & Write rw-</td></tr><tr><td>7 = Read, Write & Execute rwx</td></tr></table>→ ls -l - shows file type and access permission.→ chmod u=rwx,g=rx,o=r - setting explicit permissions for different groups.→ chown user - changing the ownership of a file/directory.<ul style="list-style-type: none">● 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.</td>	<ul style="list-style-type: none">→ 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. <ul style="list-style-type: none">● File Permission Commands<table border="1"><tr><td>0 = None ---</td></tr><tr><td>1 = Execute only --x</td></tr><tr><td>2 = Write only ~w</td></tr><tr><td>3 = Write & Execute -wx</td></tr><tr><td>4 = Read Only r--</td></tr><tr><td>5 = Read & Execute r=x</td></tr><tr><td>6 = Read & Write rw-</td></tr><tr><td>7 = Read, Write & Execute rwx</td></tr></table>→ ls -l - shows file type and access permission.→ chmod u=rwx,g=rx,o=r - setting explicit permissions for different groups.→ chown user - changing the ownership of a file/directory. <ul style="list-style-type: none">● 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.	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
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									

~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

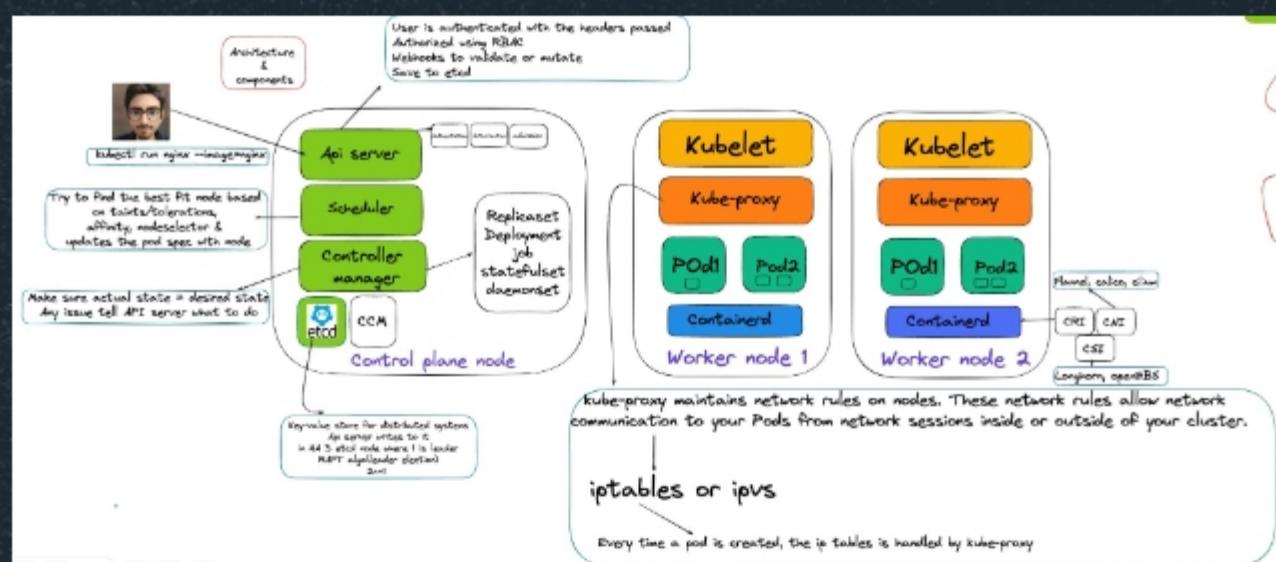
Why Kubernetes ?

- 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 Kubernetes YAML file with various annotations:

- YAML**: A box labeled "Pod example" with a red arrow pointing to the file content.
- Annotations**: A callout box with the following points:
 - Widely used in the cloud native space
 - human-readable data-serialization language
 - YAML ain't markup language
 - Very easy; indentation matters a lot
- Key: value**: Points to the `image: nginx` entry in the `spec.containers` section.
- Object**: Points to the `containers` section itself.
- attributes of the objects**: Points to the `image`, `name`, and `resources` fields in the `containers` section.
- List**: Points to the `dnsPolicy` and `restartPolicy` fields.
- List items**: Points to the `status` field.
- Placeholder**: Points to the `image: ${image_deploy_tag}` placeholder in the `image` field.
- \$ for environment variables**: Points to the `value: "${PORT}"` placeholder in the `ports` section.
- pipe symbol for multi-line string**: Points to the multi-line string starting with `apiVersion: v1`.
- Another example can be in a pod - shell script**: Points to the multi-line string in the `stdin: /bin/sh -c curl -sSf https://kubernetes.io | grep > /etc/kubernetes/podinfo` command.
- Separate different objects using this**: Points to the two separate code snippets at the bottom.

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

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

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

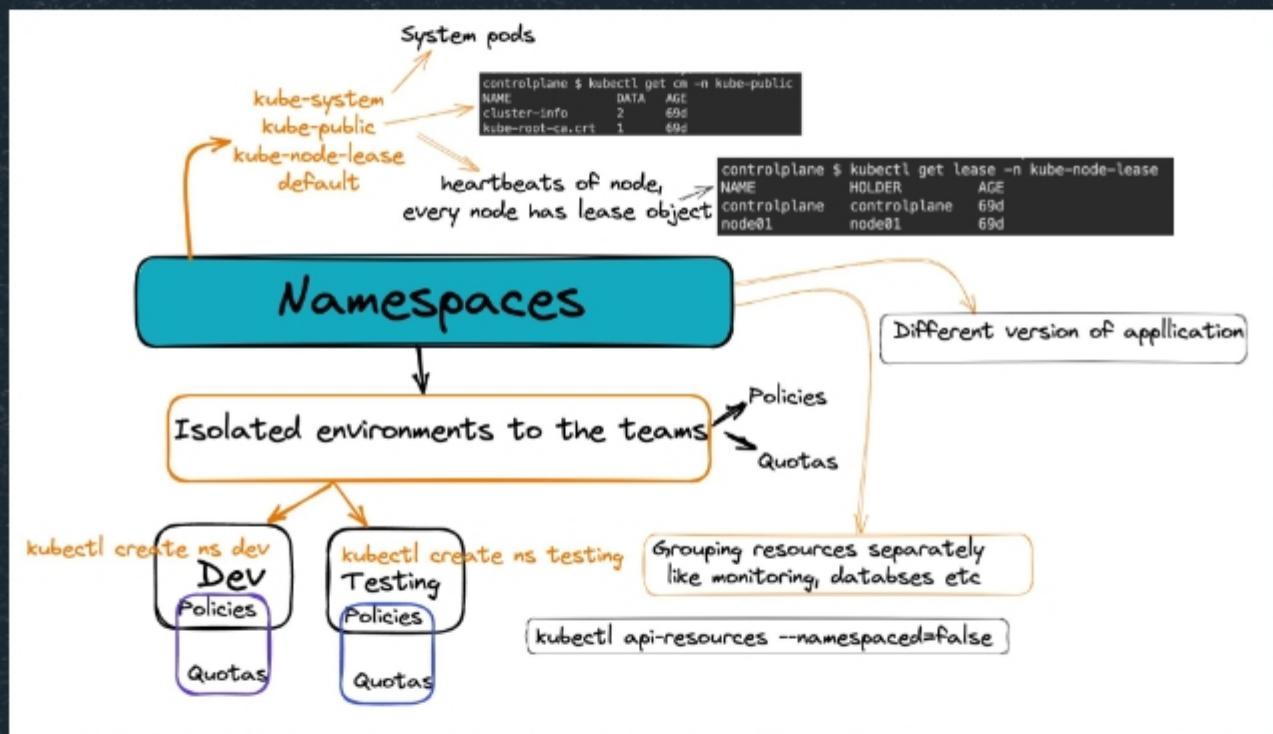
```
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    run: nginx
    name: nginx
  spec:
    containers:
      - image: nginx
        name: nginx
        resources: {}
    dnsPolicy: ClusterFirst
    restartPolicy: Always
  status: {}
```

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

Kubernetes



Kubernetes Series: #4 Namespaces



Kubernetes



Kubernetes Series: #5 Labels & Selectors

The diagram illustrates the relationship between labels and selectors in Kubernetes. It features a central box labeled "Labels and selectors" with arrows pointing to various components:

- An arrow from the top left points to a red callout: "label selectors can be".
- An arrow from the top right points to a red callout: "key-value pairs defined in metadata section".
- An arrow from the bottom left points to a red callout: "eg nodeSelector". Below it, two options are shown: "equity based label" and "set based labels".
- An arrow from the bottom right points to a red callout: "add meaning to the kubernetes object".
- A large red curved arrow on the right side points from the "key-value pairs" callout to a "Deployment" configuration file on the right.
- A red callout at the bottom center points to a terminal window showing pod selection commands.

Labels and selectors

label selectors can be
key-value pairs defined in metadata section
add meaning to the kubernetes object

eg nodeSelector
equity based label
set based labels in notin exists

Deployment configuration:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

Terminal output:

```
controlplane $ kubectl get pods --show-labels
No resources found in default namespace.
controlplane $ kubectl run nginx --image=nginx
pod/nginx created
controlplane $ kubectl get pods --show-labels
NAME     READY   STATUS    RESTARTS   AGE   LABELS
nginx   0/1    ContainerCreating   0      3s   running,nginx
controlplane $ kubectl label pod nginx live=demo
pod/nginx labeled
controlplane $ kubectl get pods --show-labels
NAME     READY   STATUS    RESTARTS   AGE   LABELS
nginx   1/1    Running   0       22s   live=demo,run=nginx
controlplane $
```

selector:
matchLabels:
 component: redis
matchExpressions:
 - {key: tier, operator: In, values: [cache]}\n - {key: environment, operator: NotIn, values: [dev]}

Kubernetes



Kubernetes Series: #6 PODS

The diagram illustrates the structure of a Kubernetes Pod. On the left, a green-bordered box represents a Node. Inside the Node is a smaller green-bordered box labeled "Pod ip". Within the "Pod ip" box, there is a container labeled "c1" which contains an "nginx" component. Below this is the text "Pod nginx". On the right, a code block shows the corresponding YAML configuration for this Pod:

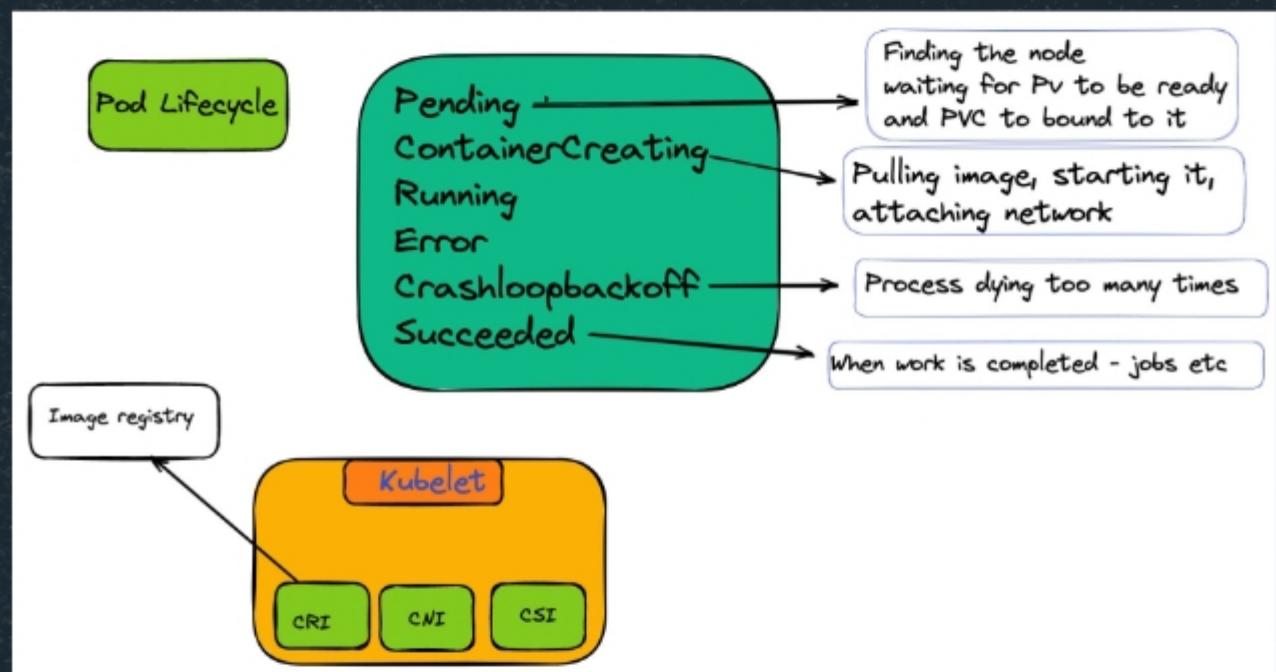
```
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    run: nginx
  name: nginx
spec:
  containers:
  - image: nginx
    name: nginx
  ports:
  - containerPort: 80
  resources: {}
dnsPolicy: ClusterFirst
restartPolicy: Always
status: {}
```

Two red arrows point from the "name: nginx" field in the YAML to the "nginx" component in the container and the "name: nginx" label in the "Pod ip" box. At the bottom left, a red-outlined box contains the command: "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

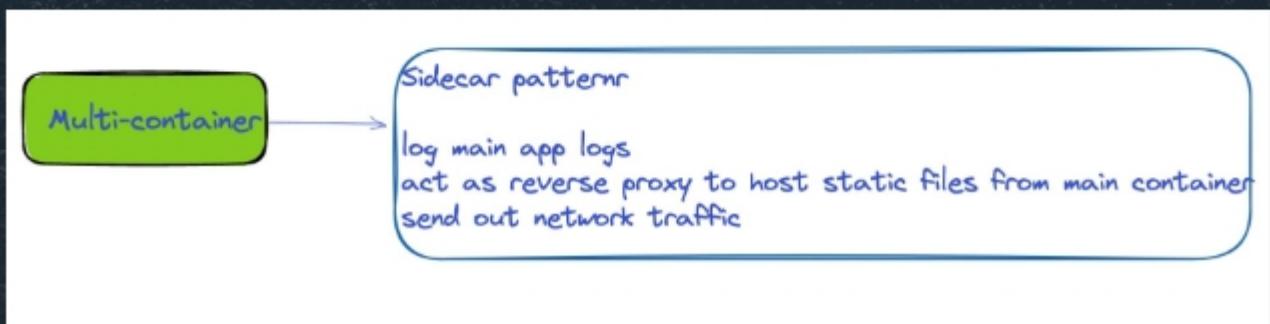
- can contain the custom code that is not present in the app
- change filesystem based on certain logic before the main container starts
- pre condition checks
- run in sequential order

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	85s	kubelet	Successfully pulled image "nginx" in 4.683285911s
Normal	Created	85s	kubelet	Created container nginx
Normal	Started	85s	kubelet	Started container nginx

Kubernetes



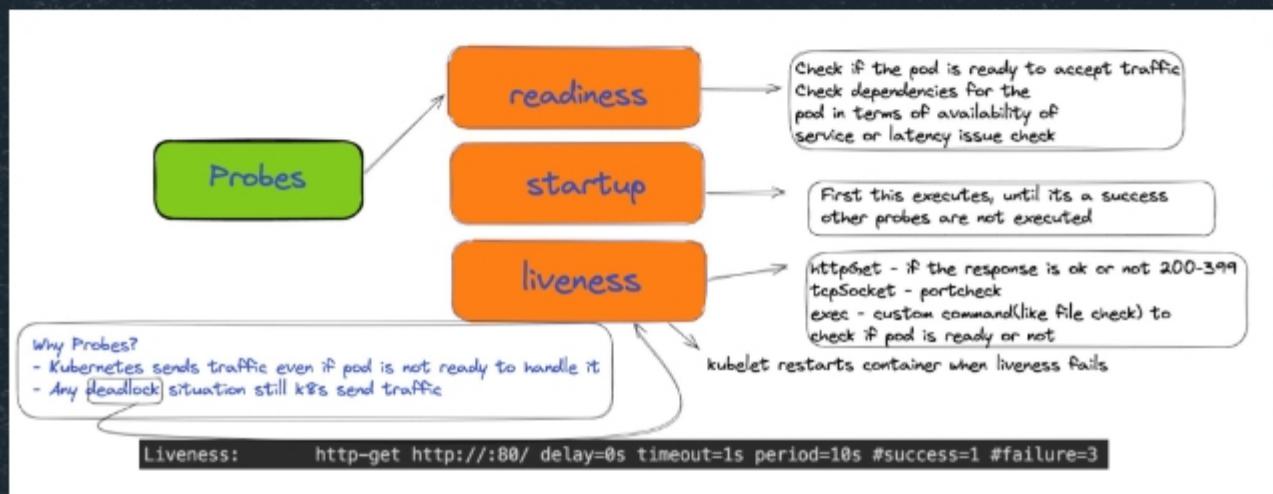
Kubernetes Series: #9 Multi-Container



Kubernetes



Kubernetes Series: #10 Probes



Kubernetes



Kubernetes Series: #11 Deployment

The diagram illustrates a **Deployment** box at the top, with two smaller boxes labeled **1** and **2** below it. An arrow points from the Deployment box to the number 1, with the text "Also create RS" above it. Another arrow points from the Deployment box to the number 2, with the text "Create 2 pods" above it. To the right of the diagram is a terminal window showing deployment configuration and a command output.

```
kubectl rollout status deployment name
```

```
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
```

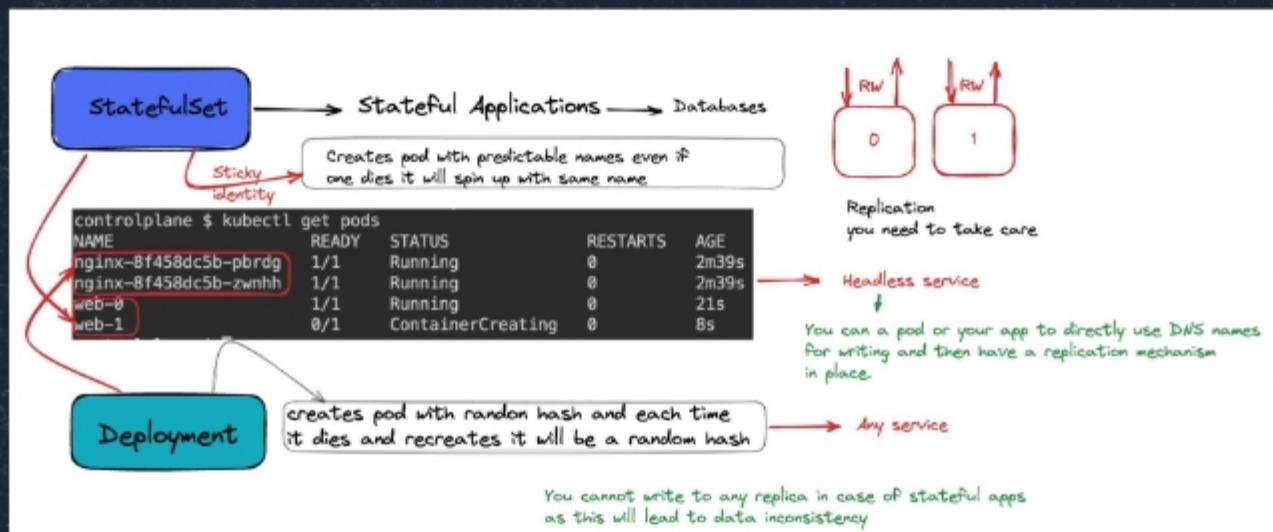
When you update the deploy image

- 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

The diagram illustrates a node's networking architecture. At the top, an interface labeled 'eth0' is shown. Below it, a 'Root Ns' contains a 'veth' pair. A 'Container' section contains two containers: 'busybox' (IP: 192.168.1.4) and 'nginx'. The 'busybox' container has an 'ip addr' configuration. The 'nginx' container is shown with a 'pause' status. Arrows indicate the connection from the 'eth0' interface to the 'Root Ns', and from the 'Root Ns' to the 'Container' section.

```
controlplane $ kubectl exec -it shared-namespace -- sh
/ # 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 127.255.255.255 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 brd :: scope host
        valid_lft forever preferred_lft forever
3: eth0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1460 qdisc noqueue
    link/ether 76:61:91:24:77:00 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.4/24 brd 192.168.1.255 scope global eth0
        valid_lft forever preferred_lft forever
        inet6 fe80::7661:91ff:fe24:77ff/64 scope link
            valid_lft forever preferred_lft forever
```

```
/ # route
Kernel IP routing table
destination     gateway         netmask        flags  metric  ref  use  iface
default         *               0.0.0.0        UG    0      0      0  eth0
199.254.1.1    *               255.255.255.0  UG    0      0      0  eth0
```

```
node01 $ ip netns list
cni-45523161-824e-46b5-8587-e917ee245752 (id: 2)
cni-48373942-7145-4615-8514-881f76d55a6 (id: 1)
cni-ae432c80-c285-9330-9b47-4335074a8fb6 (id: 0)
```

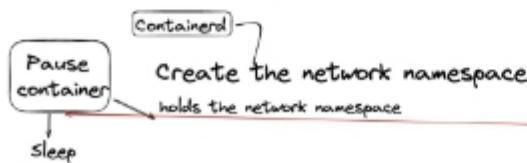
ip netns list
List of network namespaces

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 $ lsof -p 32859
      PID  COMMAND
4026532587 master process nginx -g daemon off;
4026532588 master process nginx -g daemon off;
node01 $ lsof -p 32859
      PID  COMMAND
4026531835 cgroup 1 root /sbin/init
4026531837 user    157 1 root /sbin/init
4026531838 net    3 31073:45536 /pause
4026532584 user    3 31073:45536 /pause
4026532585 ipc    3 31073:45536 /pause
4026532587 master process nginx -g daemon off;
4026532588 pid    2 32859 root nginx master process nginx -g daemon off;
```

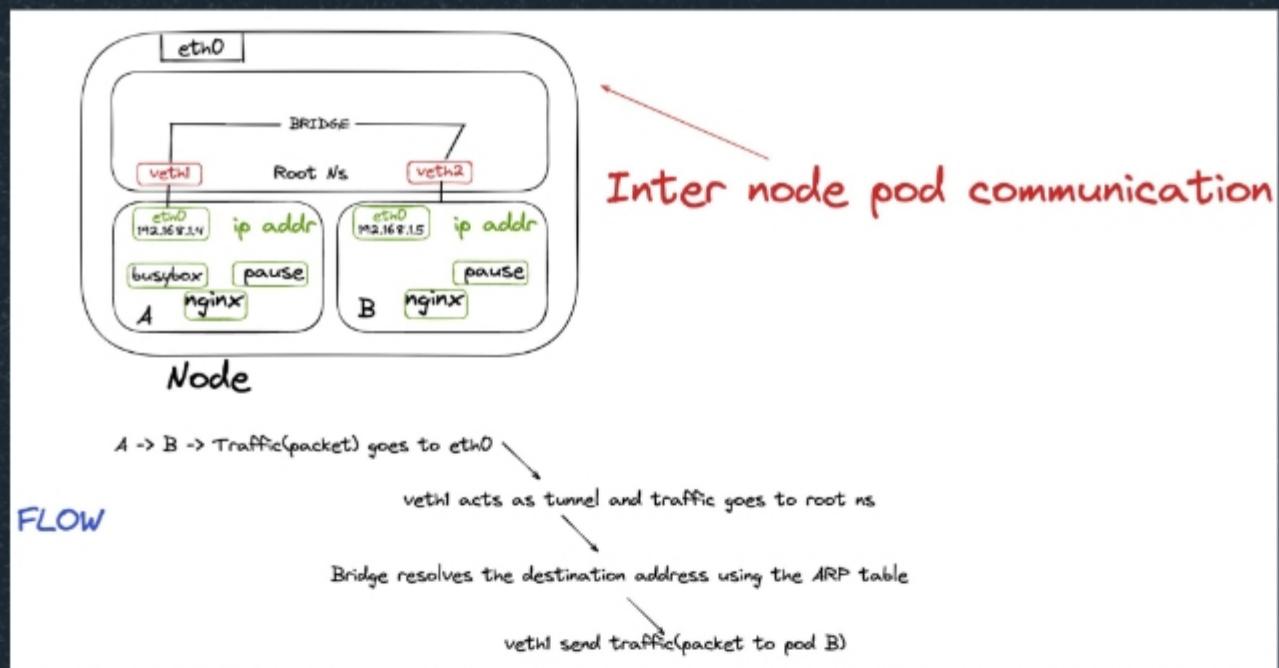
ls -lt /var/run/netns List of namespaces created

```
node01 $ ls -lt /var/run/netns
total 0
-r--r--r-- 1 root root 0 Jul 26 14:25 cni-3cf655f5-9131-e23d-8756-b7ab6536ef8f
-r--r--r-- 1 root root 0 May  8 19:46 cni-9e382128-d5db-0313-8fee-654c4be984b
-r--r--r-- 1 root root 0 May  8 19:46 cni-592cc23b-1d2b-0854-bf32-437fd23a84df
node01 $ ip netns exec cni-3cf655f5-9131-e23d-8756-b7ab6536ef8f ip link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT group default
    link/loopback brd 00:00:00:00:00:00 state UNKNOWN mode DEFAULT group default
3: eth@1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1460 qdisc noqueue state UP mode DEFAULT group default
    link/ether 9e:bd:dc:7d:4e:59 brd ff:ffff:ffff:ffff:ffff:ffff link-netnsid 0
node01 $ ip link | grep -A1 '^'
9: calic44014569391f31: <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:ffff:ffff:ffff:ffff:ffff link-netns cni-3cf655f5-9131-e23d-8756-b7ab6536ef8f
node01 $
```

Kubernetes



Kubernetes Series: #15 Internode Pod Communication



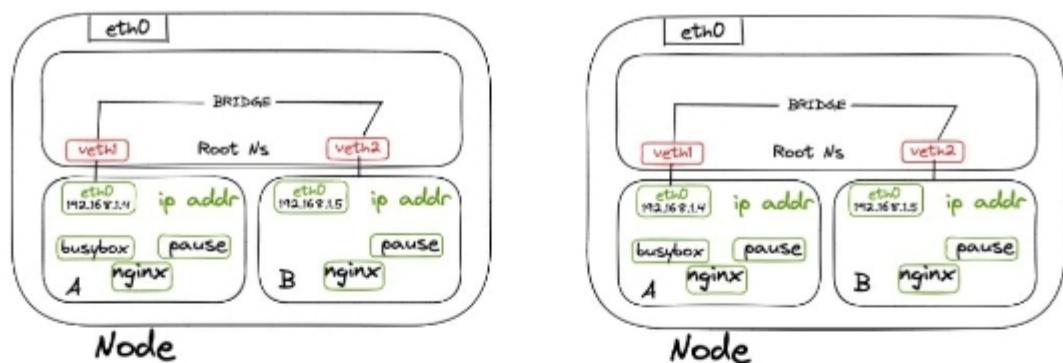
Kubernetes



Kubernetes Series: #16 Node-to-Node Communication

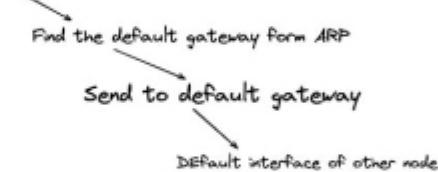
Node to node communication

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

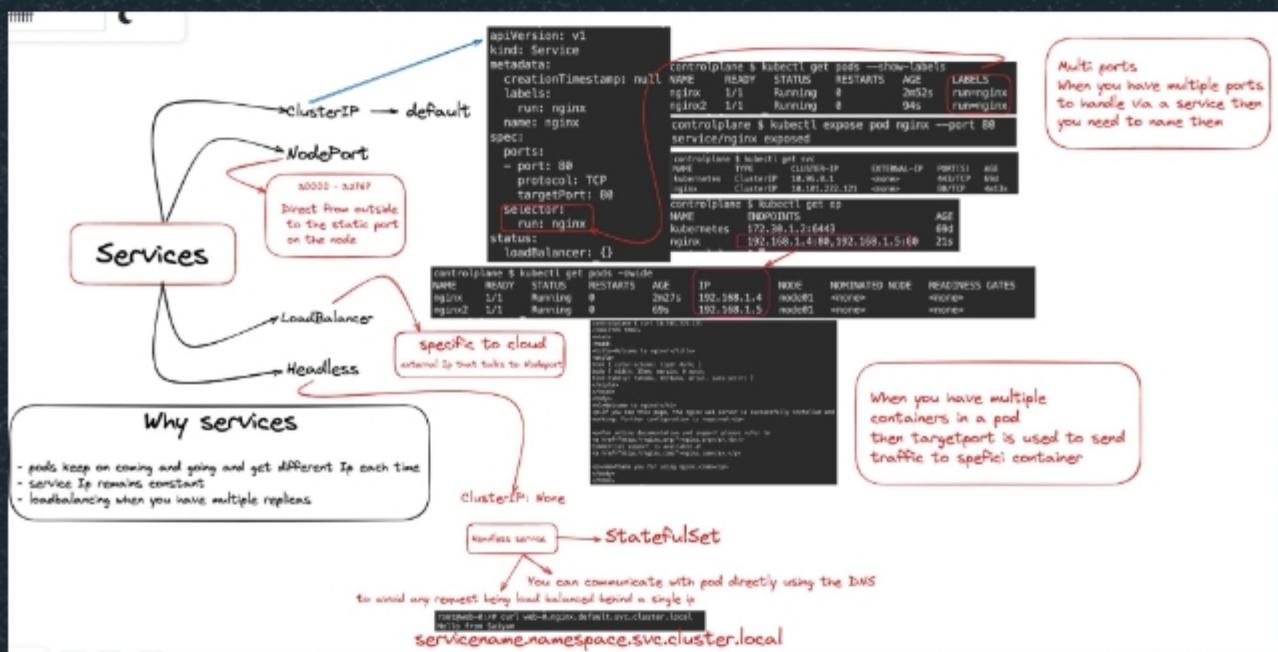


Services → iptables and netfilter

Kubernetes



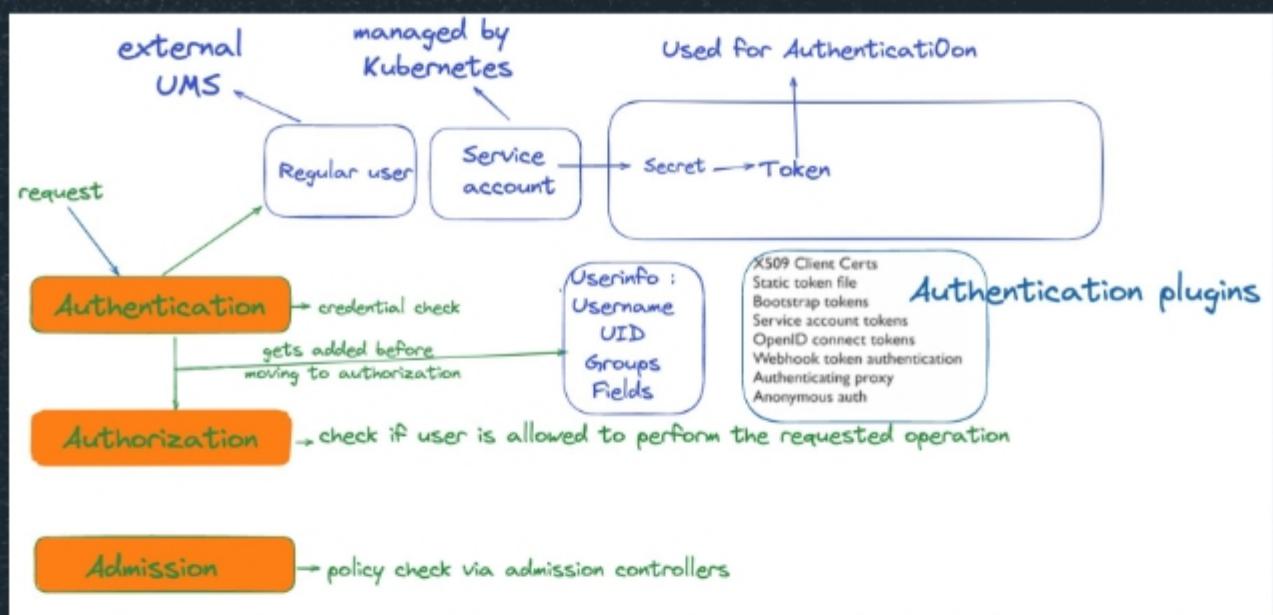
Kubernetes Series: #17 Services



Kubernetes



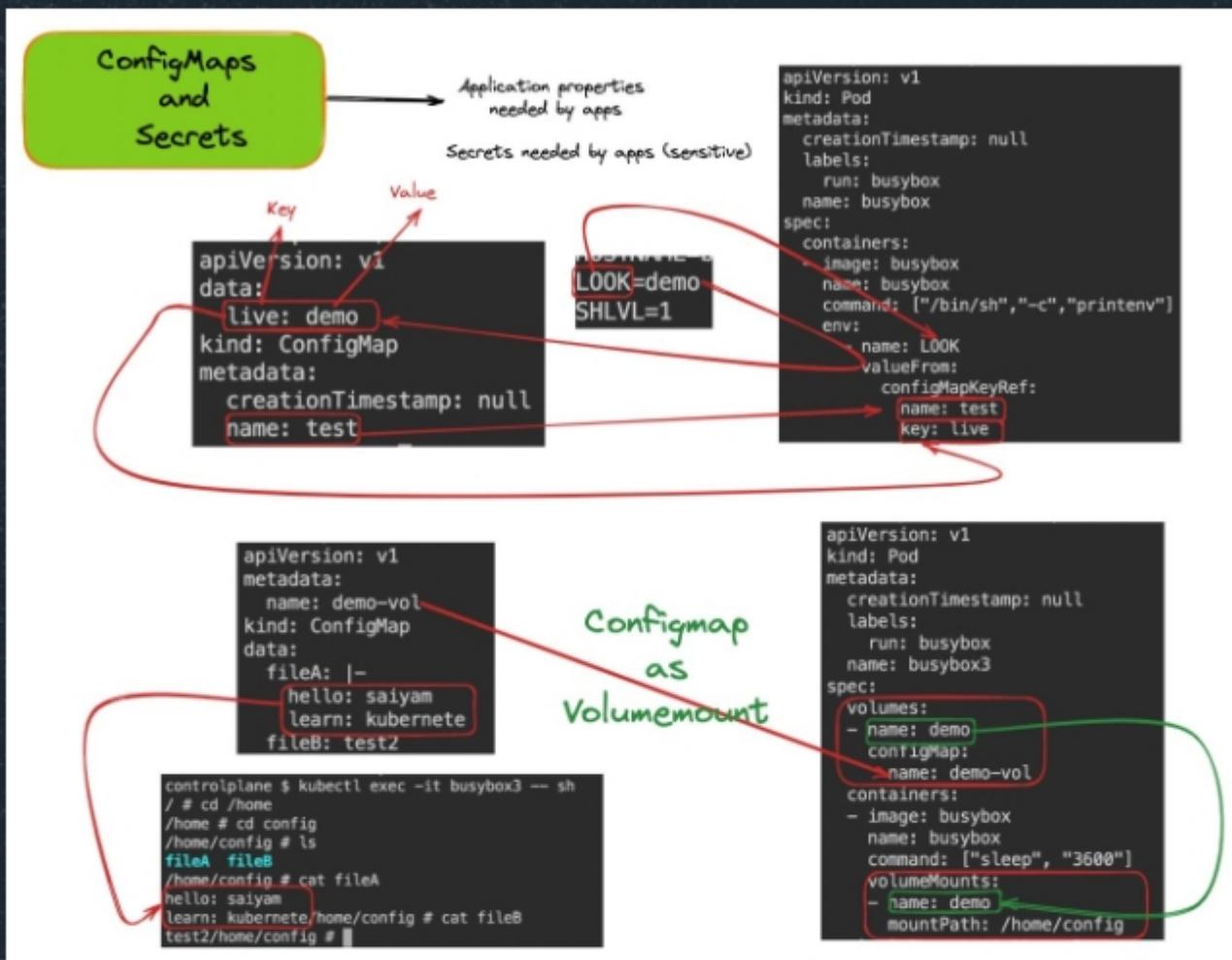
Kubernetes Series: #18 Authentication and Authorization



Kubernetes



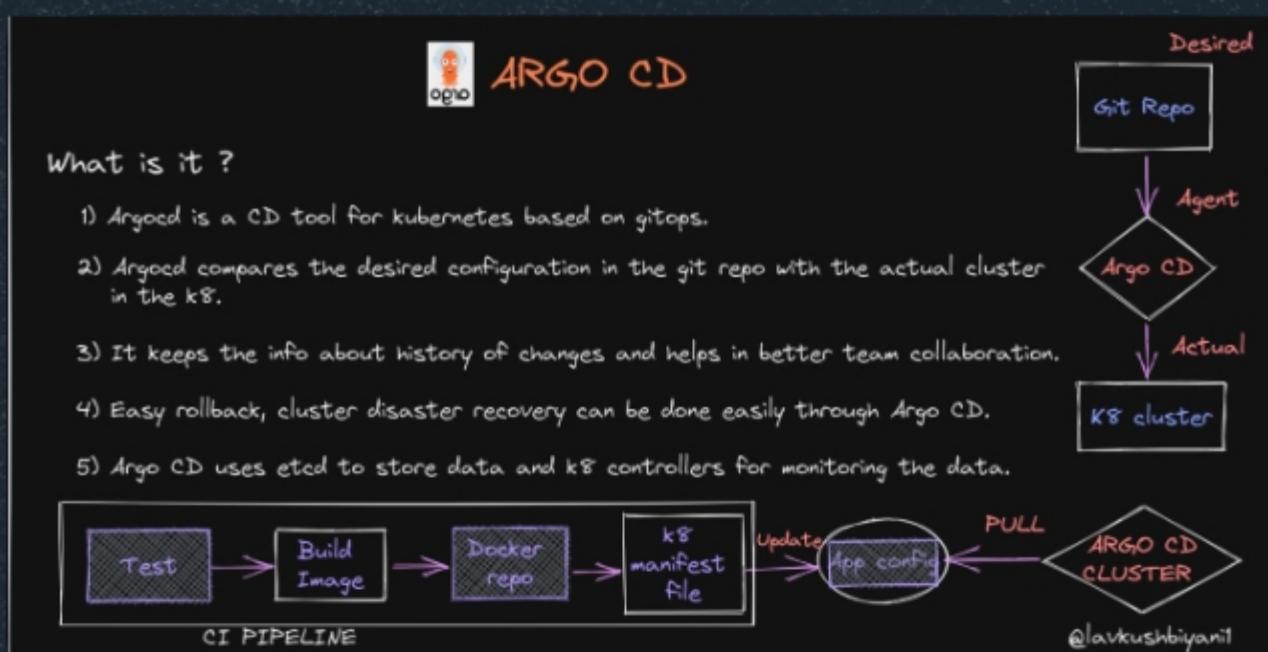
Kubernetes Series: #19 Configmaps & Secrets



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 blue Twitter bird icon with a yellow outline.

~Pavan Gudiwada

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

~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

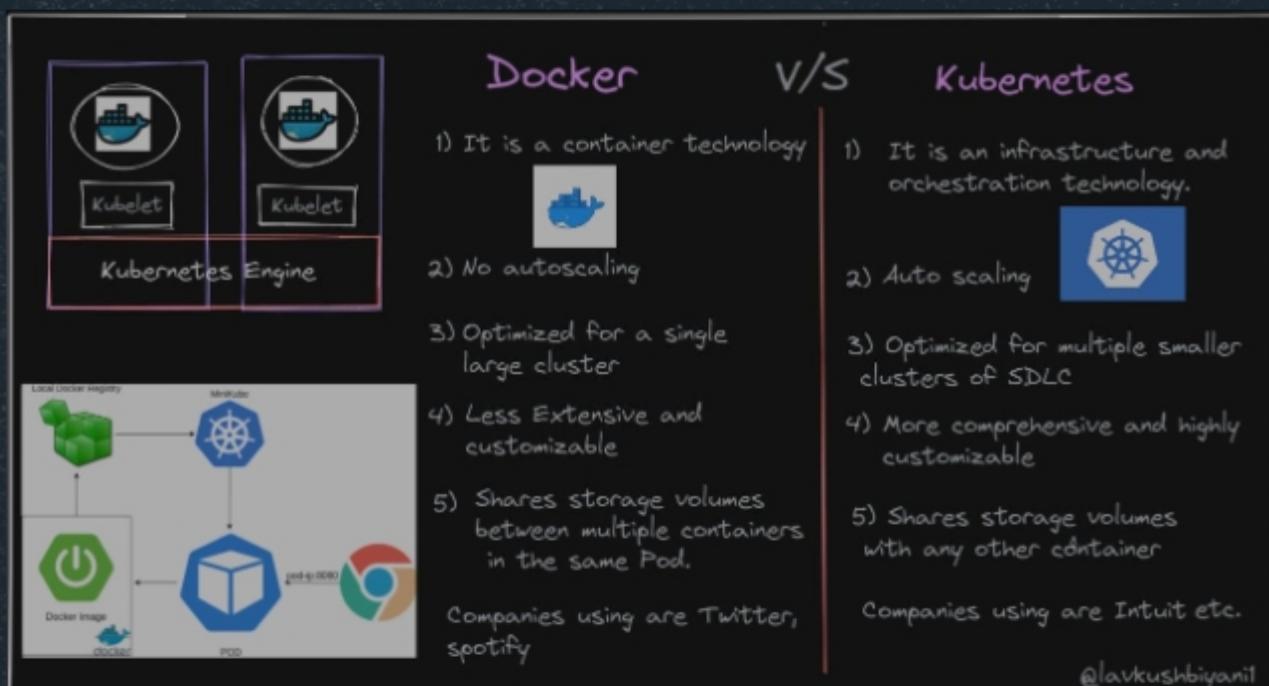
$ kubectl label namespace default istio-injection=enabled --overwrite
```

~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



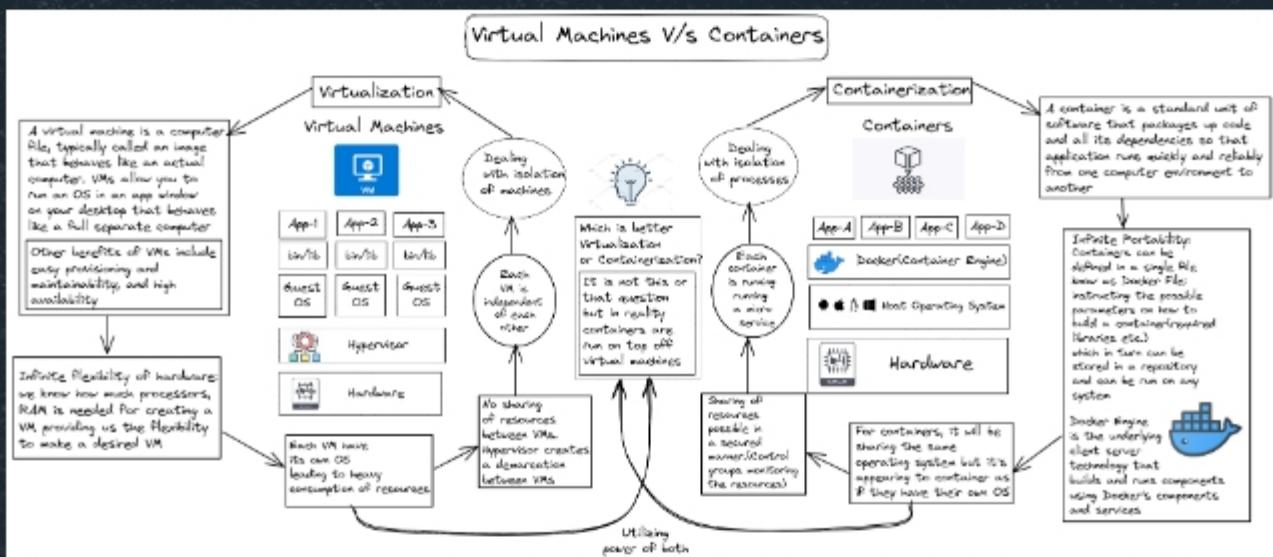
@vaidansh23

~Vaidansh Bhardwaj

Containerization & VMs



Simplifying Containers & VMs

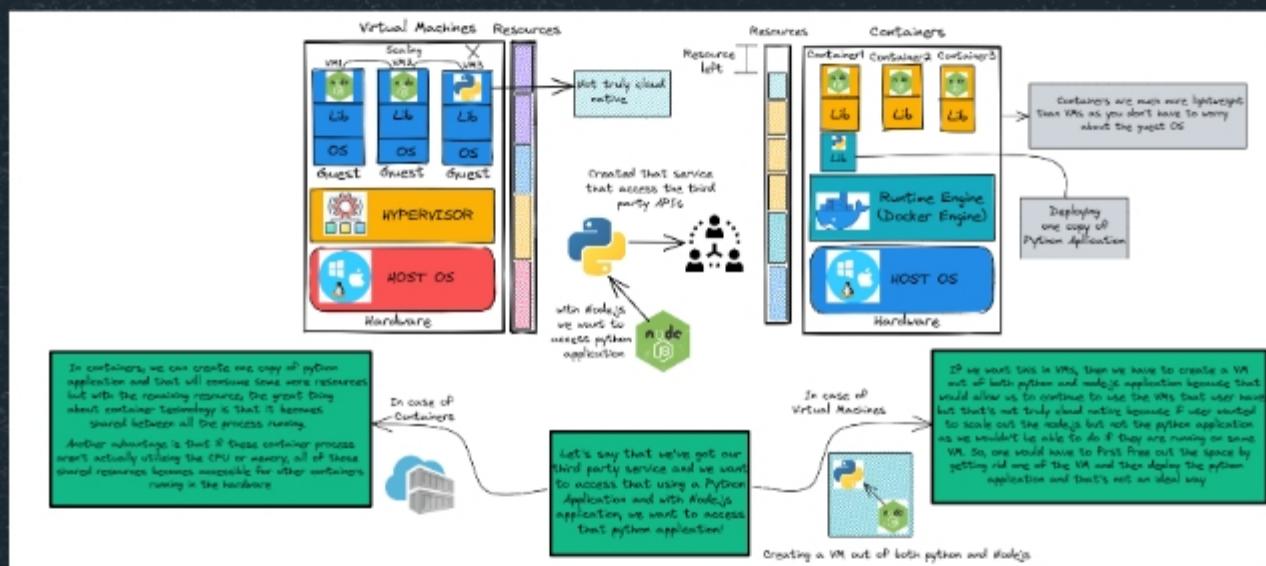


~Aviral Singh

Containerization & VMs



Resource Containers & VMs

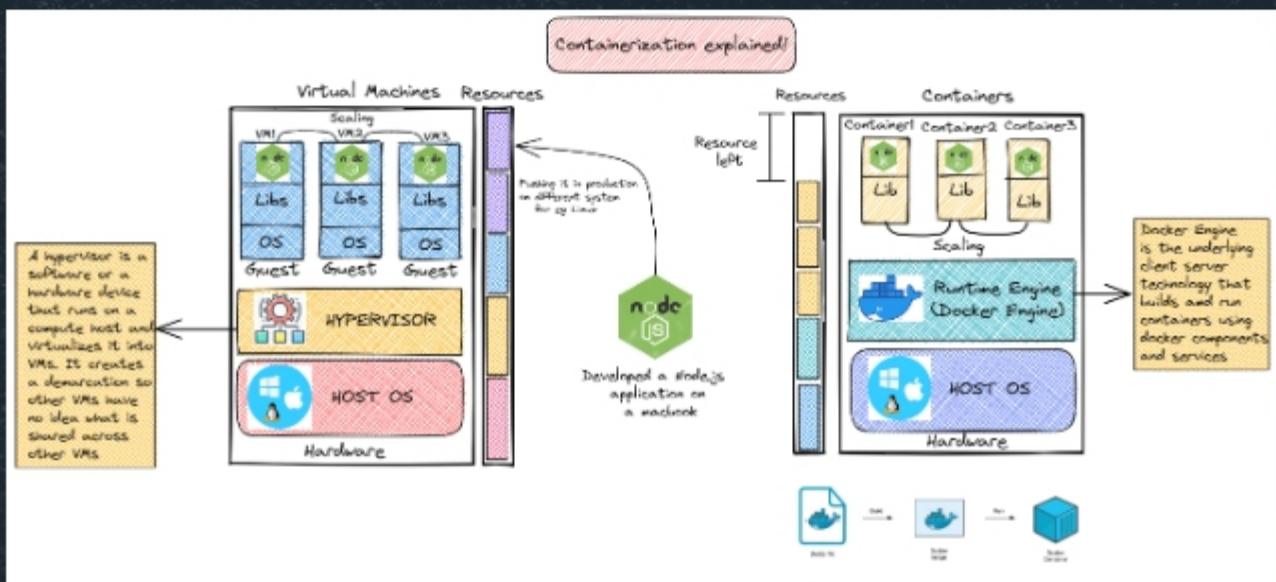


~Aviral Singh

Containerization & VMs



Resource Management

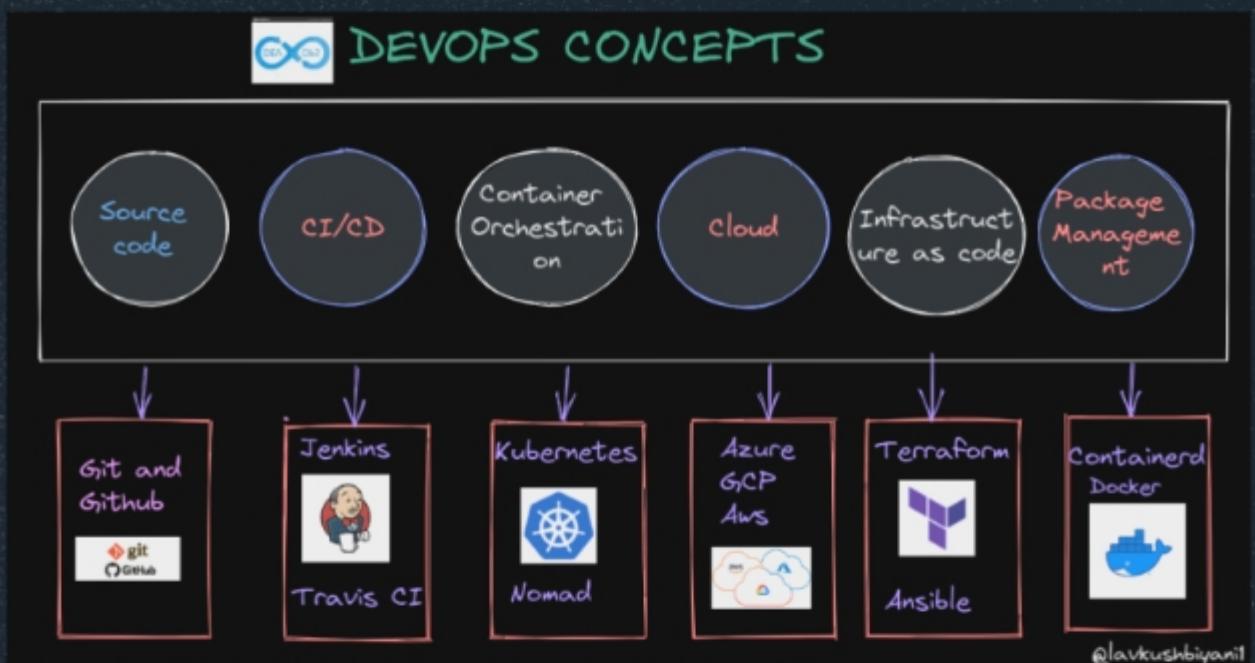


~Aviral Singh

DevOps



~An overview

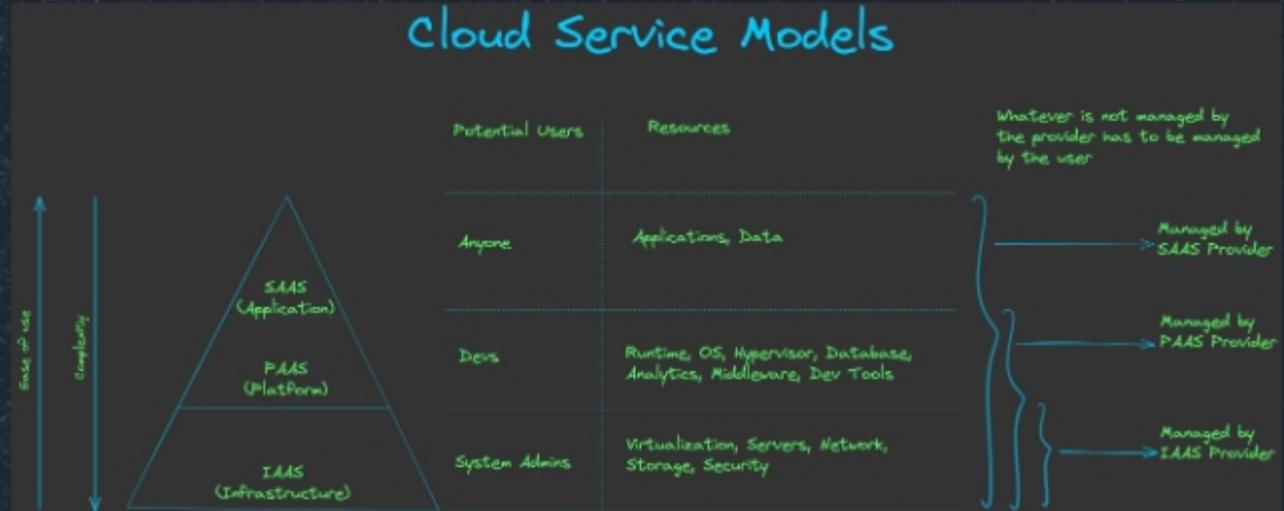


~Lavakush Biyani

Cloud Service Models



Cloud Service Models

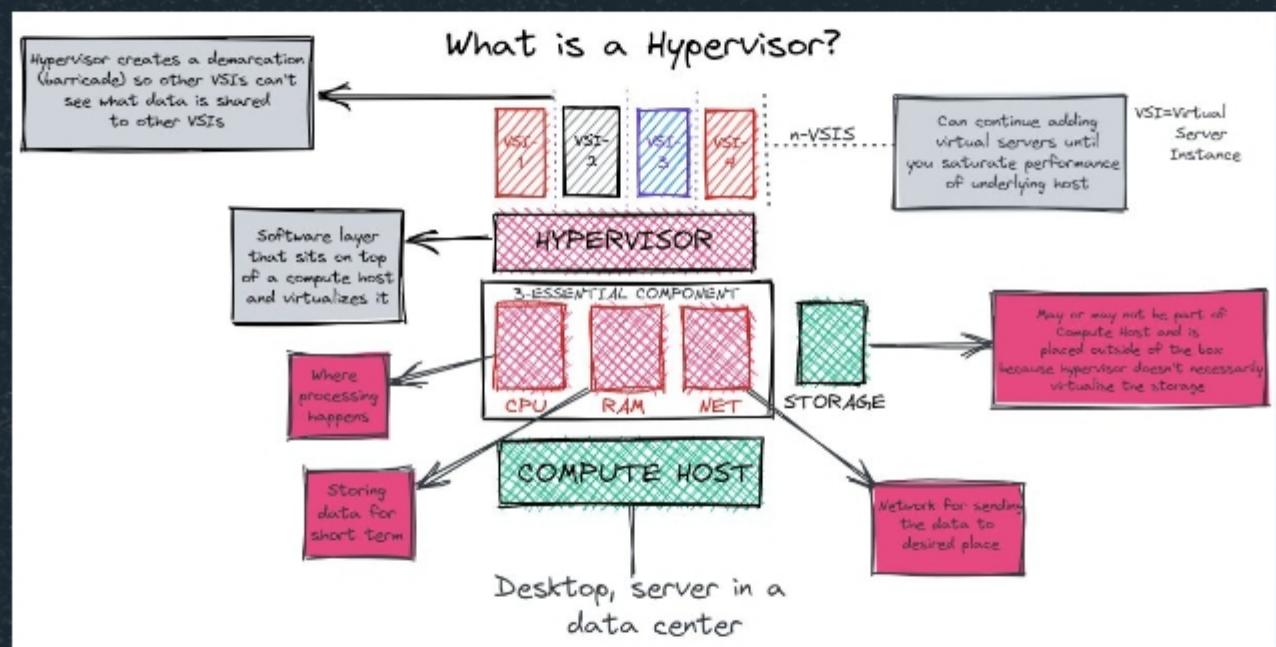


~Arnav Barman

Hypervisor



What is a Hypervisor

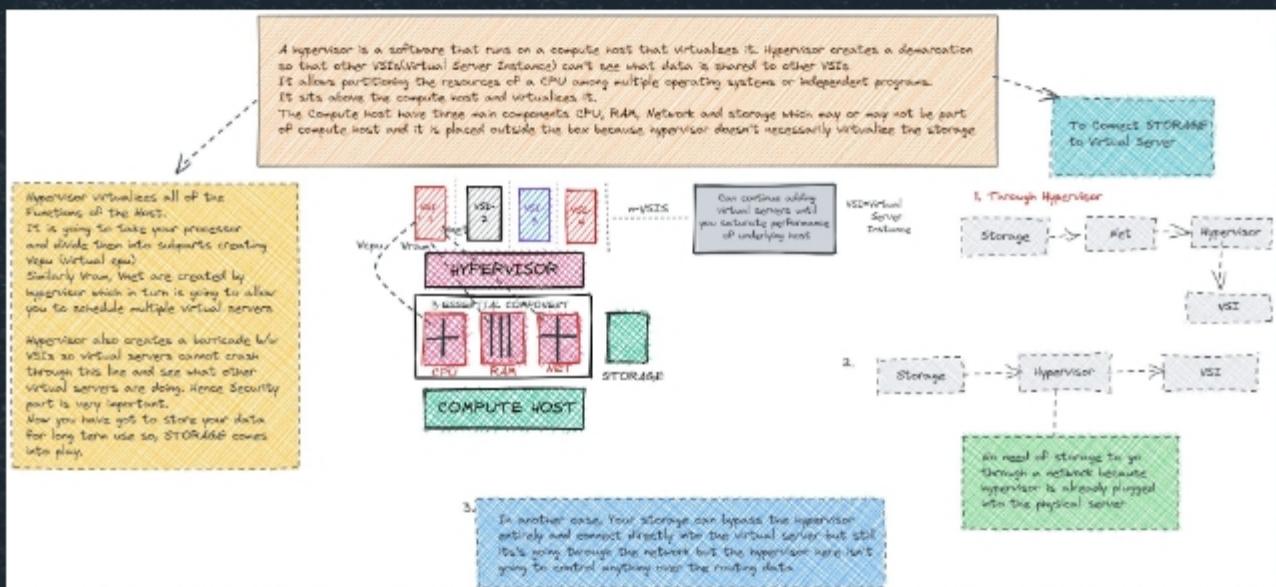


~Aviral Singh

Hypervisor



Hypervisor – VMs

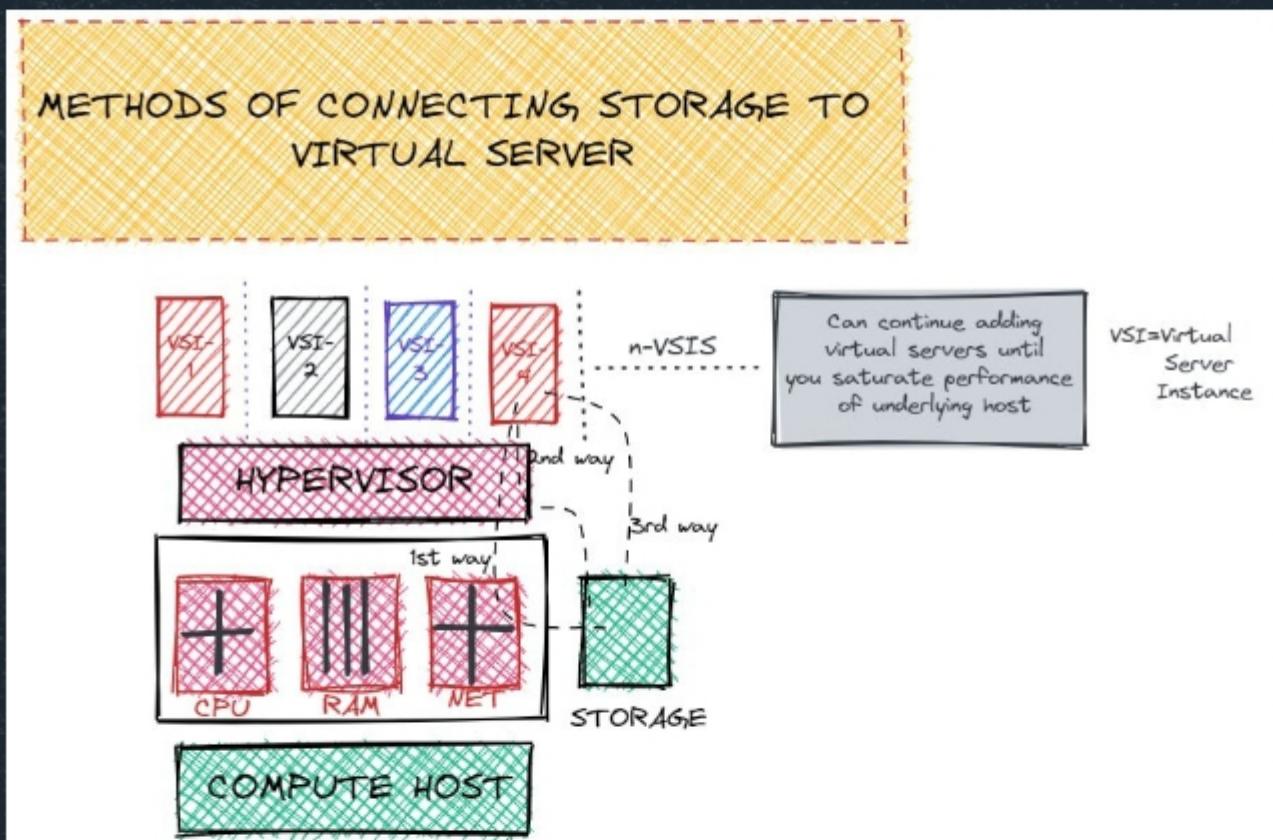


~Aviral Singh

Hypervisor



Connecting Storage to Virtual Server

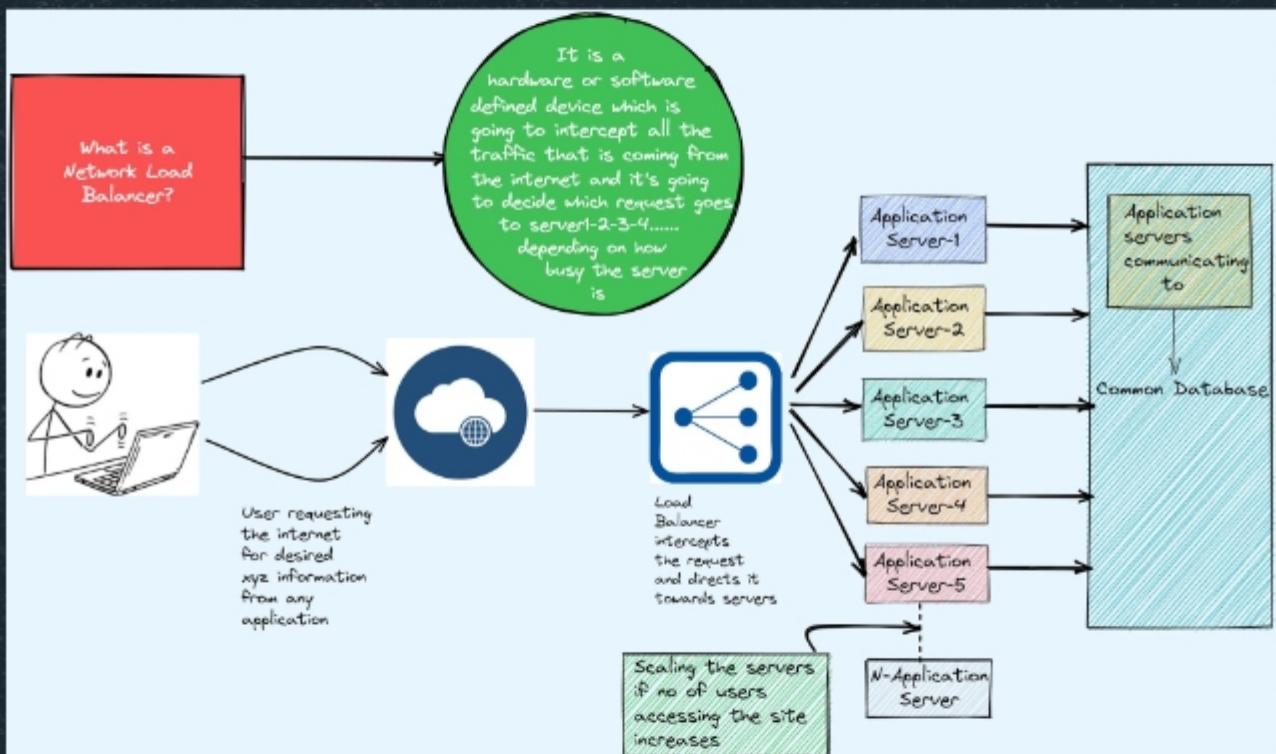


~Aviral Singh

Networking



Network Load Balancer #1

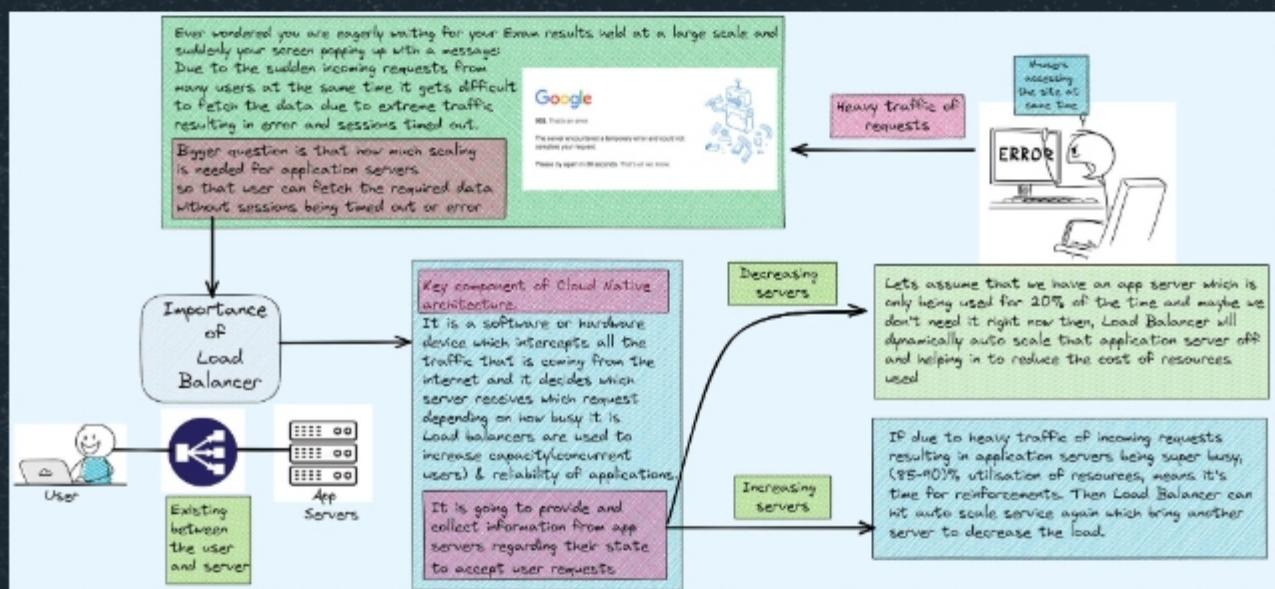


~Aviral Singh

Networking



Network Load Balancer #2

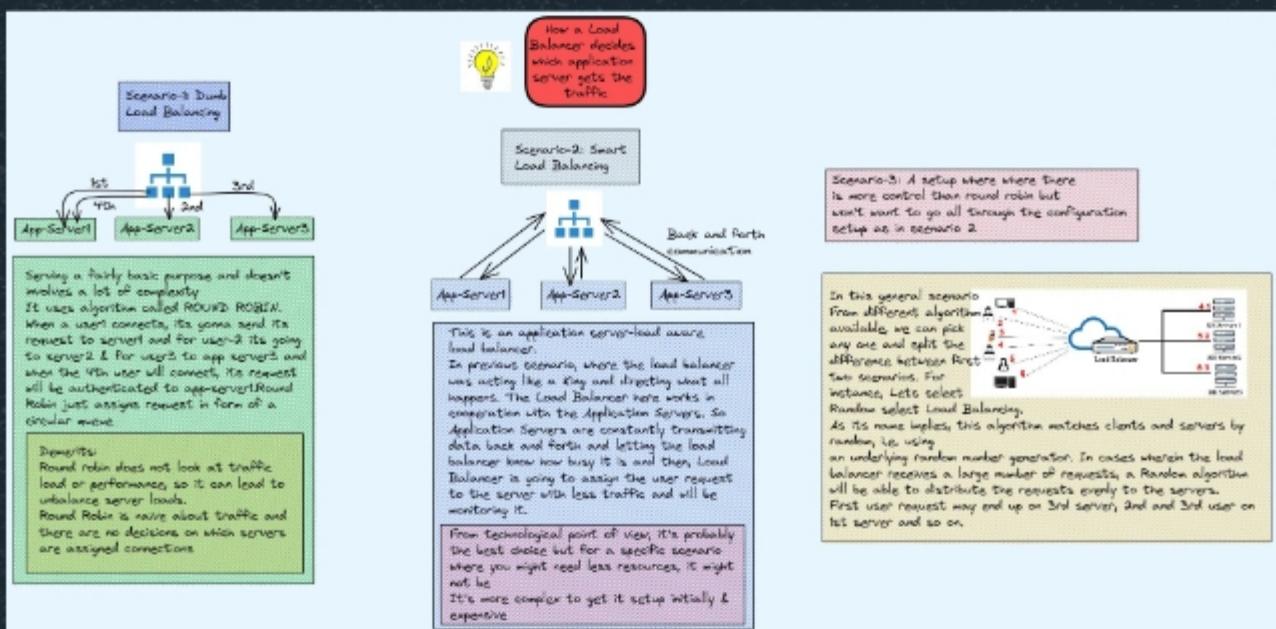


~Aviral Singh

Networking



Network Load Balancer #3

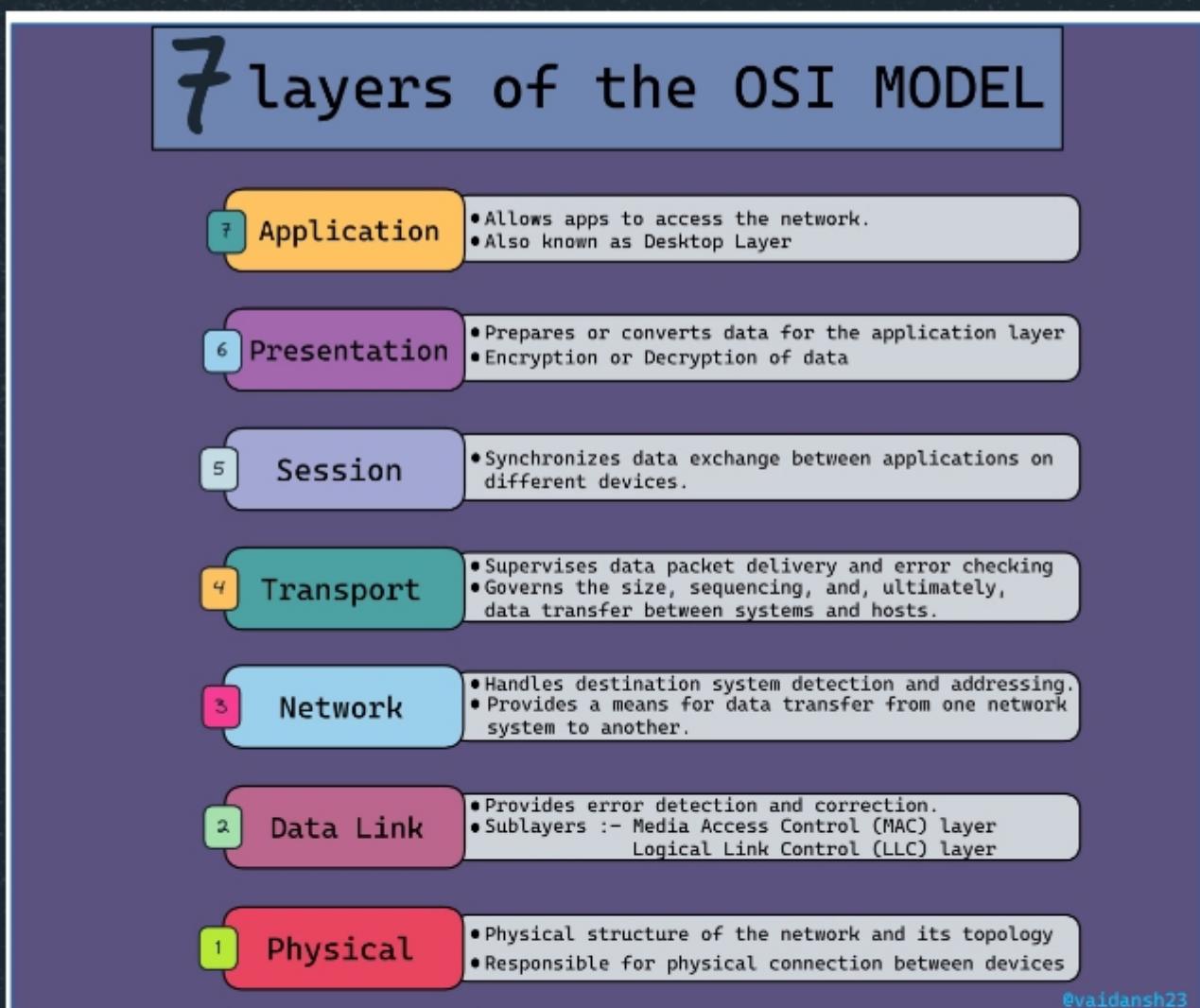


~Aviral Singh

Networking



OSI Model



@vaidansh23

~Vaidansh Bhardwaj

Prometheus



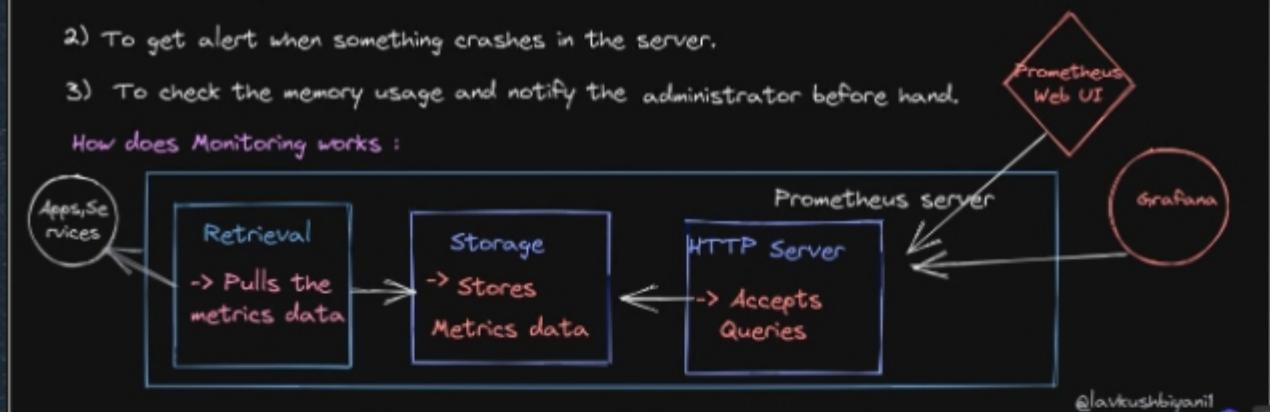
PROMETHEUS

-> 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 :



~Lavakush Biyani

Service Mesh

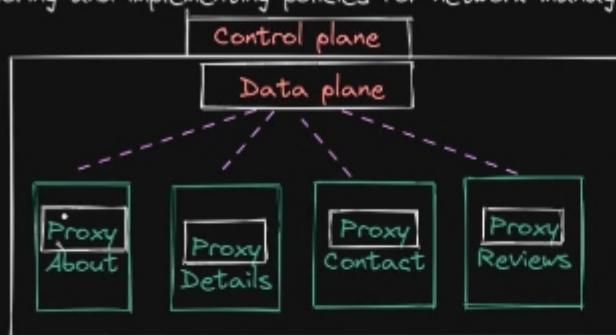


SERVICE MESHES

-> 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

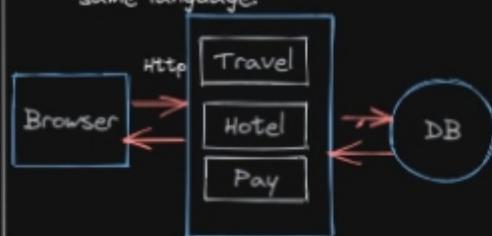
~Lavakush Biyani

Monolithic v/s Microservices

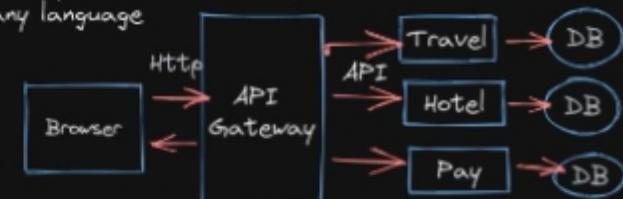


Monolithic V/s Microservices

- 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



@lavakushbiyani

~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

