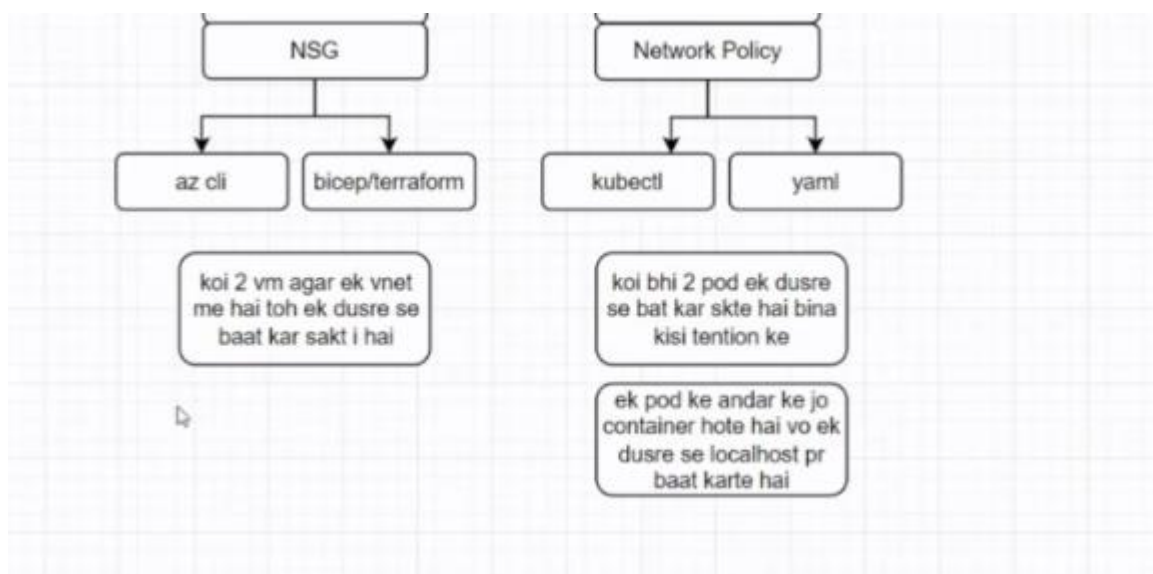
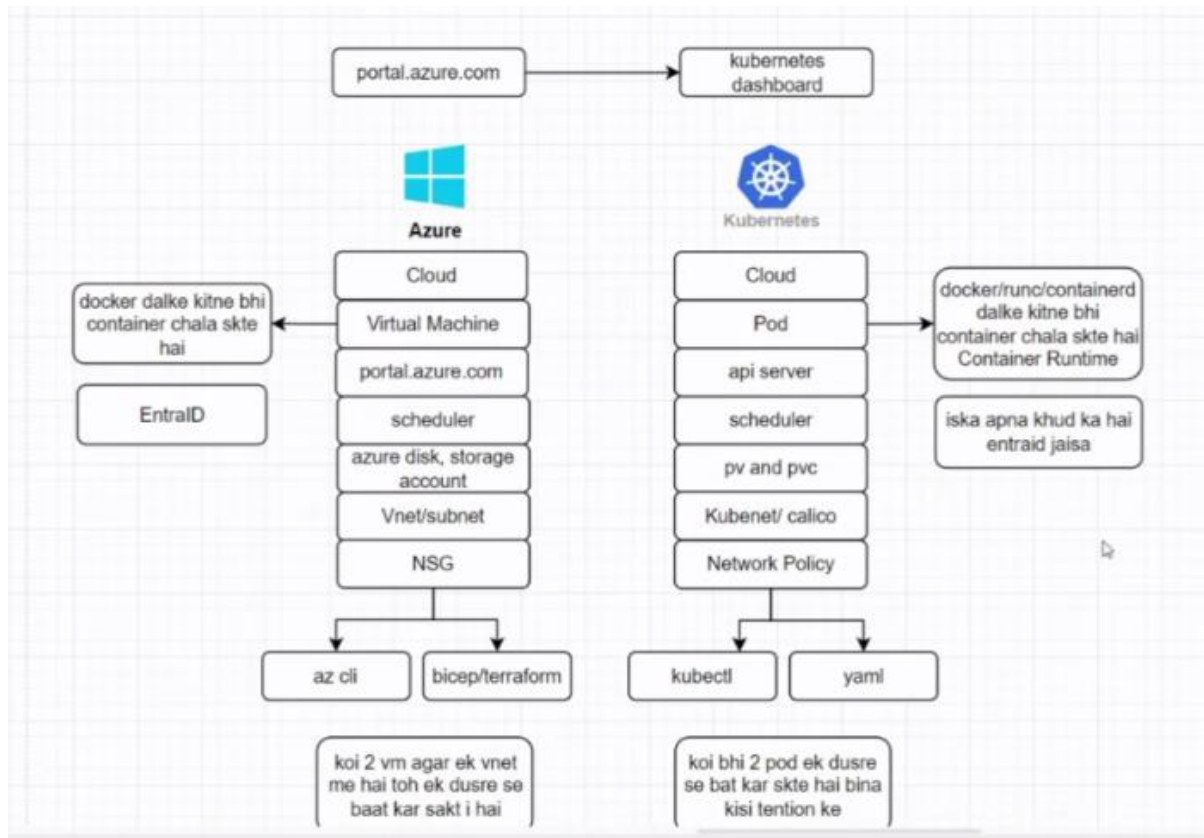
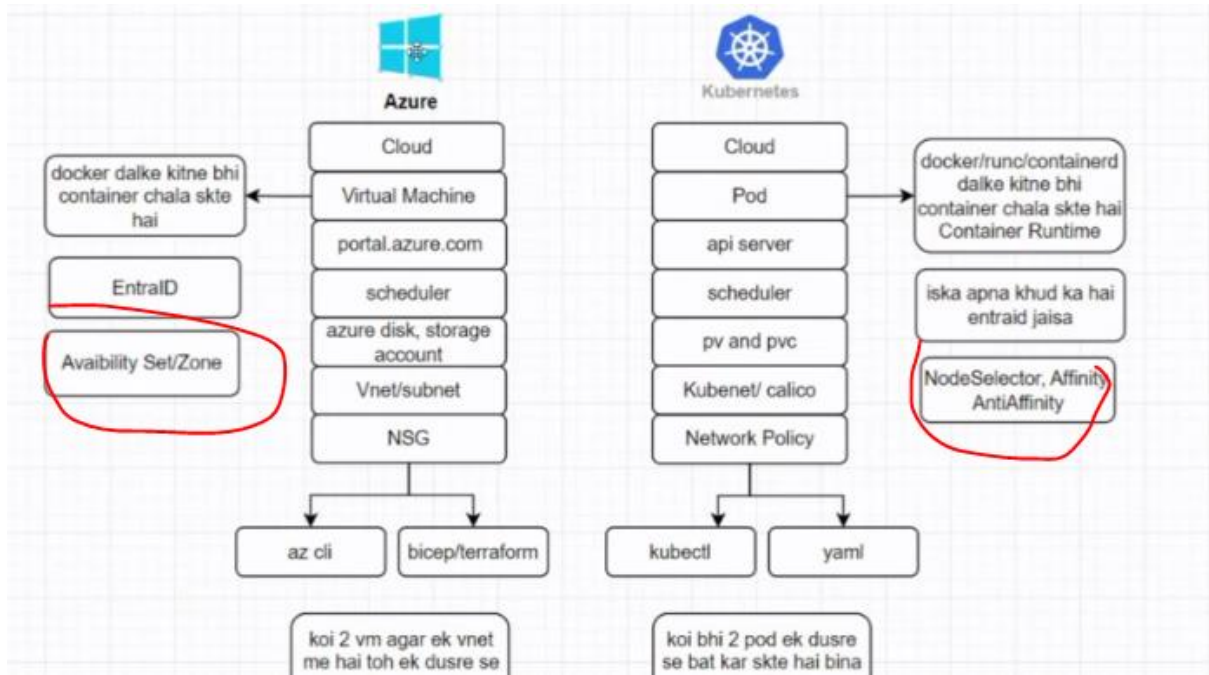


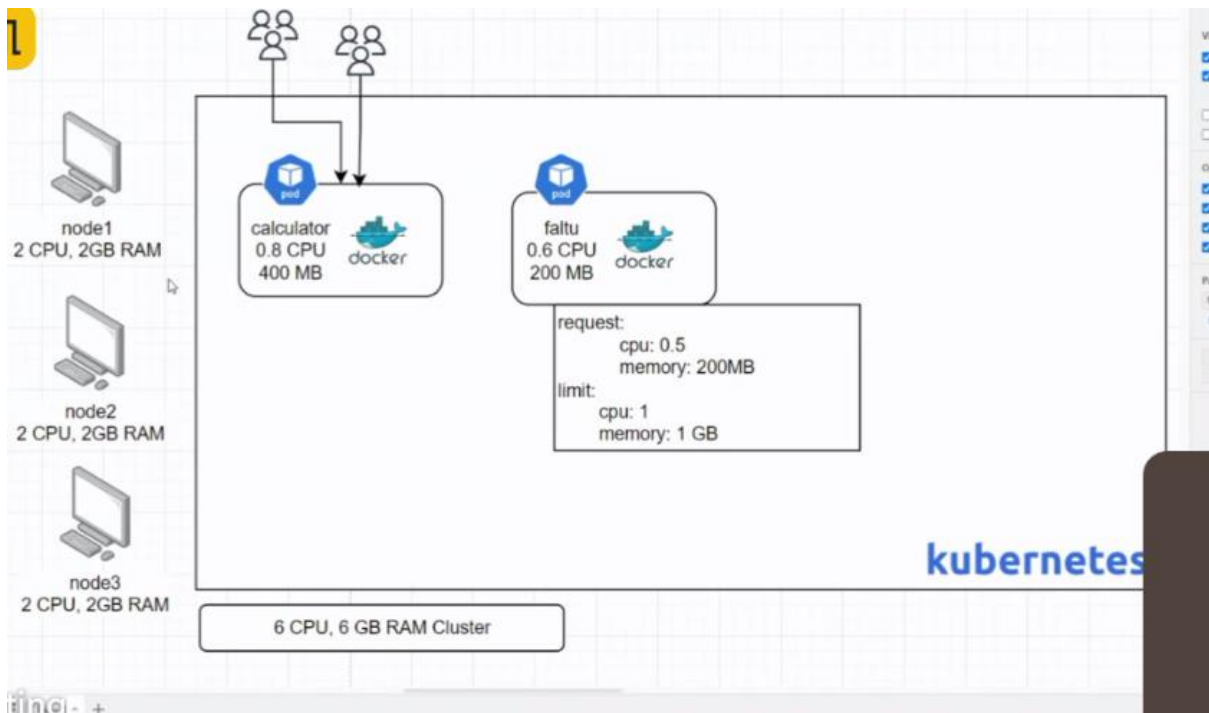
3 Nov

1)

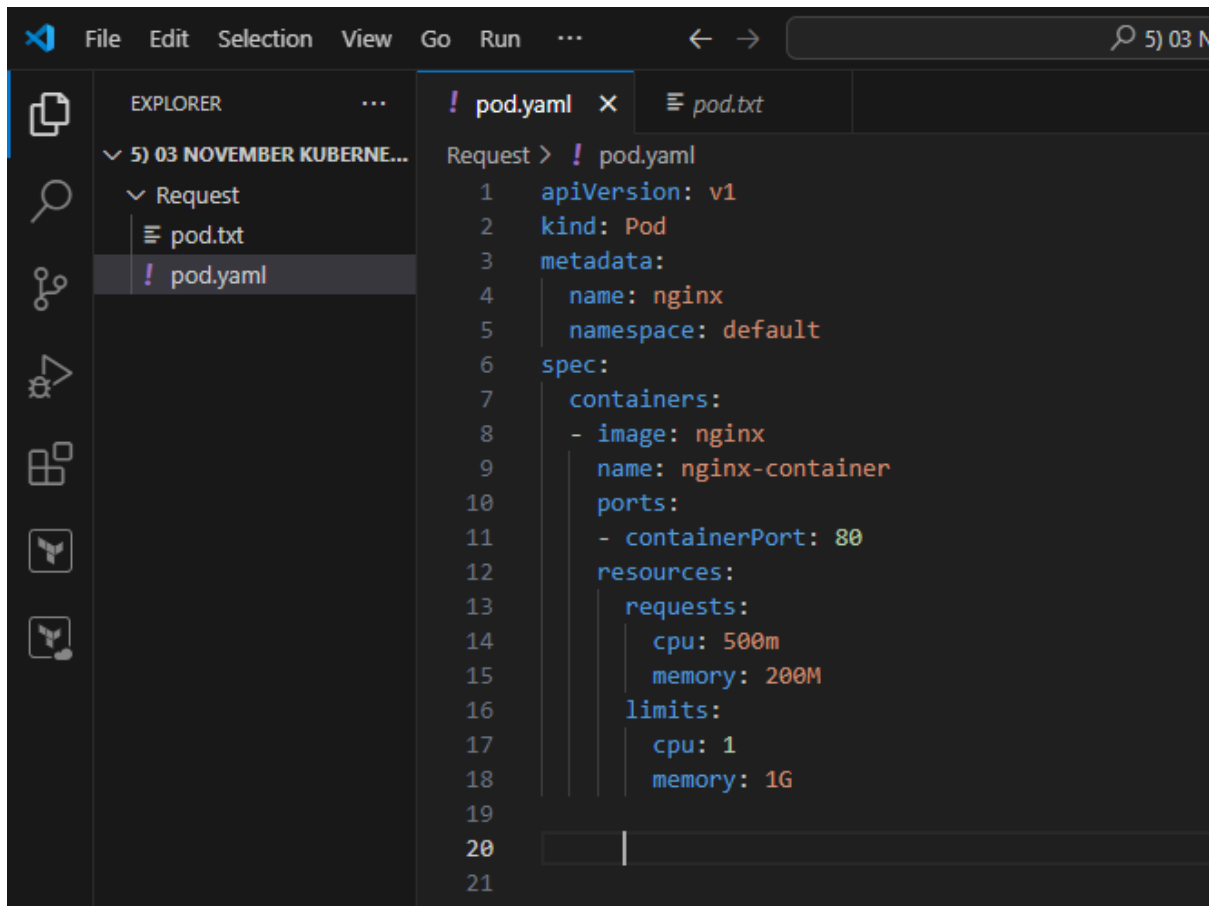




2) Difference between request and limit



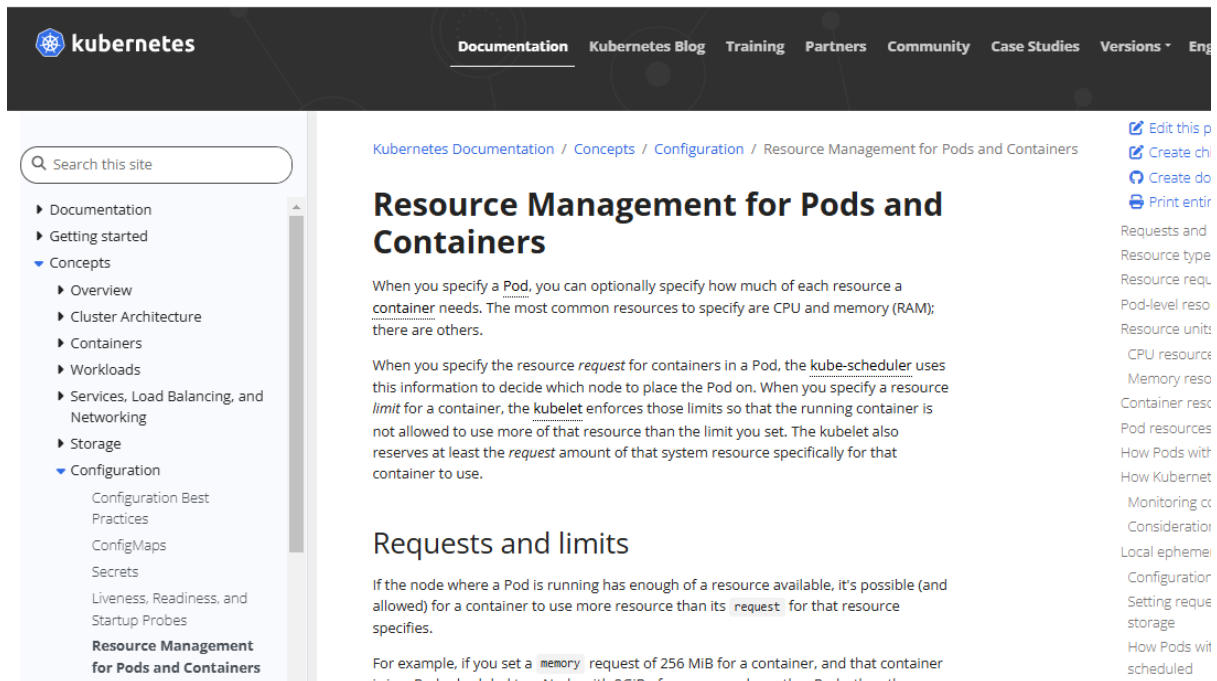
2) Create folder “5) 03 November Kubernetes” and create folder “Request” and then file “pod.yaml”.



The screenshot shows the Visual Studio Code interface. The Explorer sidebar on the left shows a folder named "5) 03 NOVEMBER KUBERNE..." containing a subfolder "Request" with two files: "pod.txt" and "pod.yaml". The "pod.yaml" file is selected and its content is displayed in the main editor. The file is a Kubernetes Pod manifest for an nginx container with resource requests and limits.

```
1  apiVersion: v1
2  kind: Pod
3  metadata:
4    name: nginx
5    namespace: default
6  spec:
7    containers:
8      - image: nginx
9        name: nginx-container
10       ports:
11         - containerPort: 80
12       resources:
13         requests:
14           cpu: 500m
15           memory: 200M
16         limits:
17           cpu: 1
18           memory: 1G
```

3) search = <https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/>



The screenshot shows the Kubernetes documentation website. The main heading is "Resource Management for Pods and Containers". The page explains how to specify resource requests and limits for containers in a Pod. It mentions that the kube-scheduler uses this information to decide which node to place the Pod on, and the kubelet enforces those limits. The page also includes a section titled "Requests and limits" which further elaborates on the concept.

Resource Management for Pods and Containers

When you specify a `Pod`, you can optionally specify how much of each resource a container needs. The most common resources to specify are CPU and memory (RAM); there are others.

When you specify the resource `request` for containers in a Pod, the `kube-scheduler` uses this information to decide which node to place the Pod on. When you specify a resource `limit` for a container, the `kubelet` enforces those limits so that the running container is not allowed to use more of that resource than the limit you set. The `kubelet` also reserves at least the `request` amount of that system resource specifically for that container to use.

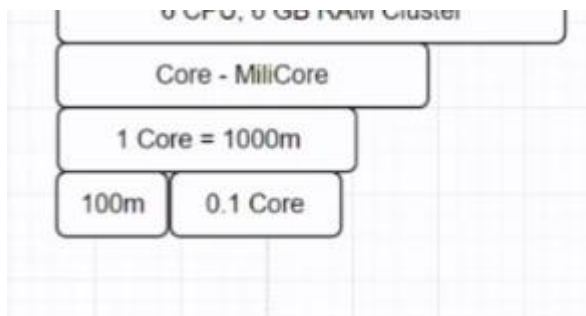
Requests and limits

If the node where a Pod is running has enough of a resource available, it's possible (and allowed) for a container to use more resource than its `request` for that resource specifies.

For example, if you set a `memory` request of 256 MiB for a container, and that container is in a Pod scheduled to a Node with 8GiB of memory and no other Pods, then the

- `spec.containers[].resources.limits.cpu`
- `spec.containers[].resources.limits.memory`
- `spec.containers[].resources.limits.hugepages-<size>`
- `spec.containers[].resources.requests.cpu`
- `spec.containers[].resources.requests.memory`
- `spec.containers[].resources.requests.hugepages-<size>`

Kubernetes, 1 CPU unit is equivalent to 1 physical CPU core, or 1 virtual core depending on whether the node is a physical host or a virtual machine



4) **kubectl apply -f pod.yaml** = create pod

```
PS C:\4) KUBERNETES\5) 03 November Kubernetes\Request> kubectl apply -f pod.yaml
pod/nginx created
PS C:\4) KUBERNETES\5) 03 November Kubernetes\Request>
```

5) **kubectl get pods**

```
PS C:\4) KUBERNETES\5) 03 November Kubernetes\Request> kubectl get pods
NAME    READY   STATUS    RESTARTS   AGE
nginx   1/1     Running   0           2m11s
PS C:\4) KUBERNETES\5) 03 November Kubernetes\Request>
```

6) **kubectl top pod** = pod kitni memory kha raha hai

```
PS C:\4) KUBERNETES\5) 03 November Kubernetes\Request> kubectl top pod
NAME    CPU(cores)   MEMORY(bytes)
nginx   0m           3Mi
PS C:\4) KUBERNETES\5) 03 November Kubernetes\Request>
```

7) SEARCH = stress utility linux

<https://www.geeksforgeeks.org/linux-stress-command-with-examples/>

8) **kubectl exec nginx -c nginx-container -i -t -- bash** = go inside container

```
PS C:\4) KUBERNETES\5) 03 November Kubernetes\Request> kubectl exec nginx -c nginx-container -i -t -- bash
root@nginx:/#
```

9) **apt update**

apt install stress

10) stress --vm 1 --vm-bytes 150M

```
root@nginx:/# stress --vm 1 --vm-bytes 150M
stress: info: [154] dispatching hogs: 0 cpu, 0 io, 1 vm, 0 hdd
[]
```

11) kubectl top pod

```
NAME      CPU(cores)  MEMORY(bytes)
nginx     631m        153Mi
PS C:\4) KUBERNETES\5) 03 November Kubernetes\Request> kubectl top pod

NAME      CPU(cores)  MEMORY(bytes)
nginx     994m        110Mi
PS C:\4) KUBERNETES\5) 03 November Kubernetes\Request> kubectl top pod

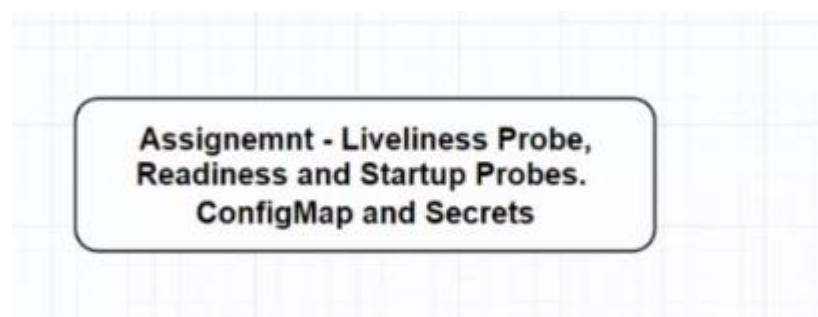
NAME      CPU(cores)  MEMORY(bytes)
nginx     994m        110Mi
PS C:\4) KUBERNETES\5) 03 November Kubernetes\Request>
```

12) jyada stress se container mar gaya tha. OOM = out of memory

```
Selecting previously unselected package stress.
(Reading database ... 7580 files and directories currently installed.)
Preparing to unpack .../stress_1.0.7-1_amd64.deb ...
Unpacking stress (1.0.7-1) ...
Setting up stress (1.0.7-1) ...
root@nginx:/# stress --vm 1 --vm-bytes 400M
stress: info: [156] dispatching hogs: 0 cpu, 0 io, 1 vm, 0 hdd
command terminated with exit code 137
PS C:\DevOpsInsiders\Batch15\azure-devsecops-batch-15\CodeSamples\Kubernetes\requestlimits>

mples\Kubernetes\requestlimits> kubectl get pod
NAME      READY   STATUS    RESTARTS   AGE
nginx     1/1     Running   1 (4s ago)  10m
PS C:\DevOpsInsiders\Batch15\azure-devsecops-batch-15\CodeSamples\Kubernetes\requestlimits> kubectl get pod -w
NAME      READY   STATUS    RESTARTS   AGE
nginx     1/1     Running   1 (18s ago)  11m
nginx     0/1     OOMKilled 1 (46s ago)  11m
nginx     0/1     CrashLoopBackOff 1 (11s ago)  11m
nginx     1/1     Running   2 (14s ago)  11m
```

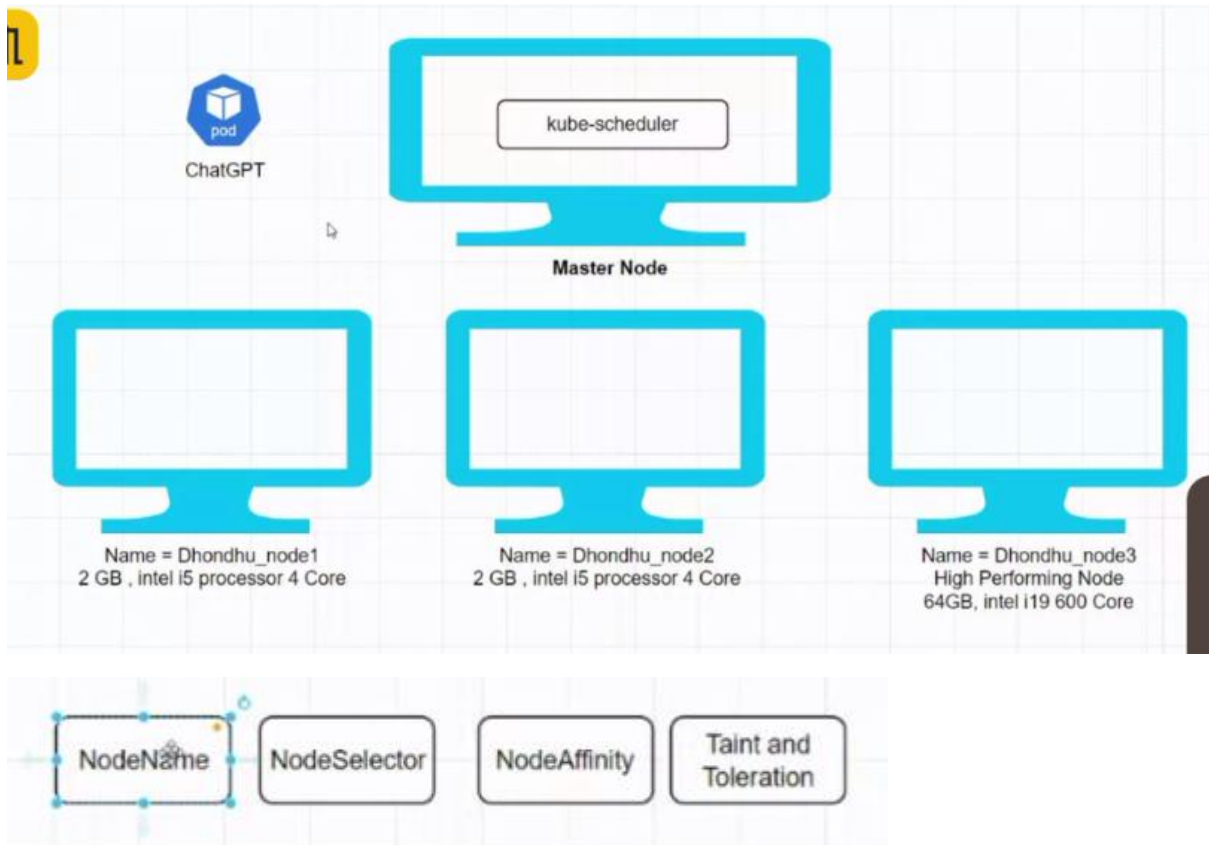
NOTE: 1) What is CrashLoopBackoff error



+++++

AGENDA – SCHEDULING

1)



2) nodeSelector: put label papa: dhondhu

```
spec:
  nodeSelector:
    papa: dhondhu
```

3) in cluster go to nodepools to set matching label

agentpool | Overview

Node pool

Search

Upgrade Kubernetes | Update image | Scale node pool | Delete | Refresh | Give feedback

Overview

Nodes

Configuration

Essentials

Provisioning state	: Succeeded	Cluster	: k8sladdu
Power state	: Running (2/2 nodes ready)	Operating system	: Ubuntu Linux
Availability zones	: None	Kubernetes version	: 1.30.6
Mode	: System	Node count	: 2 nodes
		Node size	: Standard_DS2_v2

Properties | **Monitoring**

Node pool

Max pods per node	110
Public IPs per node	Disabled
Autoscaling	Enabled

Taints and labels

Taints (edit)	None
Labels (edit)	None

Edit Labels

×

Edit the labels for your node pool.

Kubernetes labels handle the scheduling rules for nodes. Applying labels to a node pool will apply them to all nodes in the node pool. [Learn more](#)

Labels

Labels are key/value pairs that can be used to categorize or add identifying information to Kubernetes resources. [Learn more](#)

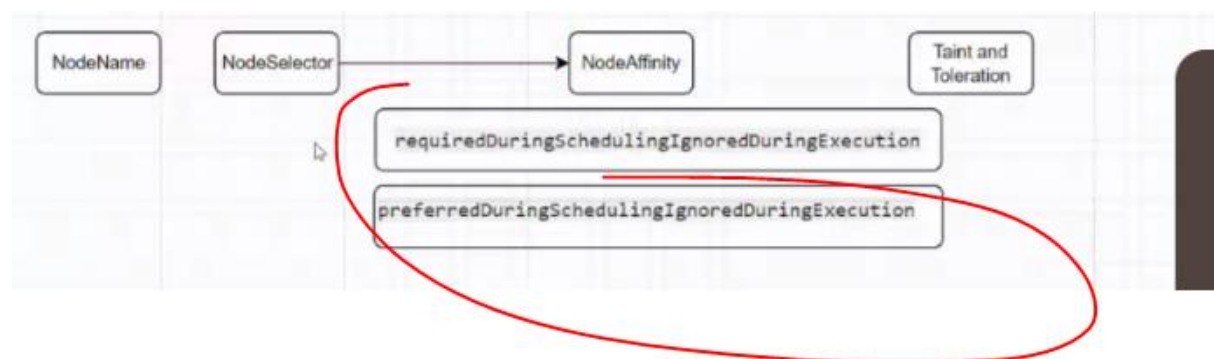
Key	Value	
papa	dhondhu	🗑️

Save

Cancel

Give feedback

4) NODE AFFINITY = Advanced version of node selector



+++++

AGENDA – TAINT & TOLERATION

agentpool | Overview

Node pool

Search x < Upgrade Kubernetes Update image Scale node pool Delete Refresh Give feedback

Overview

Nodes

Configuration

Essentials

Provisioning state	: Succeeded	Cluster	: k8sladdu
Power state	: Running (2/2 nodes ready)	Operating system	: Ubuntu Linux
Availability zones	: None	Kubernetes version	: 1.30.6
Mode	: System	Node count	: 2 nodes
		Node size	: Standard_DS2_v2

Properties

Node pool

Max pods per node	110
Public IPs per node	Disabled
Autoscaling	Enabled
Azure Spot Instance	Disabled
Maximum price	N/A
Scale eviction policy	N/A
Node image version	AKSUbuntu-2204gen2containerd-202412.04.0

Taints and labels

Taints (edit)	None
Labels (edit)	papa : dhondhu



toleration



Name = Dhondhu_node2
2 GB , intel i5 processor 4 Core



Label

key: value



Taint

socks: badbu: NoSchedule

Kubernetes Scheduling me ungli karne ke lie...

nodeName

NodeSelector

Node Affinity -
Preferred and
Required

Taint And Toleration

CODE DEKHLO LAST Ke VIDEO SE