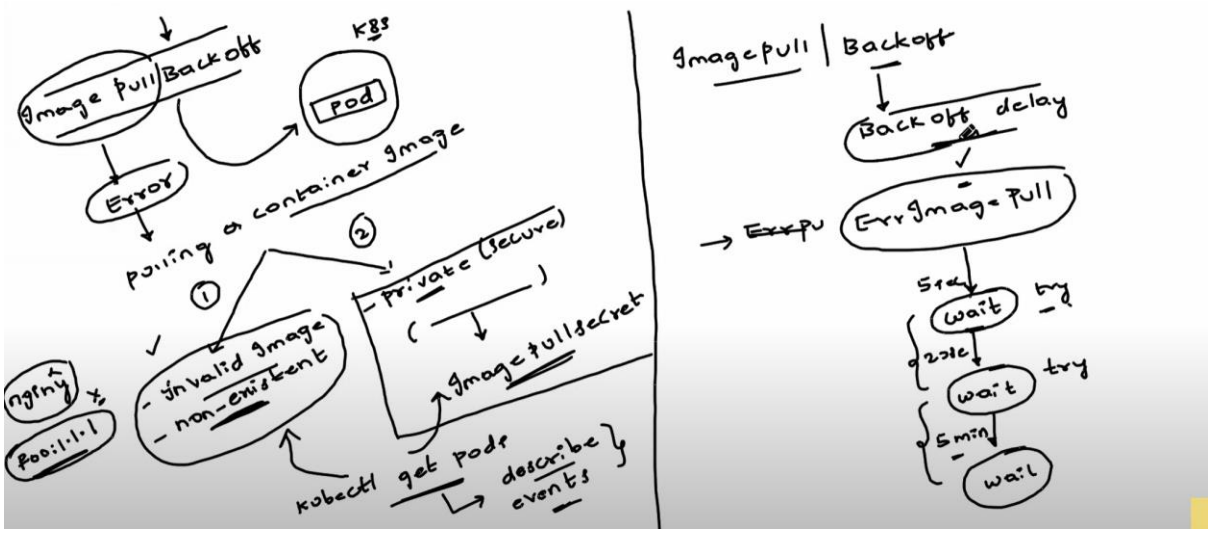
Kubernetes Trouble Shooting

Repo: https://github.com/iam-veeramalla/kubernetes-troubleshooting-zero-to-hero

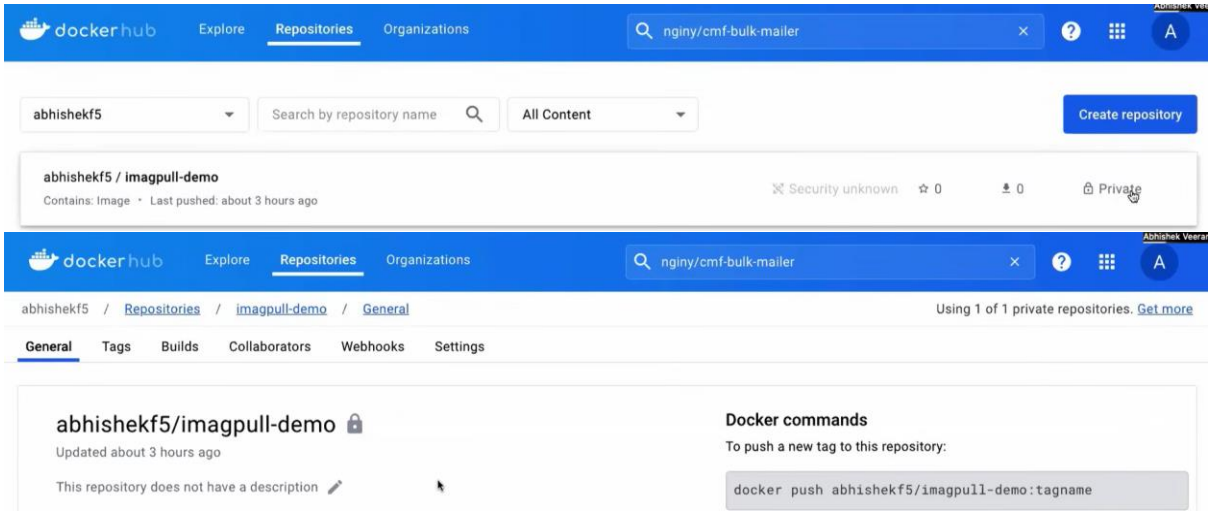
Day 1: Issue1: ImagePullBackoff

Error: We will get this error when we try to deploy pod into the cluster. If the Image we mentioned is not pulled then we will receive this error. There are two cases of receiving this error.



1st Case: If the image we mentioned is not valid or not existing then we will receive this error.

2nd Case: If the Image is listed as private then also we will get this **ImagePullBackoff** error.



This **ImagePullBackOff** is two parts. **BackOff** means once it tried to pull an image and fails then it waits for sometime to pull again. If failed again it increase the wait time and tries again.

Create a Minikube cluster:

1st Case:

```
abhishekveeramalla@aveerama-mac 01-ImagePullBackOff % minikube status
minikube
type: Control Plane
host: Running
kubelet: Running
apiserver: Running
kubeconfig: Configured
```

```
abhishekveeramalla@aveerama-mac 01-ImagePullBackOff % kubectl get deploy
No resources found in default namespace.
abhishekveeramalla@aveerama-mac 01-ImagePullBackOff % vim nginx-deploy.yaml
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

```
abhishekveeramalla@aveerama-mac 01-ImagePullBackOff % kubectl apply -f nginx-deploy.yaml
deployment.apps/nginx-deployment created
abhishekveeramalla@aveerama-mac 01-ImagePullBackOff % kubectl get pods
NAME                                READY   STATUS              RESTARTS   AGE
nginx-deployment-86dcfdf4c6-4lf4p   0/1     ContainerCreating   0          13s
nginx-deployment-86dcfdf4c6-n8ltt   0/1     ContainerCreating   0          13s
nginx-deployment-86dcfdf4c6-q2xls   0/1     ContainerCreating   0          13s
abhishekveeramalla@aveerama-mac 01-ImagePullBackOff % kubectl get pods -w
NAME                                READY   STATUS    RESTARTS   AGE
nginx-deployment-86dcfdf4c6-4lf4p   1/1     Running   0          49s
nginx-deployment-86dcfdf4c6-n8ltt   1/1     Running   0          49s
nginx-deployment-86dcfdf4c6-q2xls   1/1     Running   0          49s
```

All pods are in running status. Deployment got success. Let's delete this deployment and modify the deployment file:

```
abhishekveeramalla@aveerama-mac 01-ImagePullBackOff % kubectl get deploy
NAME               READY   UP-TO-DATE   AVAILABLE   AGE
nginx-deployment   3/3     3            3           67s
abhishekveeramalla@aveerama-mac 01-ImagePullBackOff % kubectl delete deploy nginx-deployment
deployment.apps "nginx-deployment" deleted
```

Modify the Image name:

```
abhishekveeramalla@aveerama-mac 01-ImagePullBackOff % vim nginx-deploy.yaml
  spec:
    containers:
    - name: nginx
      image: nginy:1.14.2
      ports:
      - containerPort: 80
```

if you want to use specific dockerhub registry then we need to mention it. Otherwise it will take dockerhub registry by default.

```
  spec:
    containers:
    - name: nginx
      image: docker.io/nginy:1.14.2
      ports:
      - containerPort: 80
```

```
abhishekveeramalla@aveerama-mac 01-ImagePullBackOff % kubectl apply -f nginx-deploy.yaml
deployment.apps/nginx-deployment created
abhishekveeramalla@aveerama-mac 01-ImagePullBackOff % kubectl get pods -w
NAME                                READY   STATUS              RESTARTS   AGE
nginx-deployment-66ccc45bd7-bc6v2   0/1     ErrImagePull        0          13s
nginx-deployment-66ccc45bd7-hpczv   0/1     ContainerCreating   0          13s
nginx-deployment-66ccc45bd7-ssfjf   0/1     ContainerCreating   0          13s
nginx-deployment-66ccc45bd7-hpczv   0/1     ErrImagePull        0          14s
nginx-deployment-66ccc45bd7-ssfjf   0/1     ErrImagePull        0          17s
nginx-deployment-66ccc45bd7-hpczv   0/1     ImagePullBackOff    0          25s
nginx-deployment-66ccc45bd7-bc6v2   0/1     ImagePullBackOff    0          26s
nginx-deployment-66ccc45bd7-ssfjf   0/1     ImagePullBackOff    0          28s
ErrImagePull
```

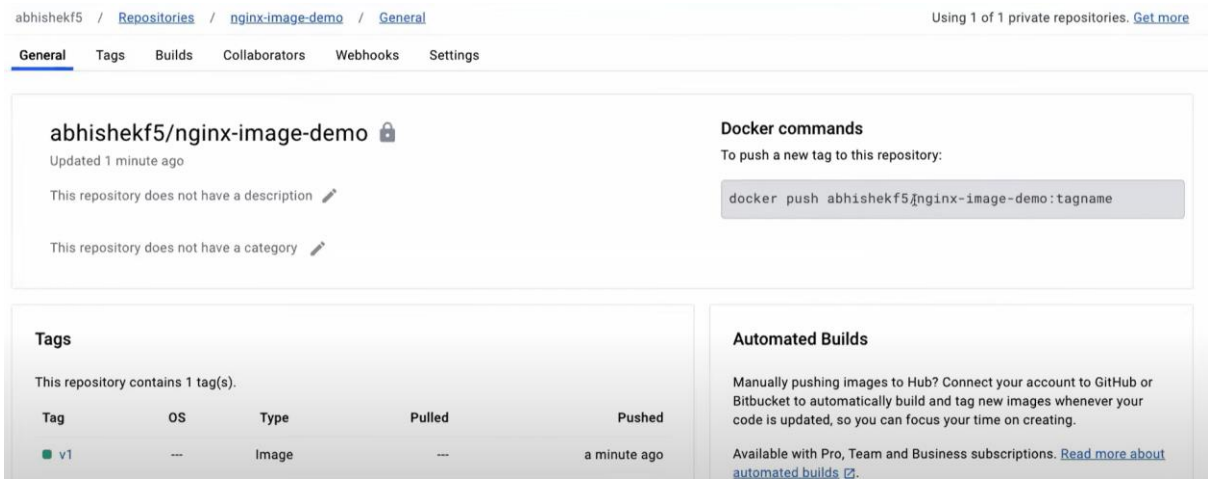
2nd Case:

You can make one repo as private in your dockerhub. Or you can pull one docker image from dockerhub and then push that image inot your dockerhub and make it private.

For the second way you need to have docker installed in your machine.

```
abhishekveeramalla@aveerama-mac 01-ImagePullBackOff % docker pull nginx:1.14.2
1.14.2: Pulling from library/nginx
27833a3ba0a5: Pull complete
0f23e58bd0b7: Pull complete
```

```
abhishekveeramalla@aveerama-mac 01-ImagePullBackOff % docker tag nginx:1.14.2 abhishekf5/nginx-image-demo:v1
abhishekveeramalla@aveerama-mac 01-ImagePullBackOff % docker push abhishekf5/nginx-image-demo:v1
The push refers to repository [docker.io/abhishekf5/nginx-image-demo]
82ae01d5004e: Mounted from library/nginx
b8f18c3b860b: Mounted from library/nginx
```



Now edit the deployment file that we edited before with this new private image from our dockerhub.

Delete the previous nginx deployment.

```
abhishekveeramalla@aveerama-mac 01-ImagePullBackOff % kubectl delete deploy nginx-deployment
deployment.apps "nginx-deployment" deleted
abhishekveeramalla@aveerama-mac 01-ImagePullBackOff % kubectl apply -f nginx-deploy.yaml
deployment.apps/nginx-deployment created
```

We will get the error to pull the private image

```
abhishekveeramalla@aveerama-mac 01-ImagePullBackOff % kubectl get pods -w
NAME                                READY   STATUS             RESTARTS   AGE
nginx-deployment-588bb445c7-9hk7s   0/1     ContainerCreating   0          6s
nginx-deployment-588bb445c7-rf56w   0/1     ContainerCreating   0          7s
nginx-deployment-588bb445c7-vpphq   0/1     ContainerCreating   0          6s
nginx-deployment-588bb445c7-vpphq   0/1     ErrImagePull        0          16s
nginx-deployment-588bb445c7-rf56w   0/1     ErrImagePull        0          20s
nginx-deployment-588bb445c7-9hk7s   0/1     ErrImagePull        0          22s
```

We need to create a secret.

Ref: https://github.com/iam-veeramalla/kubernetes-troubleshooting-zero-to-hero/blob/main/01-ImagePullBackOff/02-private-image.md

Create a Secret by providing Docker credentials on the command line

```
kubectl create secret docker-registry demo --docker-server=<your-registry-server> --docker-username=<your-name> --docker-passwor
```

If the image is in Amazon ECR

Create a Secret by providing AWS ECR credentials on the command line

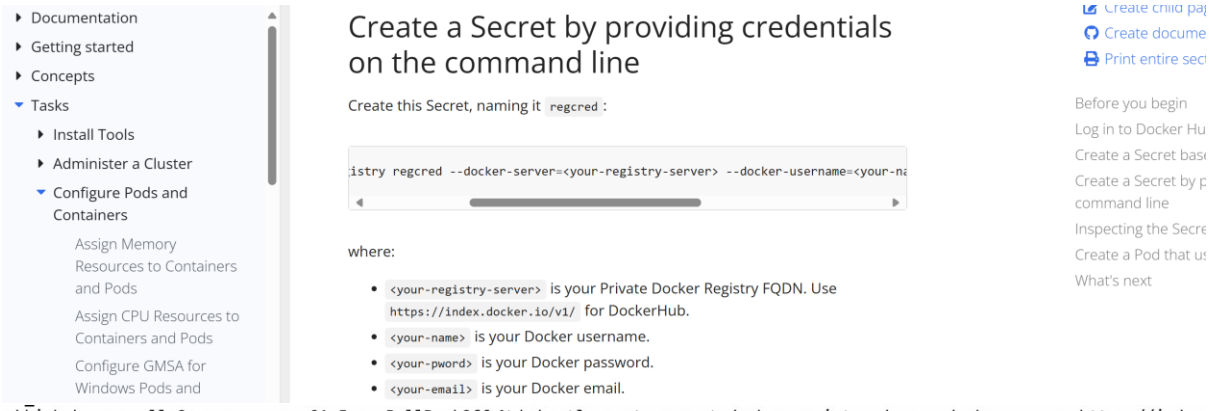
```
kubectl create secret docker-registry  \
  --docker-server=${AWS_ACCOUNT}.dkr.ecr.${AWS_REGION}.amazonaws.com \
  --docker-username=AWS \
  --docker-password=$(aws ecr get-login-password) \
  --namespace=default
```

Ref: https://docs.aws.amazon.com/AmazonECR/latest/public/docker-pull-ecr-image.html

Since, we use docker as registry. We need to use first command and give docker server, username, password and email.

For Docker Server you can refer here what to give. Docker server will be same always: https://kubernetes.io/docs/tasks/configure-pod-container/pull-image-private-

registry/#:~:text=A%20Kubernetes%20cluster%20uses%20the%20Secret%20of%20kubernetes.io/dockerconfigjson%20type%20to




```
abhishekveeramalla@aveerama-mac 01-ImagePullBackOff % kubectl create secret docker-registry demo --docker-server=https://index
.docker.io/v1/ --docker-username=abhishekf5 --docker-password= --docker-email=
```

```
abhishekveeramalla@aveerama-mac 01-ImagePullBackOff % kubectl get secrets
NAME       TYPE                             DATA   AGE
demo       kubernetes.io/dockerconfigjson   1      12s
regcred    kubernetes.io/dockerconfigjson   1      14h
```

Now secret is created. We just need to add this as reference in the deployment file.

```
  spec:
    containers:
    - name: nginx
      image: nginx-image-demo:v1
      ports:
      - containerPort: 80
    imagePullSecrets:
    - name: demo
```

What to provide is also mentioned in the documentation. Same reference link as above.


```
abhishekveeramalla@aveerama-mac 01-ImagePullBackOff % kubectl apply -f nginx-deploy.yaml
deployment.apps/nginx-deployment configured
```

Still we got error.

```
abhishekveeramalla@aveerama-mac 01-ImagePullBackOff % kubectl get pods -w
NAME                                READY   STATUS             RESTARTS   AGE
nginx-deployment-588bb445c7-9hk7s   0/1     ImagePullBackOff   0          7m23s
nginx-deployment-588bb445c7-rf56w   0/1     ImagePullBackOff   0          7m24s
nginx-deployment-588bb445c7-vpphq   0/1     ImagePullBackOff   0          7m23s
nginx-deployment-86796b4976-k6bzt   0/1     ErrImagePull       0          2m12s
nginx-deployment-86796b4976-k6bzt   0/1     ImagePullBackOff   0          2m15s
^C
abhishekveeramalla@aveerama-mac 01-ImagePullBackOff % vim nginx-deploy.yaml
```

Because we haven't specified the dockerhub registry username of our account. It is checking for the image that is not there in dockerhub.

```
  spec:
    containers:
    - name: nginx
      image: abhishekf5/nginx-image-demo:v1
      ports:
      - containerPort: 80
    imagePullSecrets:
    - name: demo
```

```
abhishekveeramalla@aveerama-mac 01-ImagePullBackOff % vim nginx-deploy.yaml
abhishekveeramalla@aveerama-mac 01-ImagePullBackOff % kubectl apply -f nginx-deploy.yaml
deployment.apps/nginx-deployment configured
```

```
abhishekveeramalla@aveerama-mac 01-ImagePullBackOff % kubectl get pods -w
NAME                                READY   STATUS              RESTARTS   AGE
nginx-deployment-554967466c-fcnhq   0/1     ContainerCreating   0          3s
nginx-deployment-588bb445c7-9hk7s   0/1     ImagePullBackOff    0          8m6s
nginx-deployment-588bb445c7-rf56w   0/1     ImagePullBackOff    0          8m7s
nginx-deployment-86796b4976-k6bzt   0/1     ImagePullBackOff    0          2m55s
nginx-deployment-554967466c-fcnhq   1/1     Running             0          7s
nginx-deployment-588bb445c7-9hk7s   0/1     Terminating         0          8m10s
nginx-deployment-588bb445c7-9hk7s   0/1     Terminating         0          8m11s
nginx-deployment-554967466c-zhqf2   0/1     Pending             0          0s
nginx-deployment-554967466c-zhqf2   0/1     Pending             0          0s
nginx-deployment-554967466c-zhqf2   0/1     ContainerCreating   0          0s
nginx-deployment-588bb445c7-9hk7s   0/1     Terminating         0          8m11s
nginx-deployment-588bb445c7-9hk7s   0/1     Terminating         0          8m13s
nginx-deployment-588bb445c7-9hk7s   0/1     Terminating         0          8m13s
nginx-deployment-588bb445c7-9hk7s   0/1     Terminating         0          8m14s
nginx-deployment-588bb445c7-9hk7s   0/1     Terminating         0          8m14s
nginx-deployment-554967466c-zhqf2   1/1     Running             0          5s
nginx-deployment-588bb445c7-rf56w   0/1     Terminating         0          8m17s
```

It is terminating the previous deployment an dcreating new one.

```
abhishekveeramalla@aveerama-mac 01-ImagePullBackOff % kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
nginx-deployment-554967466c-fcnhq   1/1     Running   0          28s
nginx-deployment-554967466c-nwps4   1/1     Running   0          15s
nginx-deployment-554967466c-zhqf2   1/1     Running   0          20s
abhishekveeramalla@aveerama-mac 01-ImagePullBackOff %
```

Day 2: Issue2: CrashLoopBackoff

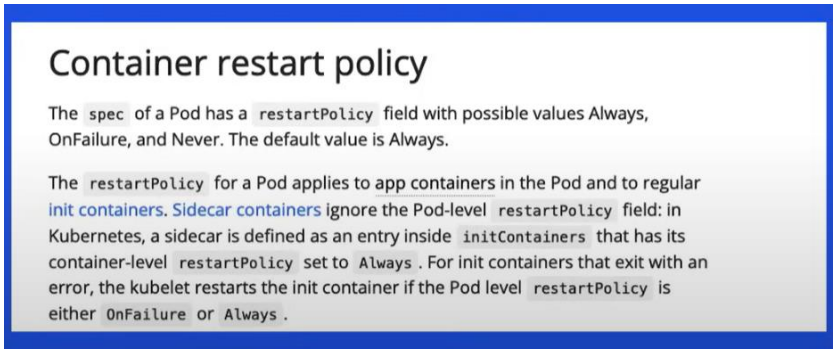
Repo: https://github.com/iam-veeramalla/kubernetes-troubleshooting-zero-to-hero/tree/main/02-CrashLoopBackOff

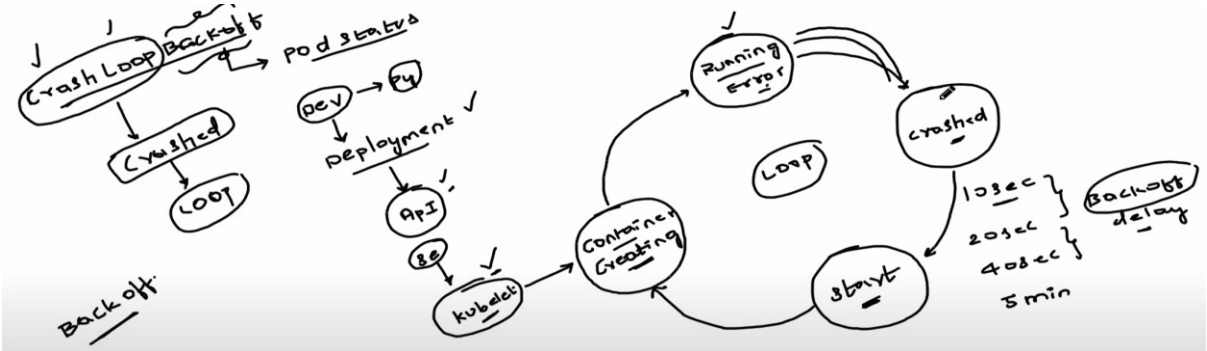
CrashLoopBackOff indicates pod repeatedly crashing. Pod status shown as CrashLoopBackOff means pod is crashing multiple times in a loop.

Developer created a python application, and then wrote a deployment file. Then using kubectl apply we executed this deployment in a cluster. Then the request went to API server and scheduler will take care of identifying the node to do deployment. Scheduler will forward this request through the kubelet on to the particular node. Kubelet will try to run the pod.

Pod 1st goes to Container creating stage. Once created, here there are 2 things can happen. The container and application inside the container running seemlesly. In such cases we will see pod status as running. If there is any issue with the Pod or container then we will see status as error.

Let's say pod is initially running, but after a while it was crashed. K8s will not leave the pod at crashing. It will again try to restart the pod. It is the default nature of the K8s to restart the pod if it got crashed anytime. It will strat creating a container again. Sometimes this happens in a loop. That's why it is called as CrashLoopBackOff.





BackOff means when pod is crashed then K8s by default tries ot restart say in 10 seconds. Then again the new one also gets crashed, it will try to start the pod in 20 seconds this time.

There are multiple reasons a pod can get crash.

Misconfigurations

Misconfigurations can encompass a wide range of issues, from incorrect environment variables to improper setup of service ports or volumes. These misconfigurations can prevent the application from starting correctly, leading to crashes. For example, if an application expects a certain environment variable to connect to a database and that variable is not set or is incorrect, the application might crash as it cannot establish a database connection.

Errors in the Liveness Probes

Liveness probes in Kubernetes are used to check the health of a container. If a liveness probe is incorrectly configured, it might falsely report that the container is unhealthy, causing Kubernetes to kill and restart the container repeatedly. For example, if the liveness probe checks a URL or port that the application does not expose or checks too soon before the application is ready, the container will be repeatedly terminated and restarted.

The Memory Limits Are Too Low

If the memory limits set for a container are too low, the application might exceed this limit, especially under load, leading to the container being killed by Kubernetes. This can happen repeatedly if the workload does not decrease, causing a cycle of crashing and restarting. Kubernetes uses these limits to ensure that containers do not consume all available resources on a node, which can affect other containers.

Wrong Command Line Arguments

Containers might be configured to start with specific command-line arguments. If these arguments are wrong or lead to the application exiting (for example, passing an invalid option to a command), the container will exit immediately. Kubernetes will then attempt to restart it, leading to the CrashLoopBackOff status. An example would be passing a configuration file path that does not exist or is inaccessible.

Bugs & Exceptions

Bugs in the application code, such as unhandled exceptions or segmentation faults, can cause the application to crash. For instance, if the application tries to access a null pointer or fails to catch and handle an exception correctly, it might terminate unexpectedly. Kubernetes, detecting the crash, will restart the container, but if the bug is triggered each time the application runs, this leads to a repetitive crash loop.

Let’s try all the practically.

Reason-1: Wrong command line argument.

Clone the repo: https://github.com/iam-veeramalla/kubernetes-troubleshooting-zero-to-hero.git

```
abhishekveeramalla@aveerama-mac 02-CrashLoopBackOff % ls
01-wrong-cmd-crashloop.yml      03-out-of-memory-crashloop.yml
02-livenessprobe-crashloop.yml  simple-python-app
abhishekveeramalla@aveerama-mac 02-CrashLoopBackOff % vim simple-python-app/app.py
abhishekveeramalla@aveerama-mac 02-CrashLoopBackOff % cd simple-python-app
abhishekveeramalla@aveerama-mac simple-python-app % ls
Dockerfile              app.py                  requirements.txt
README.Docker.md        compose.yaml
abhishekveeramalla@aveerama-mac simple-python-app %
```

```
abhishekveeramalla@aveerama-mac simple-python-app % vim Dockerfile
abhishekveeramalla@aveerama-mac simple-python-app %
```

In CMD we provided **app1** instead of **app**

```
# Expose the port that the application listens on.
EXPOSE 8000

# Run the application.
CMD python3 app1.py
:q!
```

Build an Image using this Dcoker file.

```
abhishekveeramalla@aveerama-mac simple-python-app % docker build -t abhishekf5/crashlooptest:v1 .
[+] Building 5.2s (14/14)                                                    docker:desktop-linux
 => [internal] load build definition from Dockerfile                                        0.1s
 => => transferring dockerfile: 1.66kB                                                      0.0s
 => resolve image config for docker.io/docker/dockerfile:1                                  2.7s
 => [auth] docker/dockerfile:pull token for registry-1.docker.io                            0.0s
 => CACHED docker-image://docker.io/docker/dockerfile:1@sha256:dbbd5e059e8a07ff7ea6233b213b  0.0s
 => [internal] load metadata for docker.io/library/python:3.10.0-slim                       1.7s
 => [auth] library/python:pull token for registry-1.docker.io                               0.0s
 => [internal] load .dockerignore                                                           0.1s
 => => transferring context: 66?B                                                           0.0s
```

Now push it to DockerHub.

```
abhishekveeramalla@aveerama-mac simple-python-app % docker push abhishekf5/crashlooptest:v1
The push refers to repository [docker.io/abhishekf5/crashlooptest]
e1427892e311: Layer already exists
b23a931fb10e: Layer already exists
```

We already had a deployment file created. For this error. In the deployment file we are using the same image we build now.

```
abhishekveeramalla@aveerama-mac 02-CrashLoopBackOff % vim 01-wrong-cmd-crashloop.yml
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: crashloop-example
  labels:
    app: crashlooplearning
spec:
  replicas: 1
  selector:
    matchLabels:
      app: crashlooplearning
  template:
    metadata:
      labels:
        app: crashlooplearning
    spec:
      containers:
      - name: crashlooplearning
        image: abhishekf5/crashlooptest:v1
        ports:
        - containerPort: 8000
```

```
abhishekveeramalla@aveerama-mac 02-CrashLoopBackOff % minikube status
minikube
type: Control Plane
host: Running
kubelet: Running
apiserver: Running
kubeconfig: Configured
```

```
abhishekveeramalla@aveerama-mac 02-CrashLoopBackOff % kubectl apply -f 01-wrong-cmd-crashloop.yml
deployment.apps/crashloop-example created
```

Now, what we learned about CrashLoopBackOff should happen.

```
abhishekveeramalla@aveerama-mac 02-CrashLoopBackOff % kubectl get pods -w
NAME                                READY   STATUS             RESTARTS      AGE
crashloop-example-5d4954b885-92zcs   0/1     CrashLoopBackOff   1 (7s ago)    12s
crashloop-example-5d4954b885-92zcs   0/1     Error              2             20s
```

It will start with container creating but by the time we gave this command it came to Crash loop. And then it went to Error State. Lets wait as it will continuous updates.

```
abhishekveeramalla@aveerama-mac 02-CrashLoopBackOff % kubectl get pods -w
NAME                                READY   STATUS             RESTARTS      AGE
crashloop-example-5d4954b885-92zcs   0/1     CrashLoopBackOff   1 (7s ago)    12s
crashloop-example-5d4954b885-92zcs   0/1     Error              2             20s
crashloop-example-5d4954b885-92zcs   0/1     CrashLoopBackOff   2 (14s ago)   33s
crashloop-example-5d4954b885-92zcs   0/1     Error              3 (30s ago)   49s
^C
```

You can delete it and try again.

```
abhishekveeramalla@aveerama-mac 02-CrashLoopBackOff % kubectl delete deploy crashloop-example
deployment.apps "crashloop-example" deleted
abhishekveeramalla@aveerama-mac 02-CrashLoopBackOff % kubectl apply -f 01-wrong-cmd-crashloop.yml
deployment.apps/crashloop-example created
abhishekveeramalla@aveerama-mac 02-CrashLoopBackOff % kubectl get pods -w
NAME                                READY   STATUS              RESTARTS   AGE
crashloop-example-5d4954b885-7qk4h   0/1     ContainerCreating   0          4s
crashloop-example-5d4954b885-7qk4h   0/1     Error               0          6s
crashloop-example-5d4954b885-7qk4h   0/1     Error               1 (4s ago)   9s
crashloop-example-5d4954b885-7qk4h   0/1     CrashLoopBackOff    1 (4s ago)   12s
crashloop-example-5d4954b885-7qk4h   0/1     Error               2 (20s ago)   28s
^C
```

To resolve it. Change the command in Dockerfile.

```
abhishekveeramalla@aveerama-mac 02-CrashLoopBackOff % cd simple-python-app
abhishekveeramalla@aveerama-mac simple-python-app % ls
Dockerfile              app.py                  requirements.txt
README.Docker.md        compose.yaml
abhishekveeramalla@aveerama-mac simple-python-app % vim Dco
```

```
# Expose the port that the application listens on.
EXPOSE 8000

# Run the application.
CMD python3 app.py
"Dockerfile" 51L, 1617B written
```

Build a new docker image again and pusg it to DockerHub.

```
abhishekveeramalla@aveerama-mac simple-python-app % docker build -t abhishekf5/crashlooptest:v2 .
[+] Building 4.5s (12/12) FINISHED                                                                  docker:des
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 1.66kB
 => resolve image config for docker.io/docker/dockerfile:1
 => CACHED docker-image://docker.io/docker/dockerfile:1@sha256:dbbd5e059e8a07ff7ea6233b213b36aa516b4
```

```
abhishekveeramalla@aveerama-mac simple-python-app % docker push abhishekf5/crashlooptest:v2
The push refers to repository [docker.io/abhishekf5/crashlooptest]
e1427892e311: Preparing
b23a931fb10e: Preparing
e0231e6dc506: Preparing
d8b099c84a8e: Preparing
```

Go to Deployment file and change the Image name:

```
abhishekveeramalla@aveerama-mac 02-CrashLoopBackOff % vim 01-wrong-cmd-crashloop.yml
```

```
  spec:
    containers:
    - name: crashlooplearning
      image: abhishekf5/crashlooptest:v2
      ports:
      - containerPort: 8000
```

```
abhishekveeramalla@aveerama-mac 02-CrashLoopBackOff % kubectl apply -f 01-wrong-cmd-crashloop.yml
deployment.apps/crashloop-example configured
```

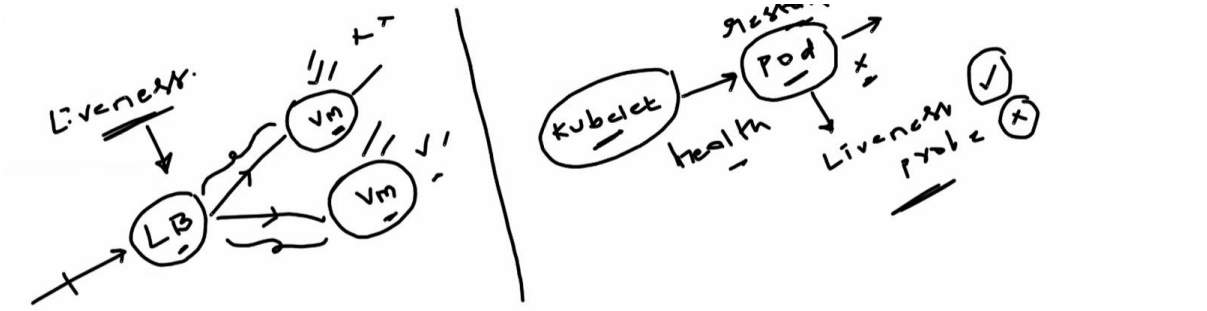
Now, the old one should be terminated and new one should be running.

```
abhishekveeramalla@aveerama-mac 02-CrashLoopBackOff % kubectl get pods -w
NAME                                READY   STATUS              RESTARTS      AGE
crashloop-example-5d4954b885-7qk4h   0/1     CrashLoopBackOff    4 (24s ago)   2m13s
crashloop-example-7fd9d4c666-fc576   0/1     ContainerCreating   0             10s
crashloop-example-7fd9d4c666-fc576   1/1     Running             0             11s
crashloop-example-5d4954b885-7qk4h   0/1     Terminating         4 (26s ago)   2m15s
crashloop-example-5d4954b885-7qk4h   0/1     Terminating         4             2m15s
crashloop-example-5d4954b885-7qk4h   0/1     Terminating         4             2m17s
crashloop-example-5d4954b885-7qk4h   0/1     Terminating         4             2m19s
crashloop-example-5d4954b885-7qk4h   0/1     Terminating         4             2m19s
crashloop-example-5d4954b885-7qk4h   0/1     Terminating         4             2m19s
^C
abhishekveeramalla@aveerama-mac 02-CrashLoopBackOff % kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
crashloop-example-7fd9d4c666-fc576   1/1     Running   0          36s
```

Reason-2: LiveNessProbe

Total there are three Probes in k8s Liveness probe, Readiness Probe and Startup probe. Propular ones are the fisrt two probes.

Liveness probe is similar to load balancer. If there are two VMS connected with Load Balancer and Load balancer continuously check the health of both VMS and passes the requests to VMS based on Algorith we defined (Round Robin or Random). If Load Balancer doesnot have the concept of checking the health of VMs. And one VM is having some issues already then the request going to that VM will fail and the other will pass. So, out of 10 request if 6 went to VM with issues they will fail to process the repsonses.



Similarly, In K8s, kubelet had a way of verifying the pod is healthy or not. We need to pass a livenessprobe configuration in order to start checking the health of pod. If the liveness probe passes then kubelet will understand pod is in healthy staus. If it fails, then kubelet will try to restart pod. Kubelet will check the health of pods.

Configuring a liveness probe for a pod is crucial to prevent unresponsive behavior.

- By setting up a liveness probe, Kubernetes will constantly check if the pod is healthy. If it detects any issues, it will restart the pod.
- Incorporating a liveness probe helps in avoiding situations where the pod hangs or becomes unresponsive to requests, ensuring continuous functionality.

CrashLoopBackOff due to failed liveness probe

- Container restarts after liveness probe failure
- Kubernetes creates a delay before restarting the container

Let’s try this example: Delete the previous deployment.

```
abhishekveeramalla@aveerama-mac 02-CrashLoopBackOff % ls
01-wrong-cmd-crashloop.yml      03-out-of-memory-crashloop.yml
02-livenessprobe-crashloop.yml  simple-python-app
abhishekveeramalla@aveerama-mac 02-CrashLoopBackOff % kubectl delete deploy crashloop-example
deployment.apps "crashloop-example" deleted
abhishekveeramalla@aveerama-mac 02-CrashLoopBackOff %
```

We have a deployment file for this example as well. And Liveness Probe is configured in the file. Docker Image also we were usinng from previous example.

Liveness Probe we defined in the deployment file is incorrect.

```
spec:
  containers:
  - name: crashlooplearning
    image: abhishekf5/crashlooptest:v2
    ports:
    - containerPort: 8000
    livenessProbe:
     exec:
      command:
       - cat
       - /tmp/healthy
     initialDelaySeconds: 0
     periodSeconds: 1
```

Periodseconds tells kubelet how frequently it has to check the health of the pod.

```
abhishekveeramalla@aveerama-mac 02-CrashLoopBackOff % kubectl apply -f 02-livenessprobe-crashloop.yml
deployment.apps/crashloop-example created
abhishekveeramalla@aveerama-mac 02-CrashLoopBackOff % kubectl get pods -w
NAME                                READY   STATUS    RESTARTS   AGE
crashloop-example-7664664bb4-2qjxk   1/1     Running   0          6s
crashloop-example-7664664bb4-2qjxk   1/1     Running   1 (1s ago)   37s
```

It restarted once already.

```
NAME                                READY   STATUS    RESTARTS   AGE
crashloop-example-7664664bb4-2qjxk   1/1     Running   0          6s
crashloop-example-7664664bb4-2qjxk   1/1     Running   1 (1s ago)   37s
crashloop-example-7664664bb4-2qjxk   1/1     Running   2 (2s ago)   73s
```

Restarted secon time. Lets wait for somemore time.

```
abhishekveeramalla@aveerama-mac 02-CrashLoopBackOff % kubectl get pods -w
NAME                                READY   STATUS    RESTARTS   AGE
crashloop-example-7664664bb4-2qjxk   1/1     Running   0          6s
crashloop-example-7664664bb4-2qjxk   1/1     Running   1 (1s ago)   37s
crashloop-example-7664664bb4-2qjxk   1/1     Running   2 (2s ago)   73s
crashloop-example-7664664bb4-2qjxk   1/1     Running   3 (2s ago)   108s
crashloop-example-7664664bb4-2qjxk   1/1     Running   4 (3s ago)   2m24s
crashloop-example-7664664bb4-2qjxk   0/1     CrashLoopBackOff   4 (0s ago)   2m57s
crashloop-example-7664664bb4-2qjxk   1/1     Running            5 (42s ago)   3m39s
```

It went into CrashLoopBack off.

Actual Liveness probe will look like this.

Path we define of our app. Kubelet tries to hit this path and check the response (500/404/200) it is getting. Depending on the repsonse kubelet will understand whether to restart the pod.

```
livenessProbe:
  httpGet:
    path: /healthz
    port: 8080
    httpHeaders:
    - name: Custom-Header
      value: Awesome
```

Reason-3: OOMKilled - Memory Limits Are Too Low

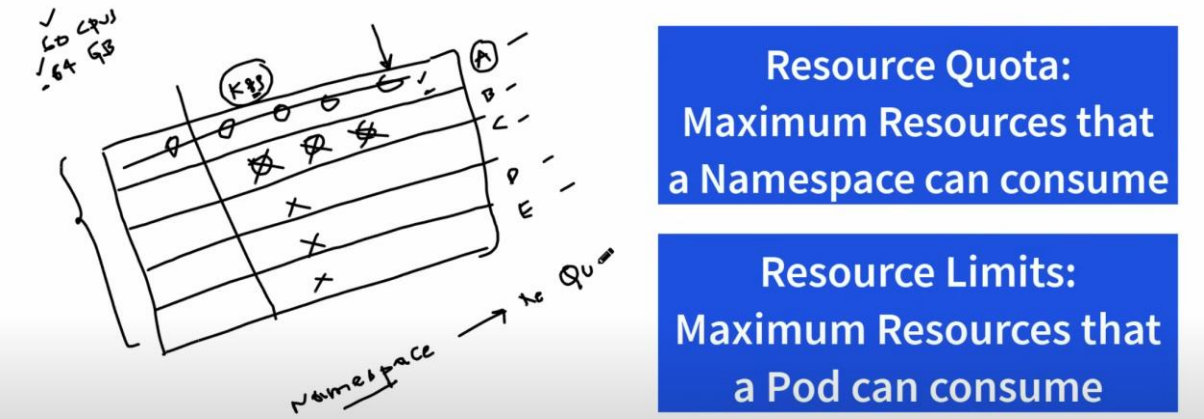
We have a deployment file for this example as well. And Liveness Probe is removed and added new section called resources to the deployment file.

```
spec:
  containers:
  - name: crashlooplearning
    image: abhishekf5/crashlooptest:v2
    ports:
    - containerPort: 8000
    resources:
     limits:
      cpu: "25m"
      memory: "25Mi"
```

```
abhishekveeramalla@aveerama-mac 02-CrashLoopBackOff % kubectl delete deploy crashloop-example
deployment.apps "crashloop-example" deleted
```

We have a single k8s cluster used by multiple teams. It is of three nodes and they are using 60 CPUs and total 64 GB RAM altogether. There are 5 namespaces. Namespace A running 5 pods. What if 5 pods in Namespace A are using entire 60 CPUs and 64 GB RAM. Other namespaces will also have some pods or they might schedule some new pods and they might not run in the cluster. Because Pods in Namespace A eating away all the resources of the cluster.

As a DevOps engineer ti is our responsibility to disstribute the resources of the cluster between the namespaces in the cluster. To do that, we will configure Resource Quota at a Namespace level. Resource Limits at Pod level.



For pod we defined the limits but we defined very very less. If we give 1000 then it is considered as 1 cpu. 25 which is 1/40 th of the CPU and memory. This will not be sufficient. When the pod is starting it will crash due to less resource allocation. Let's see it practically.

```
- containerPort: 8000
resources:
  limits:
    cpu: "25m"
    memory: "25Mi"
```

Don't Increase the CPU or Memory. It is not the solution and a very bad practice. Devs should identify the root cause on why the pod is taking more resources than expected.

```
abhishekveeramalla@aveerama-mac 02-CrashLoopBackOff % kubectl apply -f 03-out-of-memory-crashloop.yml
deployment.apps/crashloop-example created
```

We got OOMKilled error. Out Of Memory Killed error. It happen when the allocated resources for the pod are not enuf orr the pod is taking morethan the resources that it should actually take.

Command: kubectl get pods -w

```
NAME                                 READY   STATUS              RESTARTS      AGE
crashloop-example-7646cbb8cf-vb98g   0/1     ContainerCreating   0             4s
crashloop-example-7646cbb8cf-vb98g   1/1     Running             0             14s



crashloop-example-7646cbb8cf-vb98g   0/1     OOMKilled           0             100s
crashloop-example-7646cbb8cf-vb98g   1/1     Running             1 (4s ago)    104s
crashloop-example-7646cbb8cf-vb98g   0/1     OOMKilled           1 (89s ago)   3m9s
crashloop-example-7646cbb8cf-vb98g   0/1     CrashLoopBackOff    1 (16s ago)   3m24s
crashloop-example-7646cbb8cf-vb98g   1/1     Running             2 (20s ago)   3m28s
```

It is happened as we discussed. First container created then it started running. But it got crashed due to OOMKilled and then restart again in some time. Again crashed and it went into **CrashLoopBackOff.**

Day 3: Issue - 3: Pods Not Schedulable

Repo: https://github.com/iam-veeramalla/kubernetes-troubleshooting-zero-to-hero/tree/main/03-pods-not-schedulable

In Kubernetes, the scheduler is responsible for assigning pods to nodes in the cluster based on various criteria. Sometimes, you might encounter situations where pods are not being scheduled as expected. This can happen due to factors such as node constraints, pod requirements, or cluster configurations.

For this Concept we need Multi Node Cluster. To create multi node cluster it is prefer to use Kind/k3d. Kind is nothing but **K**ubernetes **in**side **D**ocker. The whole cluster will be inside a container. Since the cluster are spinning up inside the conatiners, they are very light weight. We can spin up as many clusters we want.

Kind: https://kind.sigs.k8s.io/docs/user/quick-start/#:~:text=Quick%20Start.%20This%20guide%20covers%20getting%20started%20with%20the%20kind

Install Kind and create a 4 node cluster.

```
abhishekveeramalla@aveerama-mac kind % kubectl get nodes
NAME                  STATUS   ROLES           AGE     VERSION
multi-control-plane   Ready    control-plane   3h32m   v1.29.2
multi-worker          Ready    <none>          3h32m   v1.29.2
multi-worker2         Ready    <none>          3h32m   v1.29.2
multi-worker3         Ready    <none>          3h32m   v1.29.2
abhishekveeramalla@aveerama-mac kind % KIND
```

```
abhishekveeramalla@aveerama-mac kind % kind create cluster --name=abhi
Creating cluster "abhi" ...
 ✓ Ensuring node image (kindest/node:v1.29.2) 🖼
 ✓ Preparing nodes 📦
 ✓ Writing configuration 📜
 ✓ Starting control-plane 🕹️
 ✓ Installing CNI 🔌
 ✓ Installing StorageClass 💾
Set kubectl context to "kind-abhi"
You can now use your cluster with:

kubectl cluster-info --context kind-abhi

Have a nice day! 👋
abhishekveeramalla@aveerama-mac kind % kind get clusters
abhi
multi
abhishekveeramalla@aveerama-mac kind %
```

We created cluster called abhi with one node. To create cluster with multiple nodes.

```
abhishekveeramalla@aveerama-mac kind % vim multi-node-cluster.yaml
```

```
kind: Cluster
apiVersion: kind.x-k8s.io/v1alpha4
nodes:
- role: control-plane
- role: worker
- role: worker
- role: worker
- role: control-plane
- role: worker
```

You can add role for worker node or master node based on requirement. Refer below link documentation of how to write YAML file for multi node cluster.

Ref: https://kind.sigs.k8s.io/docs/user/quick-start/#creating-a-cluster

Multi-node clusters

In particular, many users may be interested in multi-node clusters. A simple configuration for this can be achieved with the following config file contents:

```
# three node (two workers) cluster config
kind: Cluster
apiVersion: kind.x-k8s.io/v1alpha4
nodes:
- role: control-plane
- role: worker
- role: worker
```

```
abhishekveeramalla@aveerama-mac kind % kind create cluster --name abhi-multi --config=multi-node-cluster.yaml

Creating cluster "abhi-multi" ...
 ✓ Ensuring node image (kindest/node:v1.29.2)
 ✓ Preparing nodes
 ✓ Writing configuration
 ✓ Starting control-plane
abhishekveeramalla@aveerama-mac kind % kubectl get nodes
NAME                       STATUS   ROLES           AGE   VERSION
abhi-multi-control-plane   Ready    control-plane   35s   v1.29.2
abhi-multi-worker          Ready    <none>          15s   v1.29.2
abhi-multi-worker2         Ready    <none>          15s   v1.29.2
abhi-multi-worker3         Ready    <none>          11s   v1.29.2
abhishekveeramalla@aveerama-mac kind %
```

One Master and one worker node is not created because we haven't save the changes we did to the YAML file that we used to create multi node cluster.

```
abhishekveeramalla@aveerama-mac kind % kind get clusters
abhi
abhi-multi
multi
abhishekveeramalla@aveerama-mac kind % kind delete cluster abhi
ERROR: unknown command "abhi" for "kind delete cluster"
abhishekveeramalla@aveerama-mac kind % kind delete cluster --name=abhi
Deleting cluster "abhi" ...
Deleted nodes: ["abhi-control-plane"]
abhishekveeramalla@aveerama-mac kind % kind delete cluster --name=abhi-multi
```

If you give this command now:

```
abhishekveeramalla@aveerama-mac kind % kubectl get nodes
E0501 21:07:42.573215   66990 memcache.go:265] couldn't get current server API group list:
st:8080/api?timeout=32s": dial tcp 127.0.0.1:8080: connect: connection refused
E0501 21:07:42.574876   66990 memcache.go:265] couldn't get current server API group list:
st:8080/api?timeout=32s": dial tcp 127.0.0.1:8080: connect: connection refused
E0501 21:07:42.575744   66990 memcache.go:265] couldn't get current server API group list:
st:8080/api?timeout=32s": dial tcp 127.0.0.1:8080: connect: connection refused
E0501 21:07:42.576483   66990 memcache.go:265] couldn't get current server API group list:
st:8080/api?timeout=32s": dial tcp 127.0.0.1:8080: connect: connection refused
```

Kubectl is still pointing to the previous cluster which we deleted.

List the available cluster in PC:

```
abhishekveeramalla@aveerama-mac kind % kubectl config view | grep kind
  name: kind-multi
    cluster: kind-multi
    user: kind-multi
  name: kind-multi
kind: Config
- name: kind-multi
```

grep will only show lcuster created using kind. If you remove grep it will show all the clusters. We have only one cluster now called **multi.** To point the kubectl to that cluster use below command.

```
abhishekveeramalla@aveerama-mac kind % kubectl config use-context kind-multi
Switched to context "kind-multi".
```

Now it pointing to the cluster that is available.

```
abhishekveeramalla@aveerama-mac kind % kubectl get nodes
NAME                  STATUS   ROLES           AGE     VERSION
multi-control-plane   Ready    control-plane   3h41m   v1.29.2
multi-worker          Ready    <none>          3h41m   v1.29.2
multi-worker2         Ready    <none>          3h41m   v1.29.2
multi-worker3         Ready    <none>          3h41m   v1.29.2
abhishekveeramalla@aveerama-mac kind %
```

```
abhishekveeramalla@aveerama-mac kind % ls
deployment-node-selector.yaml   deployment-required-sched.yaml
deployment-preferred-sched.yaml multi-node-cluster.yaml
abhishekveeramalla@aveerama-mac kind %
```

```
abhishekveeramalla@aveerama-mac kind % vim deployment-node-selector.yaml
```

```
  spec:
    nodeSelector:
      node-name: arm-worker
    containers:
    - name: nginx
      image: nginx:1.14.2
      ports:
      - containerPort: 80
```

Lets give node-name as defined in the image.

Ref: https://github.com/iam-veeramalla/kubernetes-troubleshooting-zero-to-hero/blob/main/03-pods-not-schedulable/02-node-selector.yaml

```
abhishekveeramalla@aveerama-mac kind % kubectl apply -f deployment-node-selector.yaml
deployment.apps/nginx-deployment created
```

Expectation is pod should eb running. It's been 36 seconds still the pods are in pending status only.

```
abhishekveeramalla@aveerama-mac kind % kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
nginx-deployment-867798947d-6848f   0/1     Pending   0          36s
nginx-deployment-867798947d-f56fh   0/1     Pending   0          36s
nginx-deployment-867798947d-pgwhw   0/1     Pending   0          36s
abhishekveeramalla@aveerama-mac kind %
```

Lets check with one pod.

```
abhishekveeramalla@aveerama-mac kind % kubectl describe pod nginx-deployment-867798947d-6848f
```

It shows and failed message.

```
Events:
  Type     Reason            Age   From               Message
  ----     ------            ----  ----               -------
  Warning  FailedScheduling  54s   default-scheduler  0/4 nodes are available: 1 node(s) had untolerated taint
 {node-role.kubernetes.io/control-plane: }, 3 node(s) didn't match Pod's node affinity/selector. preemption: 0
/4 nodes are available: 4 Preemption is not helpful for scheduling.
abhishekveeramalla@aveerama-mac kind %
```

It is saying there are 4 nodes but they are not helpful for scheduling pods to nodes.

Why? We gave node-selector as arm-worker which is not even existing.

```
abhishekveeramalla@aveerama-mac kind % kubectl edit pod nginx-deployment-867798947d-6848f
```

what is the node it is trying to sechedule for this pod. This is the one we gave in deployment YAML.

```
 enableServiceLinks: true
 nodeSelector:
   node-name: arm-worker
 preemptionPolicy: PreemptLowerPriority
```

Concept -1: Node-Selector: This helps the Kube-Scheduler to schedule the pods on a particular nodes. We can mention which node label to use in the dpeloyment or pod YAML file. Then scheduler will deploy the pod if there is a node in the cluster with that label you mentioned.

How to know or add the lables of the nodes. Select any one node and try to edit it.

```
abhishekveeramalla@aveerama-mac kind % kubectl edit node multi-worker
node/multi-worker edited
```

Under the labels add **node-name:**

```
creationTimestamp: "2024-05-01T11:58:04Z"
labels:
  beta.kubernetes.io/arch: amd64
  beta.kubernetes.io/os: linux
  foo: bar
  kubernetes.io/arch: amd64
  kubernetes.io/hostname: multi-worker
  kubernetes.io/os: linux
  node-name: arm-worker
name: multi-worker
```

Give the label name for this node. You have use the same label in the deployment or pod yaml file.

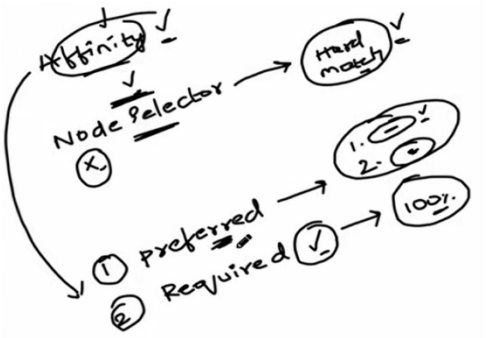
We have one node with lable as arm-worker. So, the pods will be scheduled in that node only.

```
abhishekveeramalla@aveerama-mac kind % kubectl get pods -w
NAME                                READY   STATUS    RESTARTS   AGE
nginx-deployment-867798947d-6848f   1/1     Running   0          9m47s
nginx-deployment-867798947d-f56fh   1/1     Running   0          9m47s
nginx-deployment-867798947d-pgwhw   1/1     Running   0          9m47s
^C
```

This type of scenario can be faced when we do enterprise level. They may have requirement like nodes with Windows/ARM/AMD processors. We have to add pods to specific node then we need to use Node-Selector by adding a lable to it within the deployment or pod YAMl file under spec section. Then add same label to the node as well. Or use same lable that is mentioned for nodes in the deployment or pod YAML files.

Concept – 2: Node Affinity:

Node Affinity is a more expressive way to specify rules about the placement of pods relative to nodes' labels. It allows you to specify rules that apply only if certain conditions are met. Usage: Define nodeAffinity rules in the pod's YAML definition, specifying required and preferred node selectors.



With Node selector we are restricting the scheduler to schedule pod only if it is matching the label of the node. Othewise it will not schedule any pods. It is doing a Hard Match. Where as **Affinity** provides more flexibilty to the scheduler with two things: Required YAML and Preferred YAML. Required will works same as Node Selector only. If the match is 100% then Schedule pods on specified nodes. If it is not then can schedule the pods in any node that are mentioned in the preferred YAML.

We have two YAMl files already created for these.

03-node-affinity-preferred.yaml

04-node-affinity-required.yaml

Preferred:

```
spec:
  affinity:
   nodeAffinity:
    preferredDuringSchedulingIgnoredDuringExecution:
    - weight: 1
      preference:
       matchExpressions:
       - key: node-arch
         operator: In
         values:
         - windows
```

Here we mentioned wrong conditions under match expressions. We don't have any node like that. So, that it will fail in this Node Affinity case as well for our trouble shooting. Later we will update it correctly.

```
abhishekveeramalla@aveerama-mac kind % kubectl apply -f deployment-preferred-sched.yaml
```

Pod got scheduled on different node:

```
abhishekveeramalla@aveerama-mac kind % kubectl get pods -o wide
NAME                                READY   STATUS    RESTARTS   AGE   IP           NODE            NOMINATED
NODE   READINESS GATES
nginx-deployment-797b57bbdb-mlcs5   1/1     Running   0          10s   10.244.3.9   multi-worker3   <none>
       <none>
```

Because if you see the YAML file Node Affinity is mentioned as "PreferredDuringScheduling……."

```
abhishekveeramalla@aveerama-mac kind % kubectl delete deploy nginx-deployment
deployment.apps "nginx-deployment" deleted
abhishekveeramalla@aveerama-mac kind % vim deployment-preferred-sched.yaml
```

Edit the details and give node label that is present for any of your node. We have a node name label with arm worker. So, lets give that.

```
spec:
  affinity:
   nodeAffinity:
    preferredDuringSchedulingIgnoredDuringExecution:
    - weight: 1
      preference:
       matchExpressions:
       - key: node-name
         operator: In
         values:
         - arm-worker
```

This time it will schedule pods inside this node.

```
abhishekveeramalla@aveerama-mac kind % kubectl apply -f deployment-preferred-sched.yaml
deployment.apps/nginx-deployment created
abhishekveeramalla@aveerama-mac kind % vim deployment-preferred-sched.yaml
abhishekveeramalla@aveerama-mac kind % kubectl get pods -o wide
NAME                                READY   STATUS    RESTARTS   AGE   IP            NODE           NOMINATED
NODE   READINESS GATES
nginx-deployment-76d94ccc75-5vrbq   1/1     Running   0          5s    10.244.2.18   multi-worker   <none>
       <none>
abhishekveeramalla@aveerama-mac kind % kubectl edit node multi-worker
```

The **pod** is scheduled in **multi-worker node.** Lets see what is the label we gave to that node. We added a label called **node-name** for this node.

```
labels:
  beta.kubernetes.io/arch: amd64
  beta.kubernetes.io/os: linux
  foo: bar
  kubernetes.io/arch: amd64
  kubernetes.io/hostname: multi-worker
  kubernetes.io/os: linux
  node-name: arm-worker
name: multi-worker
```

You can try for **Required** as well:

```
spec:
  affinity:
   nodeAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
     nodeSelectorTerms:
     - matchExpressions:
       - key: foo
         operator: In
         values:
         - bar1
```

Delete the Deployment. Create a deployment using YAML created for Affinity-Required. This it will not schedule any pods as labels mentioned in above YAML are not there for any node. The pods will not be scheduled. They will be in pending state only.

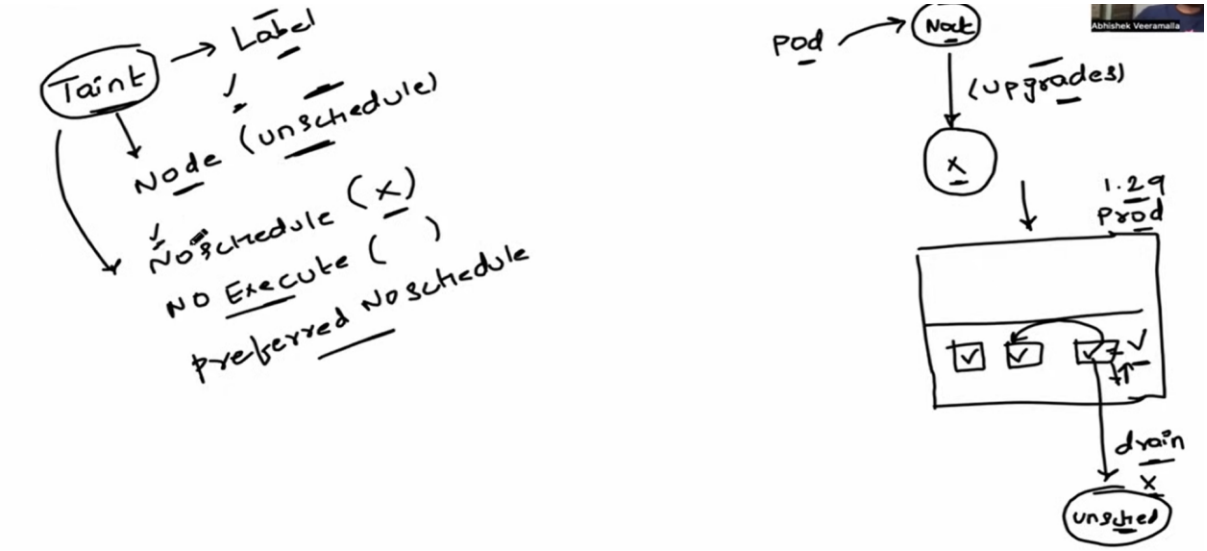
Change the label same as previous example. Apply the deployment again. This time it should schedule pods only in that node.

Concept -3: Taint:

If we didn't want a node to be scheduled for pods. Then we need to use **Taint.**

Why? If we want to **upgrade** the **node version** of any node out of 3 three nodes. Then we will drain the node first (We will move all the pods that are running in this node to a different node). Tehn we will unschedule the node. This node will be ideal. We can bring this node down and we can upgrade it and then remove the unschedulable status. New version of node is up and running now.

This is one way of doing upgrades to the K8s clusters reosurces. This is best usecase of Taint concept. Here also we will use lables only in order to let kube scheduler to understand this particular node to be unscheduled.



Ref: https://kubernetes.io/docs/concepts/scheduling-eviction/taint-and-toleration/#:~:text=Taints%20and%20tolerations%20work%20together%20to%20ensure%20that%20pods%20are

`NoExecute`

This affects pods that are already running on the node as follows:

- Pods that do not tolerate the taint are evicted immediately
- Pods that tolerate the taint without specifying `tolerationSeconds` in their toleration specification remain bound forever
- Pods that tolerate the taint with a specified `tolerationSeconds` remain bound for the specified amount of time. After that time elapses, the node lifecycle controller evicts the Pods from the node.

`NoSchedule`

No new Pods will be scheduled on the tainted node unless they have a matching toleration. Pods currently running on the node are **not** evicted.

`PreferNoSchedule`

`PreferNoSchedule` is a "preference" or "soft" version of `NoSchedule`. The control plane will *try* to avoid placing a Pod that does not tolerate the taint on the node, but it is not guaranteed.

No Schedule's best use use case is upgrading node versions. Prefer No Schedule best use case is when the node is not performing well. It might have CPU issues, memory issues, etc. No Execute is little bit dangerous. All pods on a particular node will stop working with immediate effect.

```
abhishekveeramalla@aveerama-mac kind % kubectl get nodes
NAME                  STATUS   ROLES           AGE     VERSION
multi-control-plane   Ready    control-plane   4h38m   v1.29.2
multi-worker          Ready    <none>          4h38m   v1.29.2
multi-worker2         Ready    <none>          4h38m   v1.29.2
multi-worker3         Ready    <none>          4h38m   v1.29.2
abhishekveeramalla@aveerama-mac kind % kubectl taint nodes multi-worker key1=value1:NoSchedule
node/multi-worker tainted
abhishekveeramalla@aveerama-mac kind %
```

We made a particular node as tainted. Let's see master node once.

```
abhishekveeramalla@aveerama-mac kind % kubectl edit node multi-control-plane
  providerID: kind://docker/multi/multi-control-plane
  taints:
  - effect: NoSchedule
    key: node-role.kubernetes.io/control-plane
status:
```

It already had a taint called NoSchedule. Because we usually not schedule any pods in the control plane. That's why they will always be scheduled on worker nodes.

Lets me taint for another node as well.

```
abhishekveeramalla@aveerama-mac kind % kubectl taint nodes multi-worker2 key1=value1:NoSchedule
node/multi-worker2 tainted
abhishekveeramalla@aveerama-mac kind %
```

We just left with one worker node then for scheduling the pods. Delete if there are any prebious deployments.

```
abhishekveeramalla@aveerama-mac kind % kubectl delete deploy nginx-deployment
deployment.apps "nginx-deployment" deleted
abhishekveeramalla@aveerama-mac kind %
```

What ever we deploy should go to the remaining pod. To do a dpeloyment edit node-selector YAML file and remove the node-selector lines to make it a plain nginx deployment.

```
abhishekveeramalla@aveerama-mac kind % vim deployment-node-selector.yaml
      labels:
        app: nginx
    spec:
      nodeSelector:
        node-name: arm-worker
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

```
  metadata:
    labels:
      app: nginx
  spec:
    containers:
    - name: nginx
      image: nginx:1.14.2
      ports:
      - containerPort: 80
```

```
abhishekveeramalla@aveerama-mac kind % kubectl apply -f deployment-node-selector.yaml
deployment.apps/nginx-deployment created
abhishekveeramalla@aveerama-mac kind %
```

```
abhishekveeramalla@aveerama-mac kind % kubectl get pods -o wide
NAME                                READY   STATUS    RESTARTS   AGE   IP            NODE            NOMINATE
 NODE    READINESS GATES
nginx-deployment-86dcfdf4c6-7gtcv   1/1     Running   0          15s   10.244.3.12   multi-worker3   <none>
        <none>
nginx-deployment-86dcfdf4c6-j2l9f   1/1     Running   0          15s   10.244.3.11   multi-worker3   <none>
        <none>
nginx-deployment-86dcfdf4c6-s262t   1/1     Running   0          15s   10.244.3.10   multi-worker3   <none>
        <none>
abhishekveeramalla@aveerama-mac kind %
```

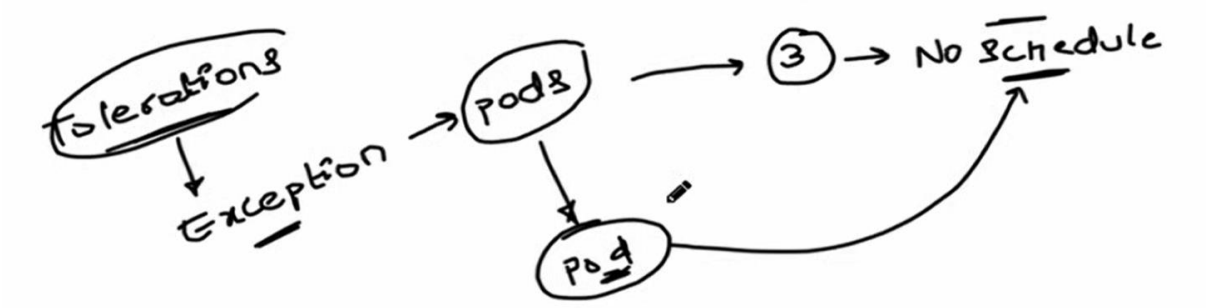
Only deploying on remaining node.

Concept – 4: Tolerations

Tolerations are applied to pods and allow them to schedule onto nodes with matching taints. They override the effect of taints.

Usage: Include tolerations field in the pod's YAML definition to specify which taints the pod tolerates.

Toleration is basically an exception given to certain pods.



There are three nodes that are tainted with any type. Then you have a specific pod which is very critical one like performance related or production related. And to that pod we will add a toleration or exception so that we can run that on any type of tainted node.

While tainting the node we added a key value pair.

```
abhishekveeramalla@aveerama-mac kind % kubectl taint nodes multi-worker2 key1=value1:NoSchedul
bck-i-search: taint_
```

If we add a toleration to the pod which is matching these key value pair of the taint type. We will add this part to the deployment file.

```
tolerations:
- key: "key1"
  operator: "Equal"
  value: "value1"
  effect: "NoSchedule"
```

Ref: https://kubernetes.io/docs/concepts/scheduling-eviction/taint-and-toleration/#:~:text=Taints%20and%20tolerations%20work%20together%20to%20ensure%20that%20pods%20are

```
abhishekveeramalla@aveerama-mac kind % vim deployment-node-selector.yaml
  spec:
    containers:
    - name: nginx
      image: nginx:1.14.2
      ports:
      - containerPort: 80
    tolerations:
     - key: "key1"
       operator: "Equal"
       value: "value1"
       effect: "NoSchedule"
```

We haven't yet applied the deployment yet.

```
abhishekveeramalla@aveerama-mac kind % kubectl get pods -o wide
NAME                                READY   STATUS    RESTARTS   AGE   IP            NODE            NOMINATED
 NODE    READINESS GATES
nginx-deployment-86dcfdf4c6-7gtcv   1/1     Running   0          18m   10.244.3.12   multi-worker3   <none>
        <none>
nginx-deployment-86dcfdf4c6-j2l9f   1/1     Running   0          18m   10.244.3.11   multi-worker3   <none>
        <none>
nginx-deployment-86dcfdf4c6-s262t   1/1     Running   0          18m   10.244.3.10   multi-worker3   <none>
        <none>
```

All the four nodes are tained now.

```
abhishekveeramalla@aveerama-mac kind % kubectl taint nodes multi-worker3 key1=value1:NoSchedule
node/multi-worker3 tainted
abhishekveeramalla@aveerama-mac kind %
```

```
abhishekveeramalla@aveerama-mac kind % kubectl delete deploy nginx-deployment
deployment.apps "nginx-deployment" deleted
abhishekveeramalla@aveerama-mac kind %
abhishekveeramalla@aveerama-mac kind % kubectl apply -f deployment-node-selector.yaml
deployment.apps/nginx-deployment created
abhishekveeramalla@aveerama-mac kind %
```

```
abhishekveeramalla@aveerama-mac kind % kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
nginx-deployment-6c686769-b9gcx   1/1     Running   0          14s
nginx-deployment-6c686769-btzvx   1/1     Running   0          14s
nginx-deployment-6c686769-jgnmm   1/1     Running   0          14s
abhishekveeramalla@aveerama-mac kind % kubectl get pods -o wide
NAME                                READY   STATUS    RESTARTS   AGE   IP             NODE             NOMINATED
ODE   READINESS GATES
nginx-deployment-6c686769-b9gcx   1/1     Running   0          18s   10.244.3.13    multi-worker3    <none>
      <none>
nginx-deployment-6c686769-btzvx   1/1     Running   0          18s   10.244.2.19    multi-worker     <none>
      <none>
nginx-deployment-6c686769-jgnmm   1/1     Running   0          18s   10.244.1.3     multi-worker2    <none>
      <none>
abhishekveeramalla@aveerama-mac kind %
```

Since replicas mentioned were three. Each replica is scheduled on different nodes. Because we gave key value pair sa,e for all the worker nodes when we made them tainted.

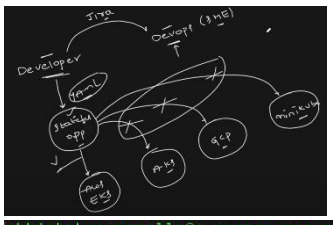
```
  metadata:
    name: nginx-deployment
    labels:
      app: nginx
  spec:
    replicas: 3
    selector:
      matchLabels:
        app: nginx
    template:
```

```
abhishekveeramalla@aveerama-mac kind % kubectl edit pod nginx-deployment-6c686769-b9gcx
```

```
 terminationGracePeriodSeconds: 30
 tolerations:
 - effect: NoSchedule
   key: key1
   operator: Equal
   value: value1
 - effect: NoExecute
```

Day – 4: Issue – 4: Stateful Set

Repo: https://github.com/iam-veeramalla/kubernetes-troubleshooting-zero-to-hero/tree/main/04-statefulset-pv



Developer created a JIRA Ticket to DevOps Engineer saying that stateful application like database application is working fine when they are deploying it into EKS. But it is not working when they are deploying same app into AKS, GCP or miniKube clusters.

Create a Minikube Cluster.

```
abhishekveeramalla@aveerama-mac statefulset % minikube status
minikube
type: Control Plane
host: Running
kubelet: Running
apiserver: Running
kubeconfig: Configured
```

```
abhishekveeramalla@aveerama-mac statefulset % ls
sample-statefulset.yaml
abhishekveeramalla@aveerama-mac statefulset % vim sample-statefulset.yaml
```

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: web
spec:
  selector:
    matchLabels:
      app: nginx
```

```
        name: web
      volumeMounts:
      - name: www
        mountPath: /usr/share/nginx/html
volumeClaimTemplates:
- metadata:
    name: www
  spec:
    accessModes: [ "ReadWriteOnce" ]
    storageClassName: ebs
```

Persistent volume is also added.

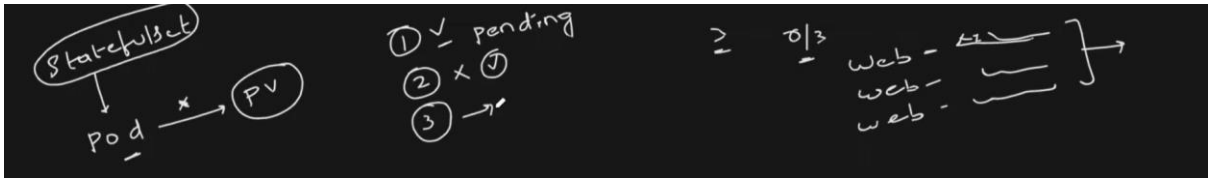
```
abhishekveeramalla@aveerama-mac statefulset % kubectl apply -f sample-statefulset.yaml
statefulset.apps/web configured
abhishekveeramalla@aveerama-mac statefulset % kubectl get statefulset
NAME   READY   AGE
web    0/3     11m
abhishekveeramalla@aveerama-mac statefulset % kubectl get pods
NAME    READY   STATUS    RESTARTS   AGE
web-0   0/1     Pending   0          11m
abhishekveeramalla@aveerama-mac statefulset %
```

there iss only replica created and stateful sets are also not ready.

```
abhishekveeramalla@aveerama-mac statefulset % kubectl describe pod web-0
                          node.kubernetes.io/unreachable:NoExecute op=Exists for 300s
Events:
  Type     Reason            Age                From               Message
  ----     ------            ----               ----               -------
  Warning  FailedScheduling  2m6s (x3 over 12m)  default-scheduler  0/1 nodes are available: pod has unbound immediate PersistentVolumeClaims. preemption: 0/1 nodes are available: 1 Preemption is not helpful for scheduling..
abhishekveeramalla@aveerama-mac statefulset %
```

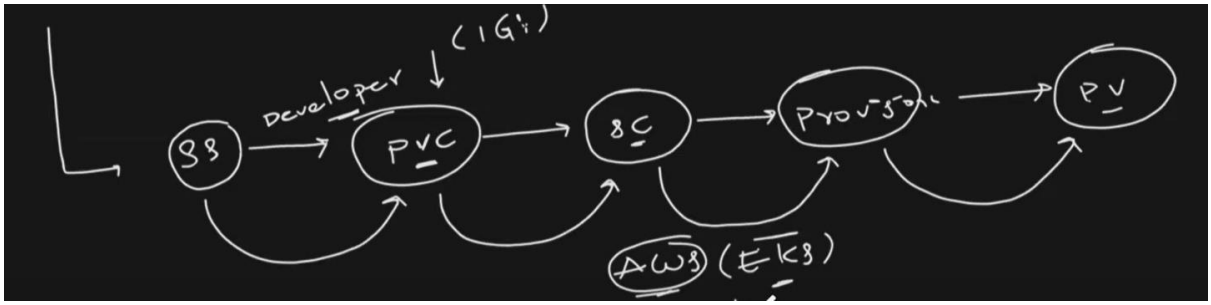
This is the issue. When we are trying to deploy a pod through a stateful set. Pod we are trying to deploy is requesting a persistent volume. Vut it is not able to find a persistent volume.

In Statefulset next pod will be created if the previous is successful only. Unlike normal deployments, if the replicas count is 3, whether there is an issue or not, it will try to deploy all three pods they might no be running but it will create.

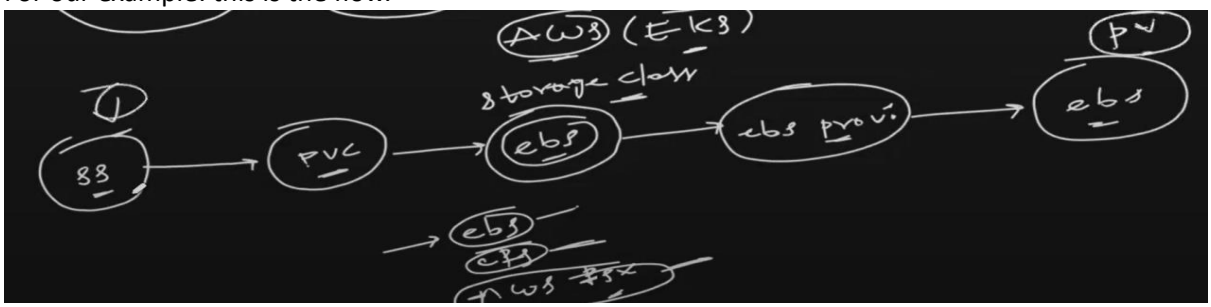


In Stateful set, if there is an issue and first pod itself is still in pending then it will not create.

In Stateful set developer will mention about the Persistent volume Claim (PVC), it will have a storage class and this storage class will talk to the provisioner and provisioner will create a persistent volume.



For our example: this is the flow.



The final PV created will be used by the stateful set. This stateful file is workign in AWS EKS because it had storage classes like ebs, efs, aws fsx.

But when we try in Minikune the storage classes are:

```
abhishekveeramalla@aveerama-mac statefulset % kubectl get storageclass
NAME                 PROVISIONER                RECLAIMPOLICY   VOLUMEBINDINGMODE   ALLOWVOLUMEEXPANSION   AGE
standard (default)   k8s.io/minikube-hostpath   Delete          Immediate           false                  89m
```

We can also see there is a provisioner for storage fof our minikube cluster.

```
abhishekveeramalla@aveerama-mac statefulset % kubectl get pods -A
NAMESPACE     NAME                               READY   STATUS    RESTARTS      AGE
default       web-0                              0/1     Pending   0             21m
kube-system   coredns-5dd5756b68-r4xj5           1/1     Running   0             90m
kube-system   etcd-minikube                      1/1     Running   0             90m
kube-system   kube-apiserver-minikube            1/1     Running   0             90m
kube-system   kube-controller-manager-minikube   1/1     Running   0             90m
kube-system   kube-proxy-hrsvp                   1/1     Running   0             90m
kube-system   kube-scheduler-minikube            1/1     Running   0             90m
kube-system   storage-provisioner                1/1     Running   1 (90m ago)   90m
abhishekveeramalla@aveerama-mac statefulset %
```

So, if we use the correct storage class of Minikube cluster then we will be able to provision PV for the Pod using Stateful Set.


```
abhishekveeramalla@aveerama-mac statefulset % kubectl delete -f sample-statefulset.yaml
statefulset.apps "web" deleted
abhishekveeramalla@aveerama-mac statefulset % vim
```

Also, delete the persistent volume claim.

```
abhishekveeramalla@aveerama-mac statefulset % kubectl get pvc
NAME        STATUS    VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS   AGE
www-web-0   Pending                                                                         ebs            24m
www-web-1   Bound     pvc-b2b3ee87-27c3-403b-87ab-31a32f4c2d73   1Gi        RWO            standard       69m
abhishekveeramalla@aveerama-mac statefulset % kubectl www-web-0 www-web-1
error: unknown command "www-web-0" for "kubectl"
abhishekveeramalla@aveerama-mac statefulset %
```

Storage class updated as standard in stateful file.

```
 volumeClaimTemplates:
 - metadata:
     name: www
   spec:
     accessModes: [ "ReadWriteOnce" ]
     storageClassName: standard
     resources:
       requests:
         storage: 1Gi
```

```
abhishekveeramalla@aveerama-mac statefulset % kubectl apply -f sample-statefulset.yaml
statefulset.apps/web created
```

```
abhishekveeramalla@aveerama-mac statefulset % kubectl apply -f sample-statefulset.yaml
statefulset.apps/web created
abhishekveeramalla@aveerama-mac statefulset % kubectl get pods -w
NAME    READY   STATUS    RESTARTS   AGE
web-0   1/1     Running   0          5s
```

One pod got created. Now it will start creating second one.

```
abhishekveeramalla@aveerama-mac statefulset % kubectl get pods
NAME    READY   STATUS              RESTARTS   AGE
web-0   1/1     Running             0          22s
web-1   0/1     ContainerCreating   0          2s
abhishekveeramalla@aveerama-mac statefulset % kubectl get pods
NAME    READY   STATUS    RESTARTS   AGE
web-0   1/1     Running   0          31s
web-1   1/1     Running   0          11s
```

Now third one:

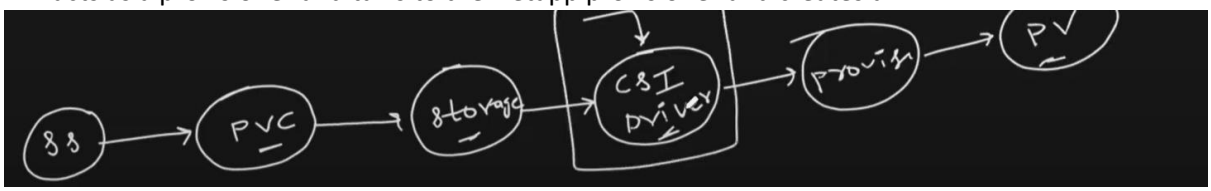
```
abhishekveeramalla@aveerama-mac statefulset % kubectl get pods -w
NAME    READY   STATUS              RESTARTS   AGE
web-0   1/1     Running             0          37s
web-1   1/1     Running             0          17s
web-2   0/1     Pending             0          0s
web-2   0/1     Pending             0          0s
web-2   0/1     Pending             0          1s
web-2   0/1     ContainerCreating   0          1s
web-2   1/1     Running             0          2s
```

```
abhishekveeramalla@aveerama-mac statefulset % kubectl get pods
NAME    READY   STATUS    RESTARTS   AGE
web-0   1/1     Running   0          50s
web-1   1/1     Running   0          30s
web-2   1/1     Running   0          10s
abhishekveeramalla@aveerama-mac statefulset %
```

Why stateful set works like this. Say one db is for read and write fumctions and the other two are for backups. There is no point of creating backup db if the main db is not created successfully.

What if we want to create a PV from an external provisioners. For this there are **CSI Drivers.**

We want to use a Netapp storage service. So, Netapp storage service will provide a storage driver called CSI Driver (Container Storage Interface Driver) we have to install that in our cluster. This driver will acts as a provisioner and talks to the Netapp provisioner and creates a PV.

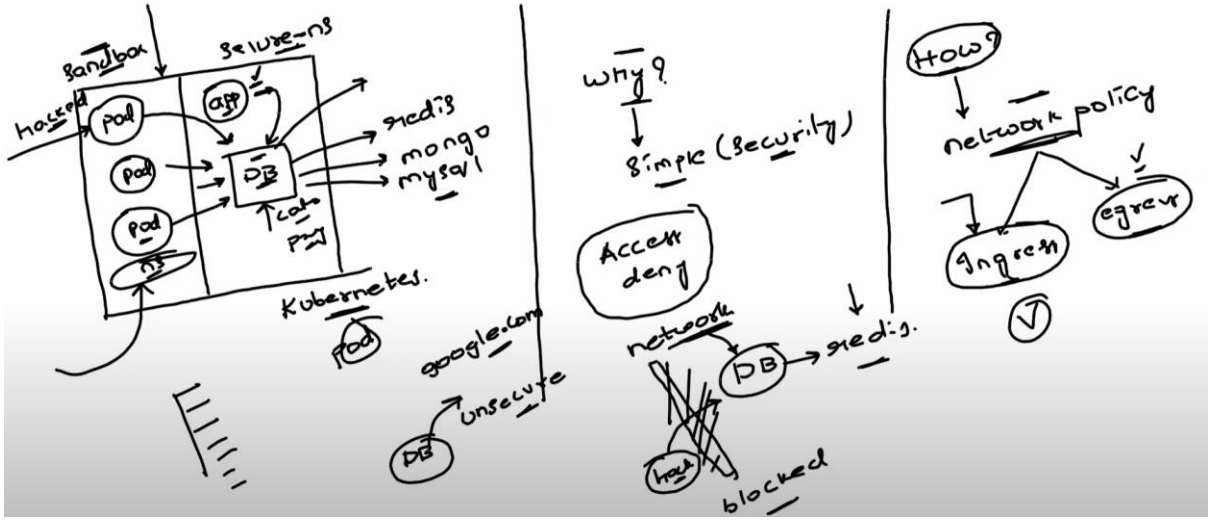


This is the flow with slight change in between. We have added a CSI Driver. Rest all is same.

Day – 5: Network Policy

Ref: https://github.com/iam-veeramalla/kubernetes-troubleshooting-zero-to-hero/tree/main/05-networkpolicy

Suppose there is a K8s cluster with two name spaces. In Secure Name Space there is a DB and one application. This DB should be accessed by only this application. In other namespace called sandbox there are 3 pods and somehow hacker hacked the namespace. He is trying to connect to the database form these pods. We should not allow it. DB should be access by the pods that are meant to be allowed. How do we achieve this? Using Network Policy.



Network Policy is divided into 2 categories. **Ingress** – Using this network policy we can retsrict inward traffic to the database. **Egress –** We can restrict any traffic going out of the pod.

Within our network policy ingress configuration we mention website that our pods should not have access to. Lets try to implement this. We will create a Pod and then DB. Without network policy this Hacked pod will be able to connect with DB. With network policy it should not.

Minikube or Kind clusters will not support network policies. We need to install network Plugins and Calico. So, it is preferred to use EKS cluster that is above or equals to 1.27 version or Openshift or Racher clusters.

We are using Openshift cluster in our example.

```
abhishekveeramalla@aveerama-mac netpol % ls
db.yaml                 hack.yaml               network-policy.yaml
abhishekveeramalla@aveerama-mac netpol %
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: redis
  labels:
    app: redis
spec:
  replicas: 1
  selector:
    matchLabels:
      app: redis
  template:
    metadata:
      labels:
        app: redis
    spec:
      containers:
      - name: redis
        image: "docker.io/redis:6.0.5"
        ports:
        - containerPort: 6379
```

We will deploy a redis database. Let's create a namespace called Secure.

```
abhishekveeramalla@aveerama-mac netpol % kubectl create ns secure-namespace
namespace/secure-namespace created
abhishekveeramalla@aveerama-mac netpol % kubectl apply -f db.yaml -n secure-namespace
deployment.apps/redis created
```

```
abhishekveeramalla@aveerama-mac netpol % kubectl get pods -n secure-namespace
NAME                     READY   STATUS    RESTARTS   AGE
redis-5496dd9bcb-jhb85   1/1     Running   0          13s
abhishekveeramalla@aveerama-mac netpol %
```

We can also login to this database. 1st go inside pod.

```
abhishekveeramalla@aveerama-mac netpol % kubectl -n secure-namespace  exec -it redis-5496dd9bcb-jhb85 --
/bin/bash
1000700000@redis-5496dd9bcb-jhb85:/data$
```

Connect to db.

```
1000700000@redis-5496dd9bcb-jhb85:/data$ redis-cli
127.0.0.1:6379>
1000700000@redis-5496dd9bcb-jhb85:/data$
```

Hacker got access to the default namespace and he is tryig to deploy a httpd pod.

```
abhishekveeramalla@aveerama-mac netpol % vim hack.yaml
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: httpd-deployment
spec:
  selector:
    matchLabels:
      app: httpd
  replicas: 1
  template:
    metadata:
      labels:
        app: httpd
    spec:
      containers:
      - name: httpd
        image: httpd:latest
        ports:
        - containerPort: 80
```

He is trying to install httpd pod inside the default namespace.

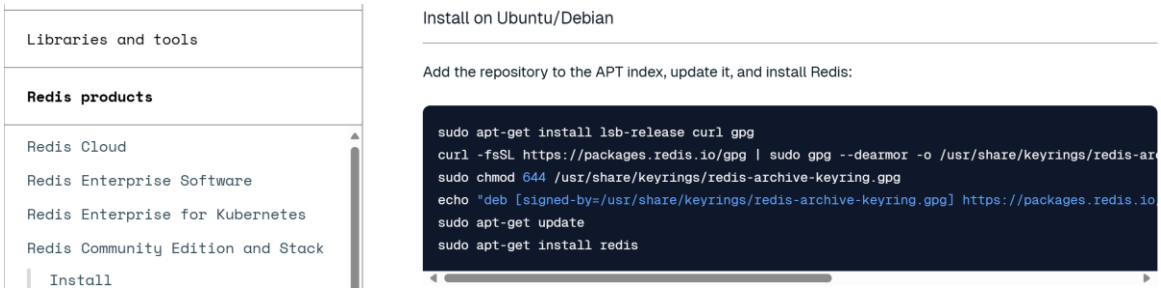
```
abhishekveeramalla@aveerama-mac netpol % kubectl apply -f hack.yaml
deployment.apps/httpd-deployment created
abhishekveeramalla@aveerama-mac netpol %
```

```
abhishekveeramalla@aveerama-mac netpol % kubectl exec -it deploy/httpd-deployment -- /bin/bash
root@httpd-deployment-6d6b866d8f-g9vvc:/usr/local/apache2#
```

Once he had access to this pod. They need to install a client to connect to the database. Since we are using Redix, lets try to install redis cli in order to connect to the database in another pod.

Ref: https://redis.io/docs/latest/operate/oss_and_stack/install/install-redis/install-redis-on-linux/#:~:text=Most%20major%20Linux%20distributions%20provide%20packages%20for%20Redis.%20Add%20the

Follow the instructions:



Execute everything one by one in this httpd pod. Lets check if redis cli is installed.

```
root@httpd-deployment-6d6b866d8f-g9vvc:/usr/local/apache2# redis-cli
Could not connect to Redis at 127.0.0.1:6379: Connection refused
not connected>
root@httpd-deployment-6d6b866d8f-g9vvc:/usr/local/apache2#
```

If you give this command it will automatically try to connect to the redis db in this pod. Since ther eis no redis db in this pod. It is not able to connect.

Now, we will try to connect to the redis db in the secure namespace.

Get the IP of the db pod in secure namepsace:

```
abhishekveeramalla@aveerama-mac netpol % kubectl get pods -n secure-namespace -o wide
NAME                     READY   STATUS    RESTARTS   AGE    IP            NODE
NOMINATED NODE   READINESS GATES
redis-5496dd9bcb-jhb85   1/1     Running   0          6m9s   10.131.0.44   ip-10-0-28-52.us-west-1.compute.internal
<none>           <none>
abhishekveeramalla@aveerama-mac netpol %
```

Now, run this command in your httpd pod.

```
root@httpd-deployment-6d6b866d8f-g9vvc:/usr/local/apache2# redis-cli -h 10.131.0.44
10.131.0.44:6379>
```

Now we have access to the database. Our database is compromised.

We will use Ingress Network policy to eliminate such issues.

Ref: https://kubernetes.io/docs/concepts/services-networking/network-policies/

You can restrict access from IP's, namespaces or Pods.

In the Network policy YAMl file in the K8s docs it had both ingress & egress. You can modify the file as per your requirement.

The labels you provided will no able to talk to the db.

```
ingress:
- from:
  - ipBlock:
      cidr: 172.17.0.0/16
      except:
      - 172.17.1.0/24
  - namespaceSelector:
      matchLabels:
        project: myproject
  - podSelector:
      matchLabels:
        role: frontend
```

```
abhishekveeramalla@aveerama-mac netpol % ls
db.yaml                 hack.yaml               network-policy.yaml
abhishekveeramalla@aveerama-mac netpol % vim network-policy.yaml
```

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: redis-network-policy
  namespace: default
spec:
  podSelector:
    matchLabels:
      app: redis
  policyTypes:
  - Ingress
  ingress:
  - from:
    - podSelector:
        matchLabels:
          role: known-redis-member
    ports:
    - protocol: TCP
      port: 6379
```

Namespace we mentioned wrong change it to secure-namespace as we will try dpeloy this in secure-namespace only. We are trying to apply the network policy to the redis pod that is we mentioned in spec → Pod Selector. Under policy types we mention the restrictions.

Let's check whether redis db had this lable or not once.

```
abhishekveeramalla@aveerama-mac netpol % vim db.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: redis
  labels:
    app: redis
spec:
  replicas: 1
```

if we have multiple databases then we can create one network policy for every db and add appropriate label.

So, only a pod with lable: role as known-redis-member will be able to access the db.

```
  policyTypes:
  - Ingress
  ingress:
  - from:
    - podSelector:
        matchLabels:
          role: known-redis-member
    ports:
    - protocol: TCP
      port: 6379
```

Our Httpd pod had different label.

```
metadata:
  name: httpd-deployment
spec:
  selector:
    matchLabels:
      app: httpd
  replicas: 1
  template:
    metadata:
      labels:
        app: httpd
```

So, this pod should be blocked from accessing db in other namespace.

```
abhishekveeramalla@aveerama-mac netpol % kubectl apply -f network-policy.yaml -n secure-namespace
networkpolicy.networking.k8s.io/redis-network-policy created
```

Let's try to connect to the redis dbagain form inside the httpd pod:

```
root@httpd-deployment-6d6b866d8f-g9vvc:/usr/local/apache2# redis-cli
Could not connect to Redis at 127.0.0.1:6379: Connection refused
not connected>
root@httpd-deployment-6d6b866d8f-g9vvc:/usr/local/apache2# redis-cli -h 10.131.0.44
10.131.0.44:6379>
root@httpd-deployment-6d6b866d8f-g9vvc:/usr/local/apache2# redis-cli -h 10.131.0.44
```

We were not able to connect.

As mentioned in the document we can restrict the ingress in three ways only. Egress with 2 ways.

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: test-network-policy
  namespace: default
spec:
  podSelector:
    matchLabels:
      role: db
  policyTypes:
  - Ingress
  - Egress
  ingress:
  - from:
    - ipBlock:
        cidr: 172.17.0.0/16
        except:
        - 172.17.1.0/24
    - namespaceSelector:
        matchLabels:
          project: myproject
    - podSelector:
        matchLabels:
          role: frontend
    ports:
    - protocol: TCP
      port: 6379
  egress:
  - to:
    - ipBlock:
        cidr: 10.0.0.0/24
    ports:
    - protocol: TCP
      port: 5978
```

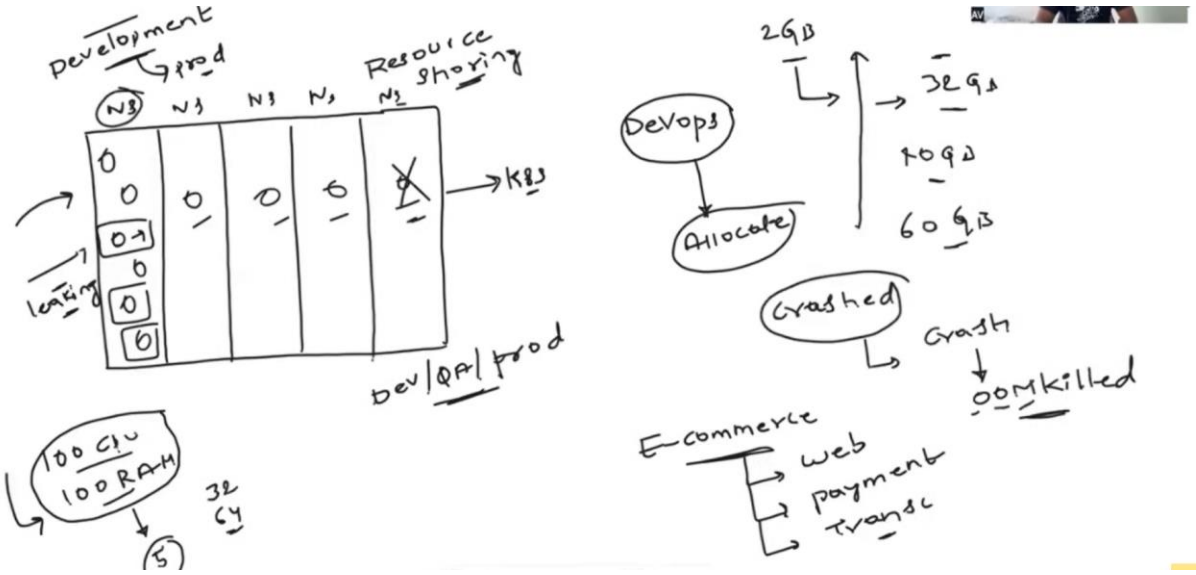
Real time K8s Problems

Problem 1: There is K8s cluster of production. We have multiple projects running in the namespace. Say like STS and VX. So, we need have multiple namespaces. A cluster will have multiple nodes. Let's say we have total of 100 CPUs and 100 GB RAM for all the nodes. In real one node will take about 32/64 GB RAM and 1/2/3/4 CPUs based on Organisations requirements.

We are trying to deploy one application in Nmaespace 1 and one/two of the pods are taking too much memory than allocated. Say one should only consume 2-3 GB of RAM but they are consuming nearly 30 GB memory. Then some service in any other namespace will definitely get's crash. Pod falls into CarshLoopBackOff due to OOMKilled. As out of 100GB Namespace 1 itself is taking 40 GB and other namespace will have to use rest of it. There are chances of this getting occur.

As a DevOps Engineer it is our responsibility to allocate the resources based on the requirement to a particular namespaces of the K8s Cluster properly.

Solution to it is, use Resource Quota and Resource Limits.



For each namepsace we will allocate the resource quota. It is a limit we will apply to the namepsace. So, the services inside this namespace should not consume more space than the allocated. Say like 15 GB RAM. We can set limit with respective to both CPU and RAM.

Developers need to tell how much GB they want. They need to do **performance bench marking.** Then they need to comeup with a ideal number.

Since we allocated 15 GB ram for the name space using resource quota one challenge with namespace is solved. 50% resolved.

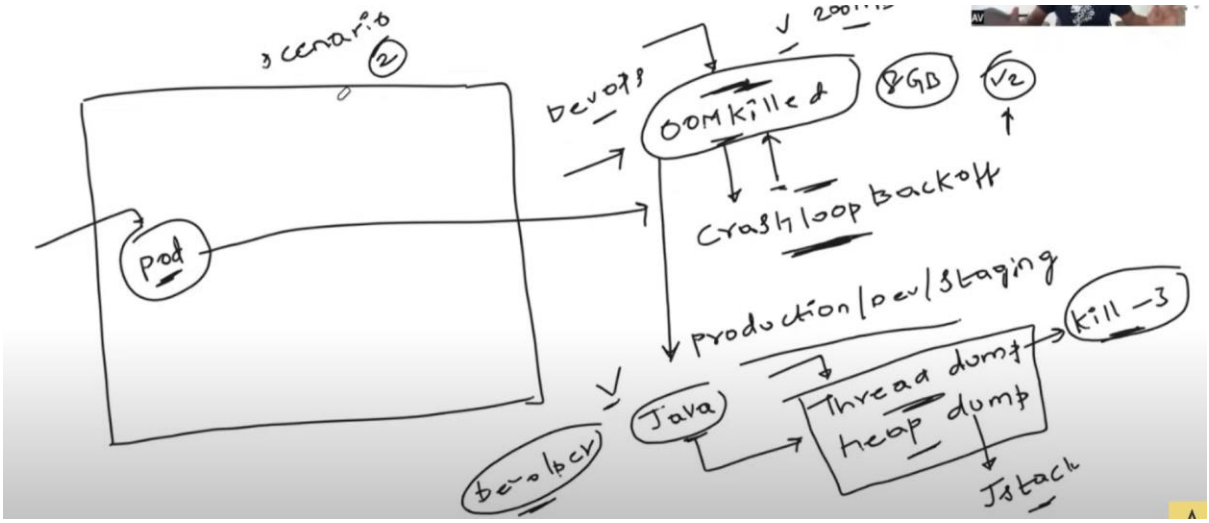
When we allocated 15GB means there will be many services in the namespace. Say, there is one pod that is leaking memory and this time only this namepsace will be effected. Resource limit needs to be setup here. We can also add resource requests which limits the requests coming in.

Resource limit is a limit that will be setup for pods, where as resource quota is for namespace level.

Each pod will have a limit now. Development team will take the performance bench marking of each microservice. When we deploy these application then we will use resource limits in our YAML files to limit the pods.

Doing these Blast radius come down to one pod.

Problem 2:



When a pod is getting status as Crashloopbackoff and error is OOMKilled. Then Based on application built language we need to provide thread Dump and Heap Dump to the developers. They will resolve the issue and come up with a new version of application.

Thread dumps and heap dumps are diagnostic tools used to troubleshoot and analyze issues in Java applications, particularly in production environments. They provide insights into the state of the application's threads and memory, respectively, helping engineers understand performance bottlenecks, memory leaks, deadlocks, and other issues.

A **thread dump** is a snapshot of all the threads running in a Java Virtual Machine (JVM) at a given point in time.

A **heap dump** is a snapshot of the memory used by a Java application. It shows the objects in the JVM's heap memory at a specific point in time, along with their sizes, references, and types.

These properties may change based on Languages.

Problem 3: Upgrades

One of the common challenege for a devops engineern is upgrades. We might have to upgrade K8s form 1.29 to 1.30

Senior Devops engineer will create a detailed manual for doing this upgrade. In the manual things like how to do backup of resources, release notes, steps divided for control plane components (specify steps for upgrading etcd, api server, scheduler) and Data plane components (How to upgrade Worker nodes).

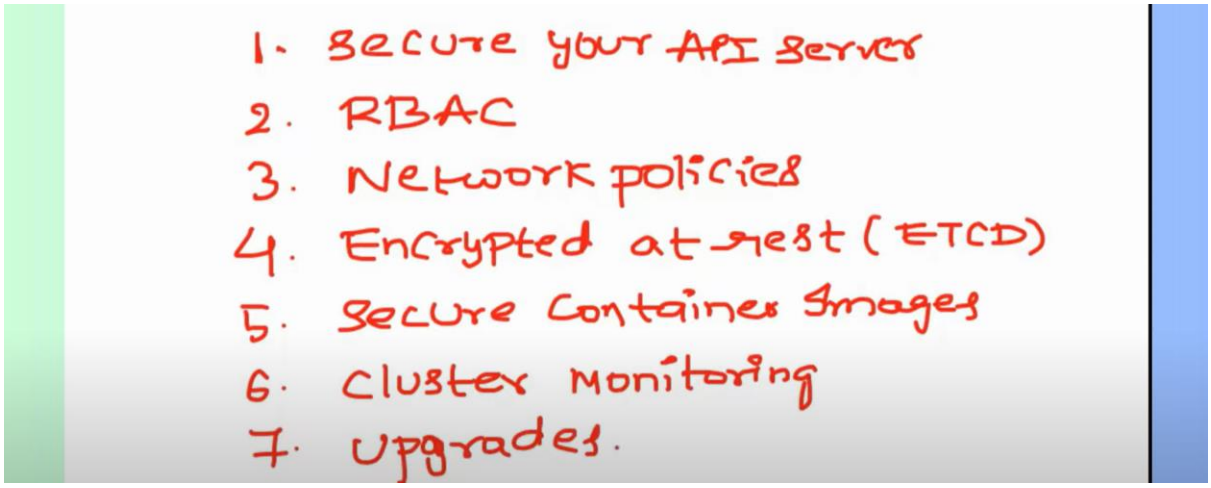
In real time worker nodes will be updated in this way:

1st we will drain the node. Pods will be moved to different node/nodes. Once the node is empty we will make it unschedulable (Not allowing to get any pods scheduled to this node). We will upgrade the Kubelet to new version and install other packages required to upgrade. Now this node will be at 1.30 version and other nodes at 1.29 version. Then we will bring this node up and joins back it into the k8s cluster. And we will remove the Taint or unschedulable on this node. Same happens to other nodes as well.

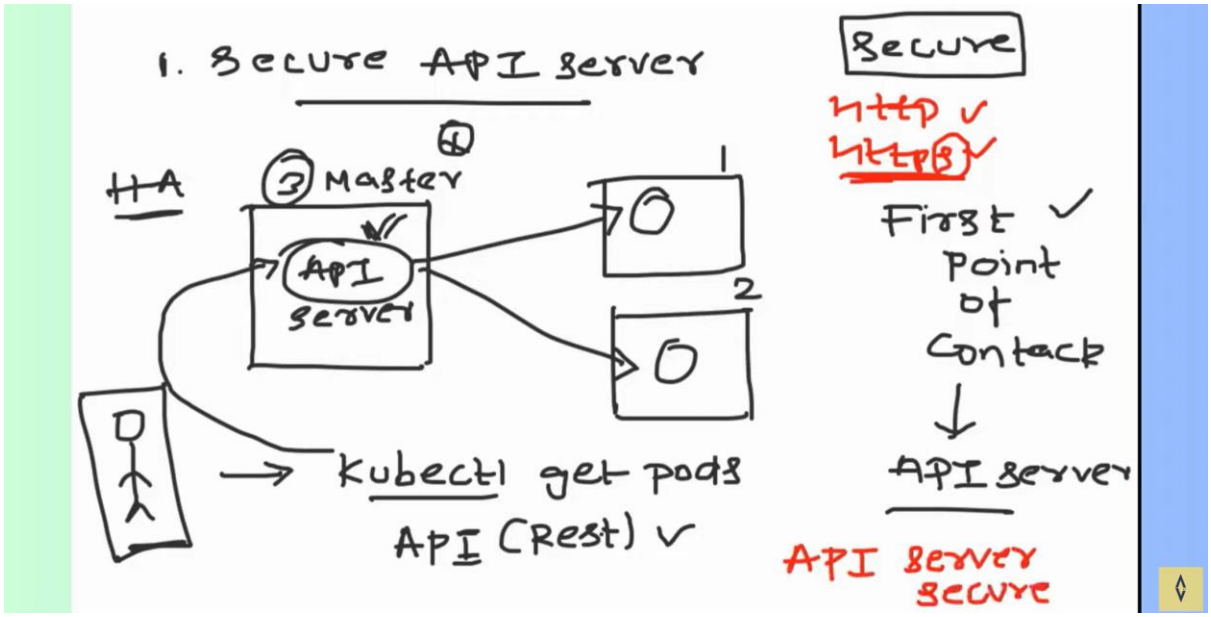
Secure Kubernetes like a Pro

Video Ref:
https://www.youtube.com/watch?v=qFwBKLZnPlQ&list=PLdpzxOOAlwvJdsW6A0jCz_3VaANuFMLpc&index=25

To secure k8s cluster like a pro, we need to maintain these.



1. Secure API Server:



First point of contact in K8s cluster is Kube API Server. If we give above command it talks to API server only. Usually, we will maintain 3 Master nodes to have High Availability. If one crashes then other two take care of functioning cluster.

If API server is secured then our Cluster is also Secured. Same like HTTP and HTTPS. HTTPS is more secured than HTTP.

How to secure API Server?

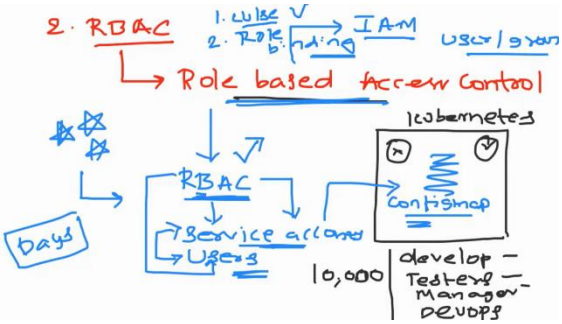
In K8s every resource is a Pod. So, Kube-API server is also a pod. So, inside the API Server's pod.yaml we will secure the API server with TLS Certificates. These certificates can be self signed or brought from any certificate provides like go daddy.

One -way: Get the api server pod name using **kubectl get pods -a** then edit the pod.yaml file of this pod using kubectl edit command then add TLS certificates under spec section.

Other – Way of securing the Api Server:

Restricting the access using **RBAC**. Role Based Access Control. Based on the role you will be able access components in the K8s using service, role and role binding.

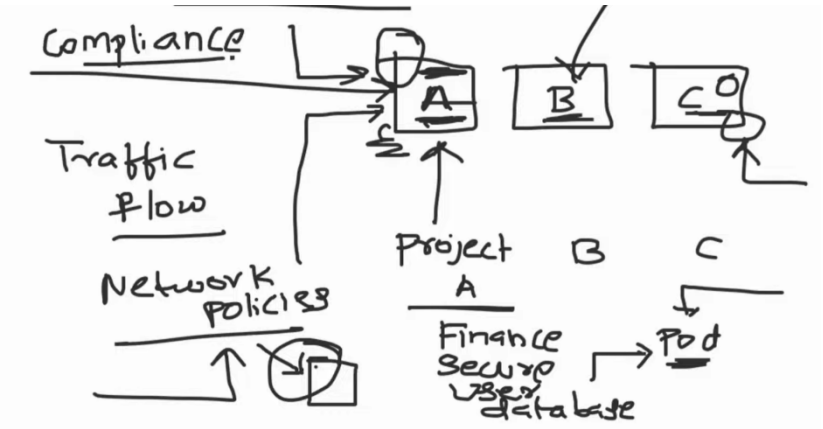
2. Role Based Access Control:



RBAC works on three topics: Service Accounts, Roles and Role Binding.

We need to define access to service accounts like pods, services, deployments because they also should be defined what to access what bot to access. Then users, for them we will create roles and then bind the roles to the users using Role Binding. This is mandatory for any K8s Cluster.

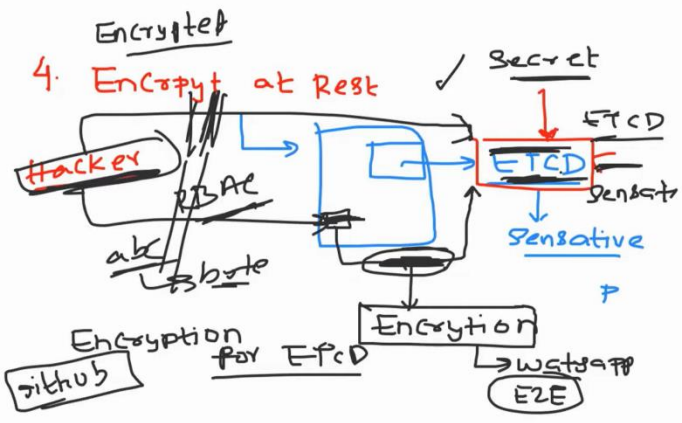
3. Network Policies:



If there are three namespaces and then A had most secure applications like user database and finance application then anybody access these pods will have data with them. So, we need to define network policies in the namespace to avoid unnecessary traffic flow.

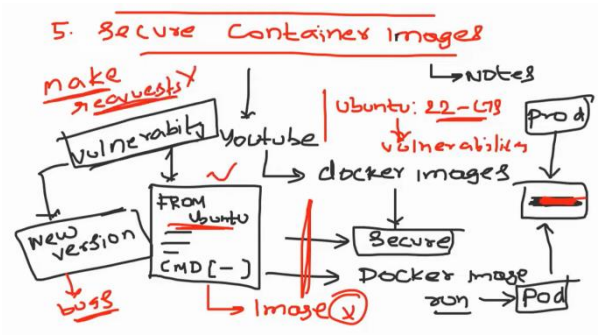
4. Encrypt Data At Rest:

One of the important component of the K8s master is **ETCD.** It is a key value K8s objects store. It stores all the sensitive information about the objects of the K8s. Sensitive information in the sense, ConfigMaps, Secrets like password, TLS certificates, API keys, etc.



RBAC should be implemented to have access to secrets. If user not part of role then they can't access secrets. But somebody accessing ETCD can have these secrets as they will also be created in ETCD. So, when we create a secret and it is getting saved in ETCD, we need to create Encryption mechanism.

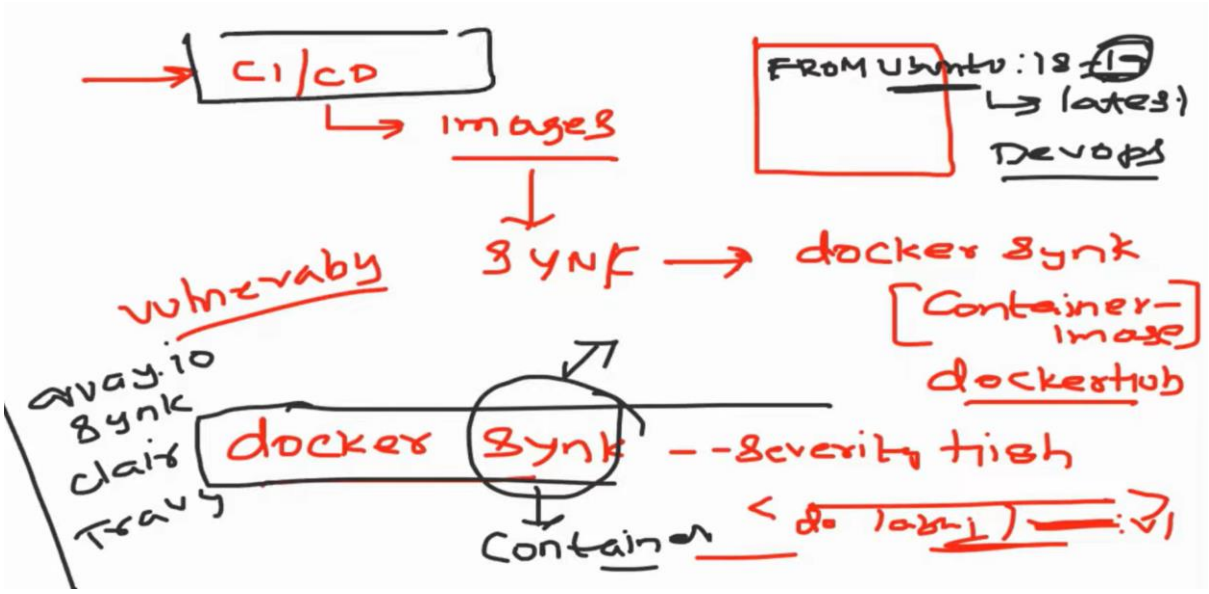
5. Secure Container Images:



The docker image we are creating and then make it as a container to run in our pods is secure or not. How do we make sure the conatiner secure. While create an Image using docker file. We were using from Ubuntu. This ubuntu might have a vulnerability just like android version had some bugs. So, if we haven't made it secure then it will be deploy inot the prod cluster.

As a DevSecOps Engineer, In the CICD pipelines or where ever we were creating these images, we need to use **Docker SYNK.**

Along with that we just need to pass an argument and then docker image link in the container registry. We can use other than synk also like clair, travy, etc.

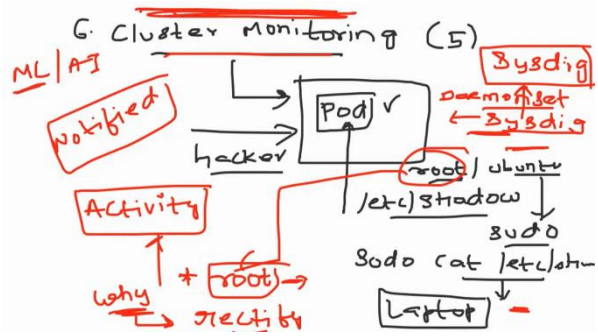


6. Cluster Monitoring:

Sysdig is one of the cluster monitoring tool.

No container should run as root user. If any container is running as a root user or accessing files in etc, then cluster monitoring tool should notify about this activity.

A hacker can access a pod and then get inside the pod and get all the sensitive data by running as a root user or ubuntu user.



7. Upgrades:

We need to ensure the tools we are using should be up to date. Because old version might have vulnerabilties. Same with k8s cluster, we need to upgrade the components of k8s as well. Old verison might a security vulenrabilities then in the new version they will be removed.