**SESSION - 58**

**Kubernates**

**-----------------**

--> 1. building the image

--> 2. running the image

--> docker is all about building the image we are not going to use docker for orchestration. Because we have disadvantages.

--> we are going to use for container orchestration.

--> we will develop dockerfiles in our laptop.

--> then pushed to github.

--> docker install in work station there build the image and pushed to docker hub.

--> workfolws will run in nodes groups.

--> application run like container.

--> everything is computer. In background AWS will manage.

--> image pulling by nodes.

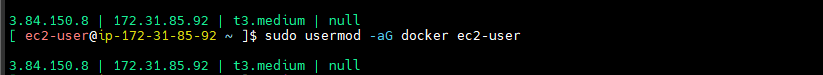
--> present we are using Dockerhub but in further systems will use ECR.

--> it can be everywhere to u have to give proper address.

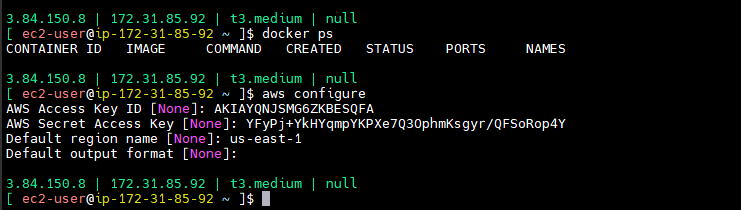
--> if you are given any address that will go docker hub and will check defaultly.

--> cluster is ready

--> **sudo usermod -aG docker ec2-user**



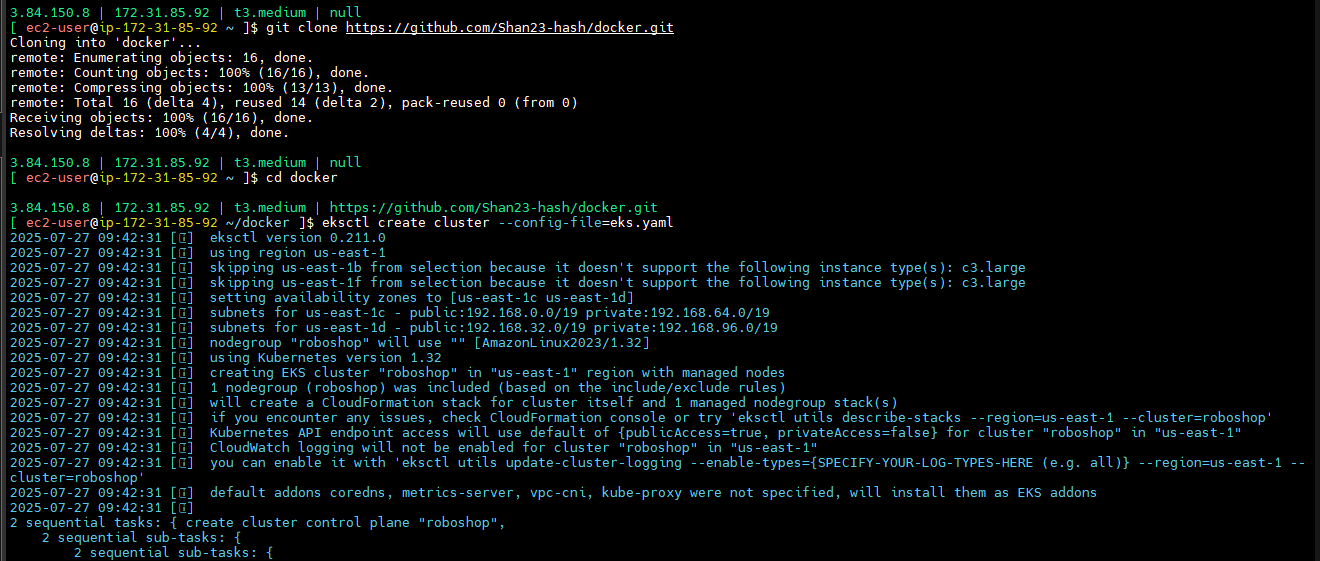
**--> aws configure**



--> **git clone https://github.com/Shan23-hash/docker.git**

--> **cd ../docker**

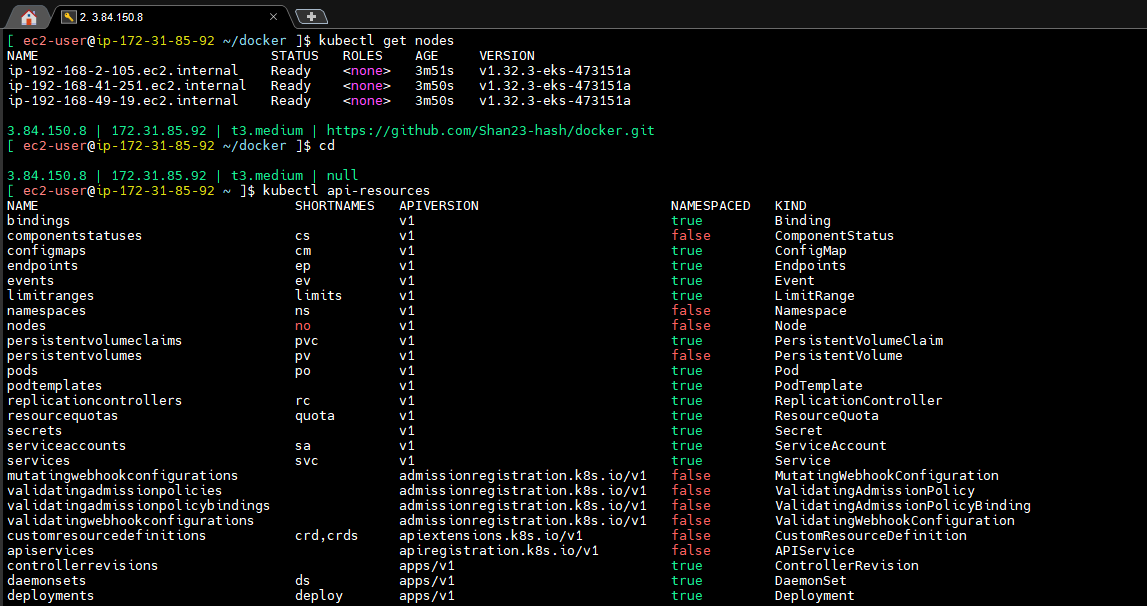
--> **eksctl create cluster --config-file=eks.yaml**



-->

--> **kubectl get nodes**

--> **kubectl api-resources**



--> everything is a resource in kubernates.

--> **syntax:**

apiVersion: v1

kind:

metadata:

name:

labels:

spec:

containers:

- name:

image:

- name:

image:

--> one pod can have so many containers.

--> pod is one smallest deployment unit in kubermates.

**03-multi-container.yaml**

apiVersion: v1

kind: Pod

metadata:

name: multi-container

labels:

project: roboshop

purpose: multi-container-demo

spec:

containers:

- name: nginx

image: nginx

- name: almalinux

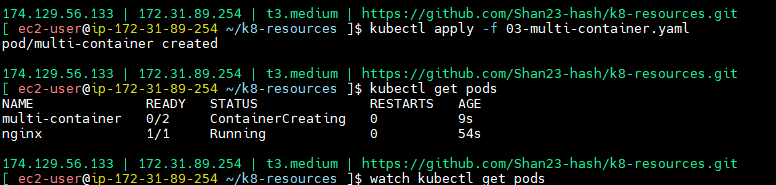
image: almalinux:9

--> push and pull the code

--> git clone URL

--> **kubectl apply -f 03-multi-container.yaml**

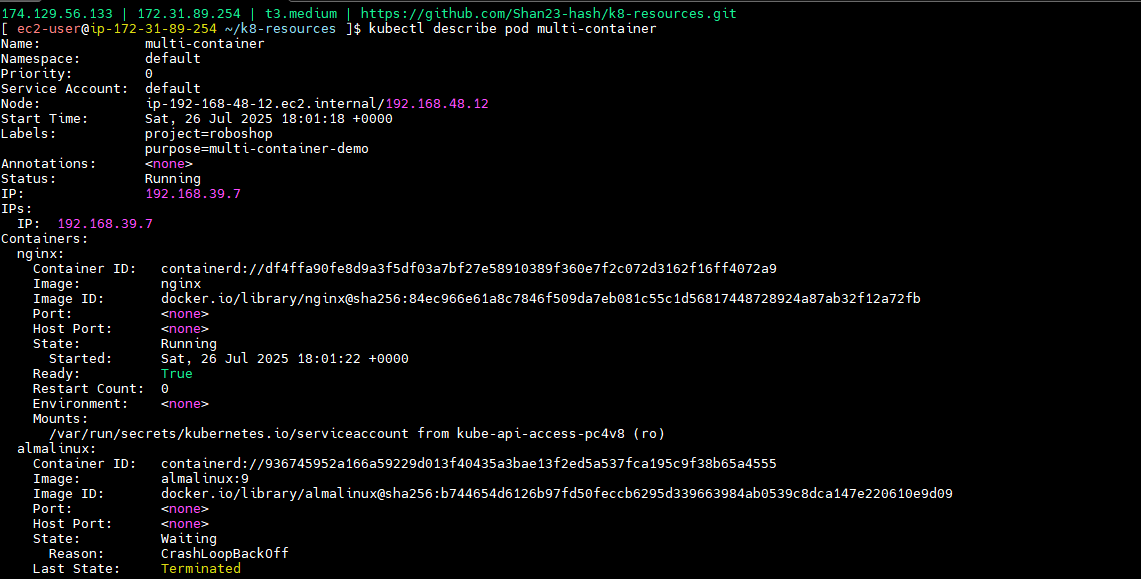
--> **kubectl get pods**



--> So creating containers.

--> **watch kubectl get pods**

--> **kubectl describe pod multi-container** -- if you want see error



--> one container need to run non stop that one should be running in infinity time.

--> that keep container for infinity pod.

--> CMD will be in continuous running.

--> generally we are giving

command: ["sleep", "1000"]

--> you should have some command to keep the container busy to run infinite time.

--> if you are not running continuously you will get error

--> back-off restarting failed error -- famous error

--> CrashLoopBackOff -> container is unable to start

--> push and pull the code

--> **kubectl delete -f 03-multi-container.yaml** -- got error that why deleting and applying

--> **kubectl apply -f 03-multi-container.yaml**

**--> kubectl get pods**

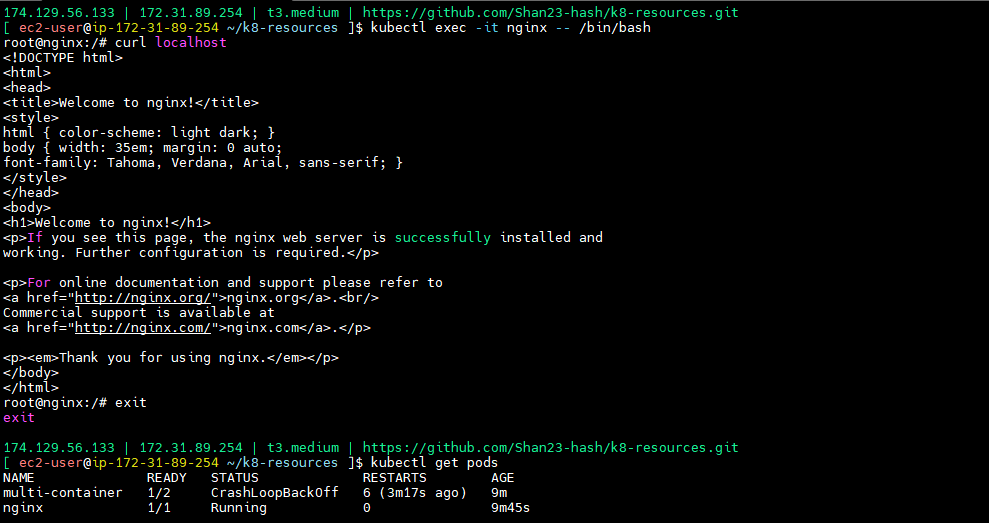
**--> kubectl apply -f 02-pod.yaml**

**--> kubectl get pods**

**--> kubectl exec -it nginx --bash**

**--> curl localhost** -- I’m in inside nginx pod, I’m getting the local page of nginx.

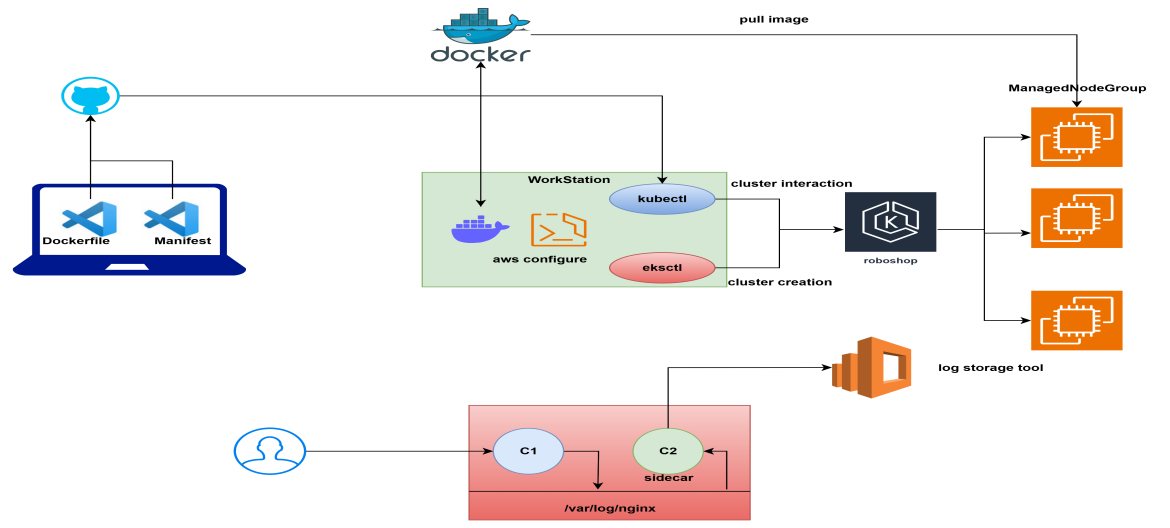
--> **kubectl get pods**



--> here two is there which one I want to login mention name

**--> kubectl exec -it multi-container -c almalinux -- bash --** I’m login into alma linux, now I’m in multi-container pod in alma linux.

--> why in one pod two containers? --



**Container vs Pod**

pod is the smallest deployable unit in kubernates, a pod can have 1 or many containers. all containers inside pod share same network and storage.

--> pods all are temporary.

--> In elastic log we have to store this pods. This is log storage tool.

--> here we will run another container called sidecar.

--> it has some extra responsibilities.

--> because I said same storage. Responsibility is access the logs and pushed to log storage tool.

--> if pods deleted also it will available in log storage tool.

--> usually multi container will useful in sidecar patterns and proxy patterns

--> proxy means proxy get the request.

multi containers are useful in sidecar patterns and proxy patterns. proxy means first proxy container gets the request, it checks whether request should be forwarded to main container or not.

--> environment two nginx in same pod. So because those two containers try to block port no 80.

--> everything you can store it in single pod but port numbers should be different.

catalogue, user, cart, shipping, payment, frontend can be in same pod as multiple containers but diff ports should be opened by containers..

--> theoretically it is possible but practically you will not if you did like this that is not container approach.

--> each application have each container that is better. Because if one container is down that one only will down.

**Kubernates and labels**

--> labels will use in kebernates have some functionality.

**04-labels.yaml**

apiVersion: v1

kind: Pod

metadata:

name: labels

labels:

project: roboshop

component: mysql

tier: backend

environment: dev

spec:

containers:

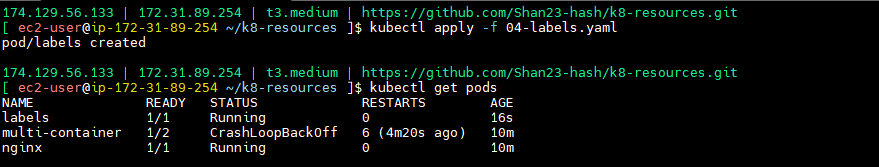
- name: nginx

image: nginx

--> Push and pull the code

--> **kubectl apply -f** **04-labels.yaml**

**--> kubectl get pods**



**-->** labels will use for use

--> Labels are key value pairs

lables are used as selectors for other resources inside kuberentes... useful for filtering resources. lables values we can't keep long values, only 63char

--> labels will use for selecting interal service.

annotations are similar to labels, it is key value pair, but not used as selectors. annotations are used to provision external resources to kuberentes. we keep external information build url, image registry, etc...annotations values can be 256char

--> annotations will use for external services like load balancer, target group

--> labels is metadata

--> annotations also meta data external information build url, image registry some kind of this information.

--> annonations also key value pairs but values can be complex like URL.

Labels vs annotations ( search in google)

|  |  |  |
| --- | --- | --- |
| **Feature** | **Labels** | **Annotations** |
| Purpose | Identification, grouping, selection | Non-identifying metadata, decoration |
| Kubernetes Usage | Used by Kubernetes for selection | Not used by Kubernetes for selection |
| Length/Format | Limited length, specific characters | No length restrictions, any characters |

**05-annotations.yaml**

apiVersion: v1

kind: Pod

metadata:

name: annotations

labels:

project: roboshop

component: mysql

tier: backend

environment: dev

annotations:

description: "This pod is created to demonstrate pod annotations"

jenkins: "https://jenkins.com/build/job/roboshop-catalouge/3"

spec:

containers:

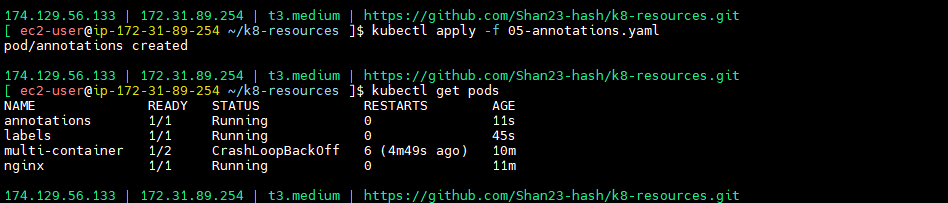
- name: nginx

image: nginx

--> push and pull the code

--> **kubectl apply -f** **05-annotations.yaml**

**--> kubectl get pods**



**resource limiting**

**=================**

--> in Container Main advantage is it can use the resources dynamically

--> it will not block like vms.

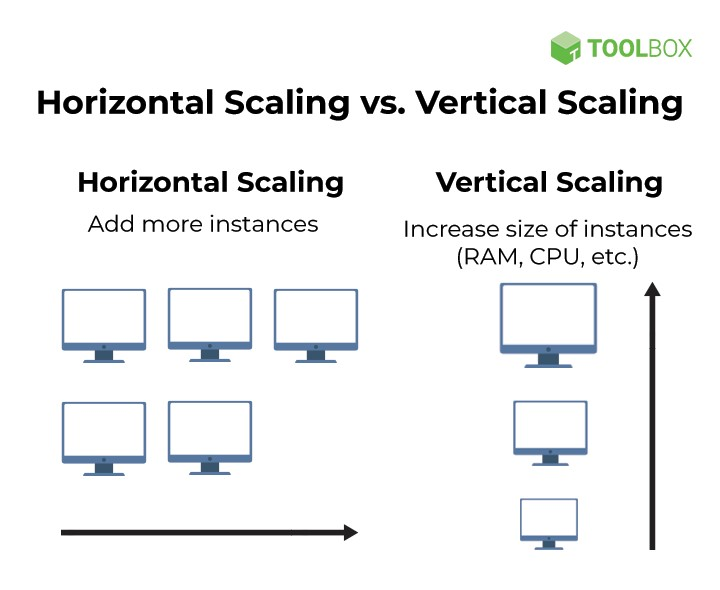
--> Containers have freedom to use as many resources as it pods.

--> advantage can become, disadvantage any time.

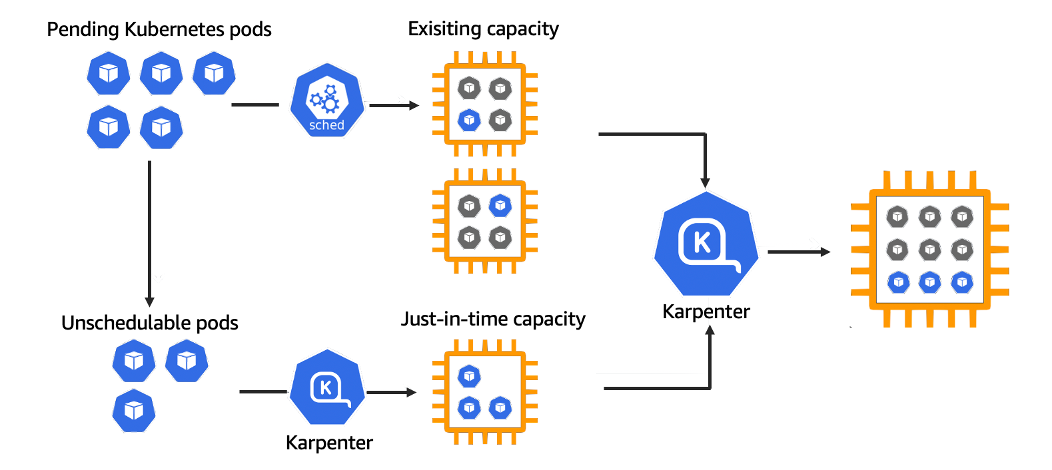
--> kubernates resource limit -- search in google

1 processor = 1000m, 250m = 0.25

**horizontal scaling vs vertical scaling**



|  |  |  |  |
| --- | --- | --- | --- |
|  | Horizontal scaling | Vertical Scaling | |
| Description | Increase or decrease the number of nodes in a cluster or system to handle an increase or decrease in workload | Increase or decrease the power of a system to handle increased or reduced workload |
| Example | Add or reduce the number of virtual machines (VMs) in a cluster of VMs | Add or reduce the CPU or memory capacity of the existing VM |
| Execution | Scale in/out | Scale up/down |
| Workload distribution | Workload is distributed across multiple nodes.  Parts of the workload reside on these different nodes | A single node handles the entire workload. |
| Concurrency | Distributes multiple jobs across multiple machines over the network at a time. This reduces the workload on each machine | Relies on multi-threading on the existing machine to handle multiple requests at the same time |
| Required architecture | Distributed | Any |
| Implementation | Takes more time, expertise, and effort | Takes less time, expertise, and effort |
| Complexity and maintenance | Higher | Lower |
| Configuration | This requires modifying a sequential piece of logic to run workloads concurrently on multiple machines | No need to change the logic. The same code can run on a higher-spec device |
| Downtime | No | Yes |
| Load balancing | Necessary to actively distribute the workload across the multiple nodes | Not required in the single node |
| Failure resilience | Low because other machines in the cluster offer backup | High since it’s a single source of failure |
| Costs | High costs initially; optimal over time | Low-cost initially; less cost-effective over time |
| Networking | Quick inter-machine communication | Slower machine-to-machine communication |
| Performance | Higher | Lower |
| Limitation | Add as many machines as you can | Limited by the resource capacity, the single machine can handle |



1 house 20 year back --> 4

This is called as vertical scaling

1. you shouldn't stay in the house while remodelling

2. your basement may not be strong to bear another 2 floors

--> Sever got full continuously in that RAM and CPU you want to change you should stop the server. Add the RAM.

--> if I increased that is vertical scaling.

--> vertical scale will have down time and single point of failure.

--> Horizontal scaling is parallelly another one.

--> Horizontal scaling is always better.one is busy create another pod that one also busy create one more that is called Horozontal.

--> always prefer horizontal.

**06-resources.yaml**

apiVersion: v1

kind: Pod

metadata:

name: resources

spec:

containers:

- name: nginx

image: nginx

resources:

requests: # soft limit

memory: "68Mi"

cpu: "100m"

limits: # hard limit

memory: "128Mi"

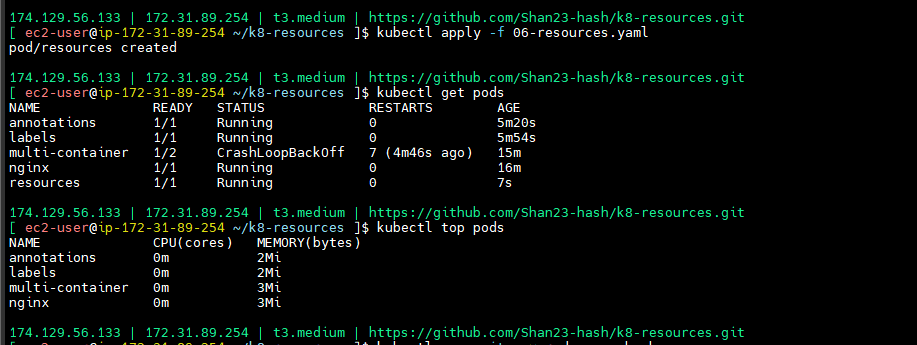
cpu: "150m"

--> Push and pull the code

--> **kubectl apply -f** **06-resources.yaml**

**--> kubectl get pods**

**--> kubectl top pods --**  for checking each one memory (memory 2mb present usage)



--> architects will measure

**Kubernates POD Environment**

**07-env.yaml**

apiVersion: v1

kind: Pod

metadata:

name: envar-demo

labels:

purpose: demonstrate-envars

spec:

containers:

- name: nginx

image: nginx

env:

- name: COURSE

value: "KUBERENETS"

- name: TRAINER

value: "SIVAKUMAR REDDY M"

--> see the code container and give env it will show above all

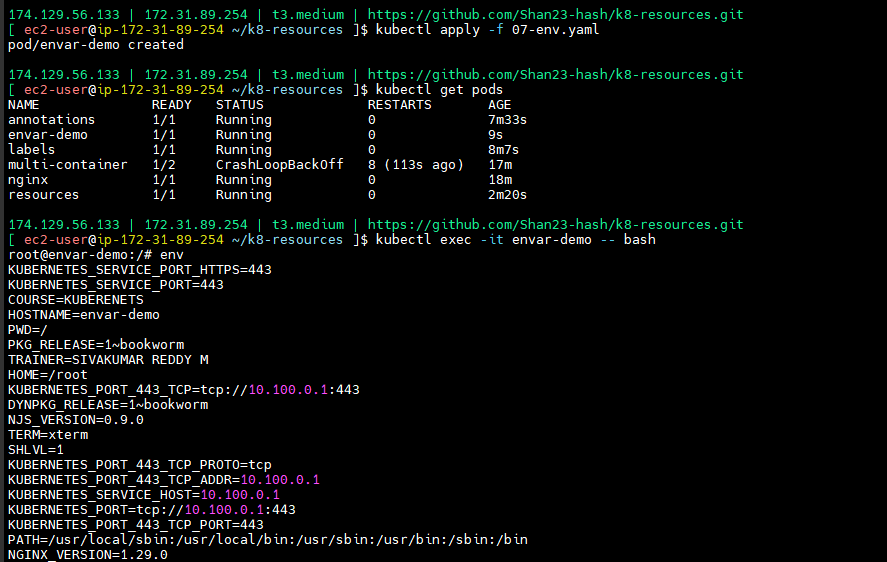
--> Push and Pull the code

--> **kubectl apply -f** **07-env.yaml**

**--> kubectl get pods**

**--> kubectl exec -it envar-demo -- bash**

**--> env** -- you can see those variables here



--> as an application I may need atleast 20 env variables

--> code and configuration. try to keep configuration always outside(loose coupling)

--> we try to keep the configuration outside of pod definition. i.e configmap

--> map means key value pair -- so you keep these in resource called config flag and refer it inside the code.

**CONFIG MAP**

A ConfigMap is an API object used to store non-confidential data in key-value pairs. [Pods](https://kubernetes.io/docs/concepts/workloads/pods/" \o "" \t "https://kubernetes.io/docs/concepts/configuration/configmap/_blank) can consume ConfigMaps as environment variables, command-line arguments, or as configuration files in a [volume](https://kubernetes.io/docs/concepts/storage/volumes/" \o "" \t "https://kubernetes.io/docs/concepts/configuration/configmap/_blank).

A ConfigMap allows you to decouple environment-specific configuration from your [container images](https://kubernetes.io/docs/reference/glossary/?all=true" \l "term-image" \o "" \t "https://kubernetes.io/docs/concepts/configuration/configmap/_blank), so that your applications are easily portable.

**08-config-map.yaml**

apiVersion: v1

kind: ConfigMap

metadata:

name: pod-config

labels:

purpose: config

env: dev

data:

COURSE: "DevOps with AWS"

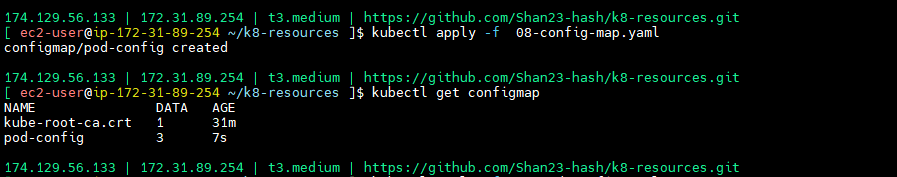
DURATION: "120HRS"

TRAINER: "SIVAKUMAR REDDY M"

--> Push and pull the code

--> **kubectl apply -f**  **08-config-map.yaml**

**--> kubectl get configmap**



**09-pod-config.yaml**

apiVersion: v1

kind: Pod

metadata:

name: env-configmap

spec:

containers:

- name: nginx

image: nginx

envFrom:

- configMapRef:

name: pod-config

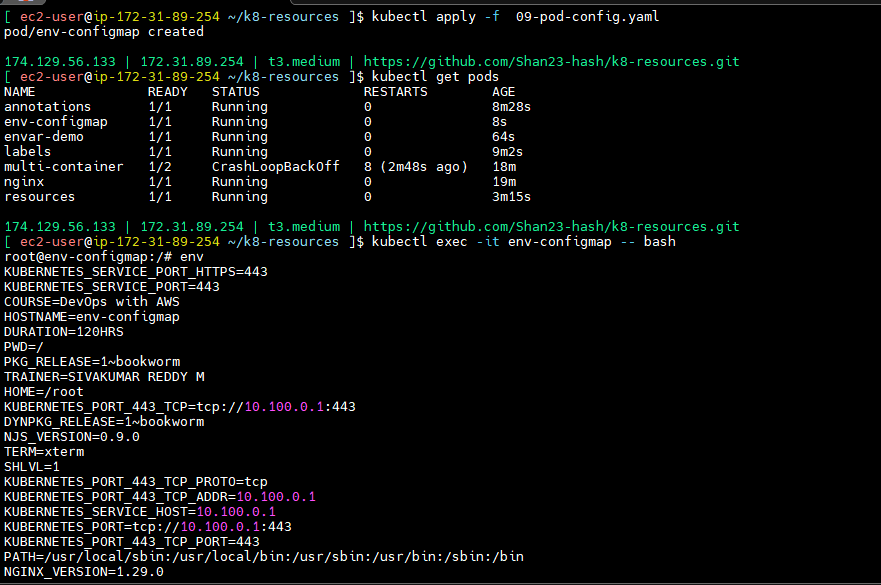
--> push and pull the code

--> **kubectl apply -f** **09-pod-config.yaml**

--> **kubectl get pods**

**--> kubectl exec -it env-configmap -- bash**

**--> env** -- see the mentioned values



1. **secret.yaml**

apiVersion: v1

kind: Secret

metadata:

name: pod-secret

labels:

purpose: secret

env: dev

type: Opaque

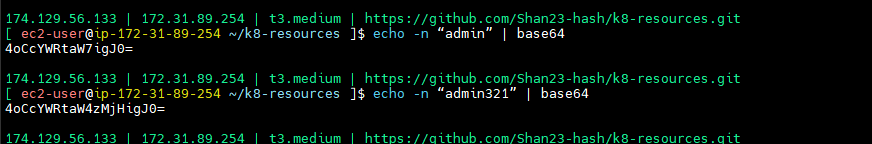
data:

username: "YWRtaW4="

password: "YWRtaW4zMjE="

--> in git bash -- **echo -n “admin” | base64**

--> **echo -n “admin321” | base64**



--> Push and pull the code

1. **pod-secret.yaml**

apiVersion: v1

kind: Pod

metadata:

name: env-secret

spec:

containers:

- name: nginx

image: nginx

envFrom:

- secretRef:

name: pod-secret

--> **echo -n “admin321” | base64**

--> push and pull the code

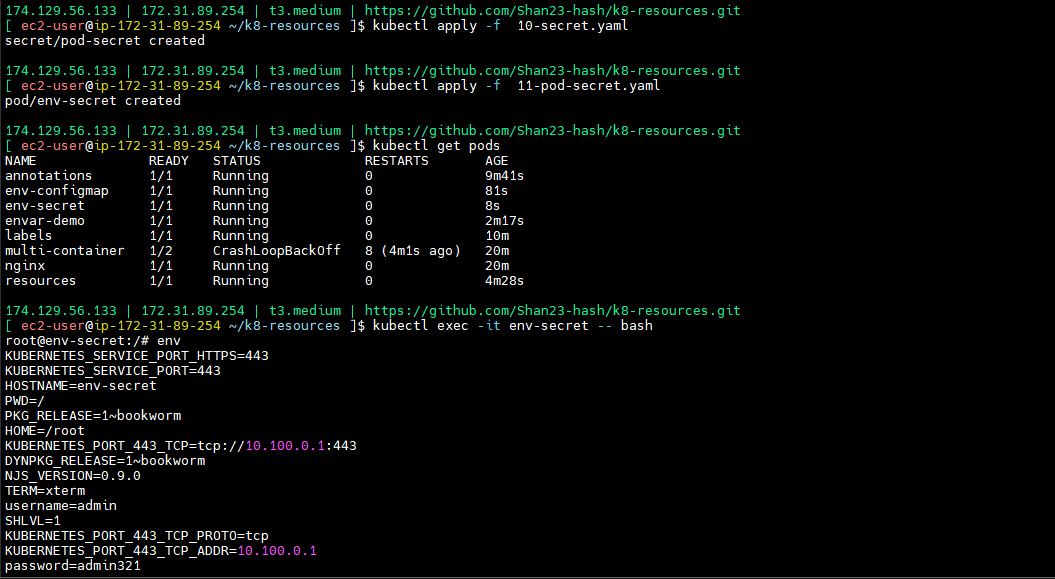
--> **kubectl apply -f 10-secret.yaml**

--> **kubectl apply -f 11-pod-secret.yaml**

**--> kubectl get pods**

**--> kubectl exec -it env-secret -- bash**

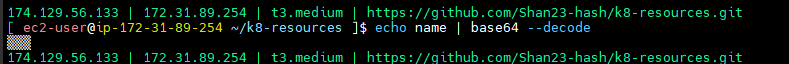
**--> env** -- username and password it will show



--> This is secure

--> anyone this can decode.

--> **echo name | base 64 --decode** -- I will get value



--> encryption - private and public key , who have private key that peoples only can decryption.

--> devops ---> daeavaoapas --> encoding

devops --> fhasghauq9871345908752465fdhgfhghfh --> encryption