**DEPLOY 3 PODS IN k8 AND Create the dashboard**

**/ VISUALISE k8 DASHBOARD**

* Create instances of EC2 on Amazon
  + Specification(Amazon Linux)
  + AMI -Amazon Linux 2 AMI(HVM) -kernel 5.10, SSD Volume Type
  + Instance Type – t2.medium
  + Key pair name – Proceed without a key pair
  + Configure Storage – (1x 20 GiB gp2)
* Connect to your Instance
* Update the packages and install the docker
* Sudo su
* yum update -y (update the package )
* yum install docker -y ( to install docker )
* systemctl start docker ( to start the docker )
* systemctl enable docker( to enable the. Docker)
* Install the Conntrack, minikube,
* yum install conntrack -y ( to install the contrack )
* curl -LO <https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64> ( keep the whole command to get the minikube )
* sudo install minikube linux amd64/usr/local/bin/minikube( keep the whole command to install

minikube)

* /usr/local/bin/minikube start --force --driver=docker ( to start the docker on minikube )
* docker login ( to login with your username and password)
* **Creat a jar with the maven and install GIT and Java**
* cd /opt/( to open the floder of opt)
* yum install maven -y ( to install the maven )
* yum install git -y ( to install the git)
* yum install java -y( to install the java )
* Install KUBECTL and setup
* curl -o kubectl <https://amazon-eks.s3.us-west2.amazonaws.com/1.20.4/2021-04-12/bin/linux/amd64/kubectl> ( executed the command in one line )
* chmod +x ./kubectl ( Makes the kubectl file executable)
* mkdir -p $HOME/bin ( Creates a 'bin' directory in your home folder if it doesn't exist)
* cp ./kubectl $HOME/bin/kubectl ( Copies the kubectl file to the $HOME/bin directory)
* export PATH=$HOME/bin:$PATH ( Temporarily adds $HOME/bin to the PATH environment variable)
* echo 'export PATH=$HOME/bin:$PATH' >> ~/.bashrc (Adds this PATH update to .bashrc for future sessions)
* source $HOME/.bashrc (Reloads the .bashrc file to apply changes immediately)
* kubectl version --short --client ( Verifies the kubectl installation by showing its client version)
* git clone <https://github.com/vivekreddy5959/java-2>( clone the git respiratory )
* The important step now is replacing the user of your docker in three place where it mark as skybule
* mvn clean install -DskipTests( to clean and install the Dskiptest)
* cd shopfront/ ( to open the shopfront folder )
* docker build -t monster123159/shopfront:latest .( to build the image on the shopfront)
* docker push monster123159/shopfront:latest ( to push the image )
* cd .. ( to come back )
* cd productcatalogue/ ( to open the producatalogue/)
* docker build -t monster123159/productcatalogue:latest . ( put the whole command in same line to build the image )
* docker push monster123159/productcatalogue:latest( to push the image )
* cd .. ( to come back )
* cd stockmanager/ ( to open the stockmanger )
* docker build -t monster123159/stockmanager:latest . .( to build the image on the stockmanger)
* docker push praveensingam1994/stockmanager:latest.( to push the image on docker )

A screenshot of a computer

AI-generated content may be incorrect.

* GO TO KUBERNETES FOLDER IN SAME

PROJECT and set-up

* cd .. ( to go back )
* cd Kubernetes( to open the Kubernetes)
* kubectl apply -f shopfront-service.yaml ( to set the shopfront server)
* kubectl apply -f productcatalogue-service.yaml yaml ( to set the productcatalogue server)
* kubectl apply -f stockmanager-service.yaml yaml ( to set the stockmanager server)
* check the kubectl with the update servers
* kubectl get pods(to check the pods)
* kubectl get svc( to check the svc)
* kubectl get nodes( to check the nodes)
* **start** the Kubernetes dashboard in EC2
* /usr/local/bin/minikube dashboard( creat the dashboard with the minikube )
* In Ec2 go to the security group and edit the inbound rules and add then ( all traffic and anywhere and then save )

A screenshot of a computer

AI-generated content may be incorrect.

* open new EC2 and
* kubectl proxy --address='0.0.0.0' --accept-hosts='^\*$'(to allow traffic from the outside)

A screenshot of a computer

AI-generated content may be incorrect.

* you will recive message that the server is starting on 8001

* Getting access to the dashboard
* Whenever you run the dashboard and Kubernetes setup,up you will get a link

A screenshot of a computer

AI-generated content may be incorrect.

* Now edit the URL with IP address of your EC2 instance address
* Now get the ip address IPV4 from the EC2-IP

http://<EC2-IP>:8001/api/v1/namespaces/kubernetes-da

shboard/services/http:kubernetes-dashboard:/proxy/#/pod?namespace=default

A screenshot of a computer

AI-generated content may be incorrect.

* Now, hit the URL and get the Kubernetes dashboard and you can analyze the entire K8

A screenshot of a computer

AI-generated content may be incorrect.

* As per the dashboard, one of the pods is failed ( K8 has auto-healing so wait for it

A screenshot of a computer

AI-generated content may be incorrect.

* Replication of the pods

A screenshot of a computer

AI-generated content may be incorrect.

* Let’s see some more result like service , events , pods and containers

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.

**Conclusion**

The entire process of deploying three pods in Kubernetes on AWS EC2 instances demonstrates a streamlined DevOps pipeline from infrastructure provisioning to application deployment and monitoring. Below are the key takeaways and highlights of the process:

**1. Infrastructure Setup**

* EC2 instances were provisioned using **Amazon Linux 2** with a T2.medium instance type to ensure adequate resources for running Minikube and Kubernetes components.
* Proper setup of Docker, Minikube, and necessary dependencies (conntrack, Maven, Git, Java) was essential to establish the environment for container orchestration.

**2. Application Build and Image Deployment**

* Using Maven for building the Java application ensured a clean and efficient build process.
* Docker images for all services (**shopfront**, **productcatalogue**, and **stockmanager**) were successfully built and pushed to Docker Hub for Kubernetes to pull during deployment.

**3. Kubernetes Deployment**

* Kubernetes manifests, including YAML configurations for services and deployments, were applied to launch the respective pods.
* The Kubernetes CLI (kubectl) was used to verify the health of the pods, services, and nodes, confirming that the deployment was functional.

**4. Dashboard Setup and Visualization**

* The Kubernetes Dashboard was deployed using Minikube, allowing for real-time monitoring of the cluster.
* Adjusting the EC2 security group inbound rules and running the kubectl proxy command enabled external access to the dashboard.
* Accessing the dashboard via the EC2 public IP provided an overview of all cluster resources, including services, events, pods, and containers.

**5. Observations and Auto-Healing**

* During the deployment, one of the pods failed, which showcased Kubernetes’ **self-healing capabilities**. This feature allowed the cluster to automatically manage and recover the failed pod.

**6. Scaling and Replication**

* Kubernetes demonstrated its robust scaling and replication features, enabling the creation of highly available and resilient services. Commands like kubectl get pods, kubectl get svc, and kubectl get nodes were instrumental in verifying the state of the cluster.

**Key Lessons and Benefits**

* **Automation & Consistency**: Kubernetes simplifies deployment with declarative configurations, ensuring consistent application behavior.
* **Scalability**: By deploying multiple replicas, the cluster can handle increased traffic and provide failover capabilities.
* **Visibility & Monitoring**: The Kubernetes Dashboard provides a centralized view of cluster resources, making debugging and performance monitoring straightforward.
* **Resilience**: The auto-healing mechanism ensures high availability, even when individual pods fail.

**Key Achievements**

1. Successfully deployed three Kubernetes pods (**shopfront**, **productcatalogue**, and **stockmanager**) with Docker images built and pushed to Docker Hub, ensuring smooth application lifecycle management.
2. Configured and accessed the **Kubernetes Dashboard**, enabling real-time monitoring and visualization of cluster resources, enhancing troubleshooting and performance analysis.
3. Demonstrated **Kubernetes auto-healing** by observing the recovery of failed pods, showcasing the platform's resilience and self-management capabilities.
4. Effectively scaled services through replication and managed container orchestration on a Minikube cluster deployed on AWS EC2 instances.
5. **Achieved end-to-end automation** by integrating Maven for building the Java application and Docker for containerizing, resulting in faster, more efficient deployments.