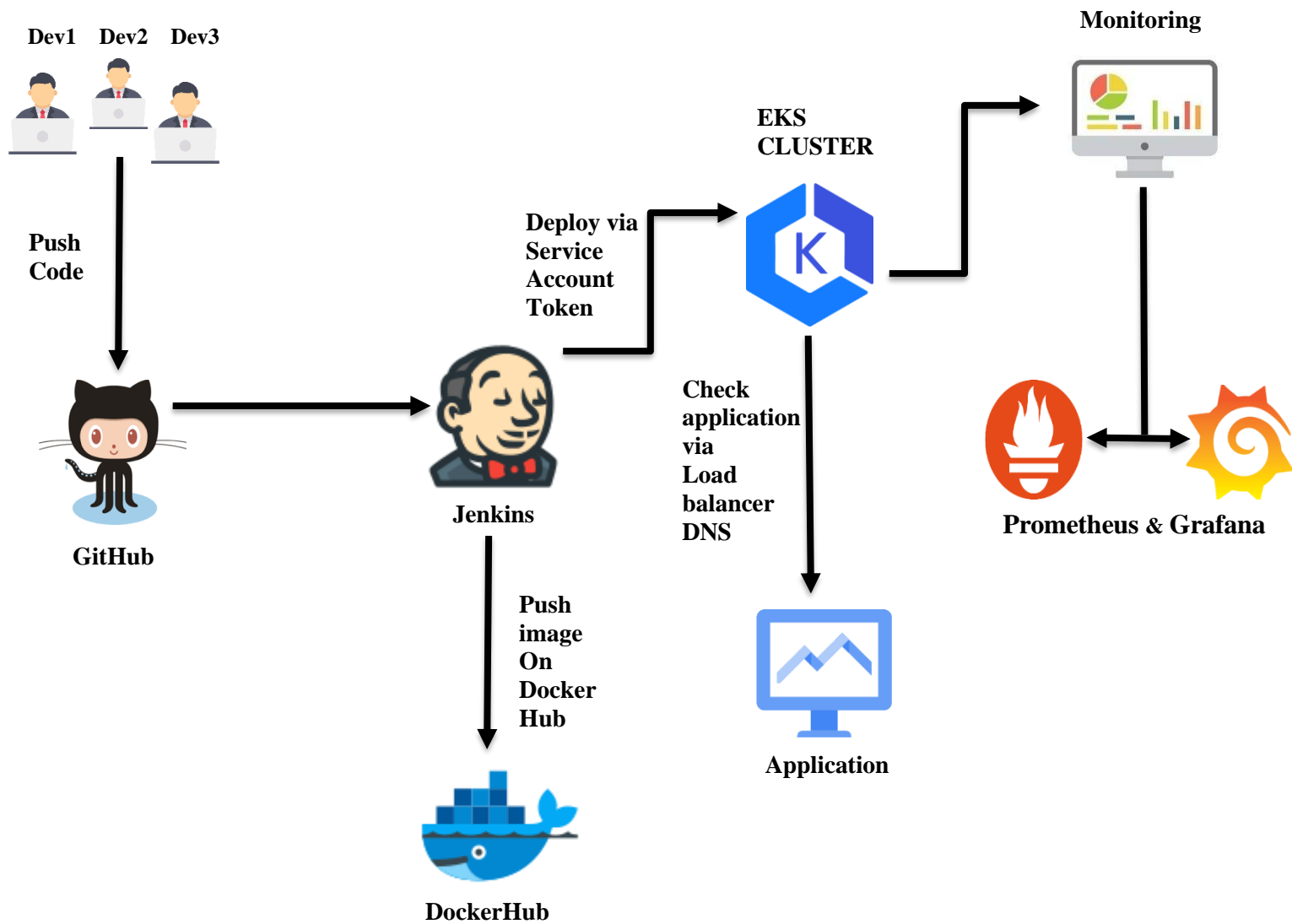


PROJECT

Title -: E-Commerce Web Deployment: Multi-Tier Application with AWS EKS, Jenkins, Docker, and Kubernetes

TOOL USE

- **AWS Console**: Used for managing servers and resources.
- **AWS EKS (Elastic Kubernetes Service)**: For managing Kubernetes clusters on AWS.
- **IAM**: Manages user permissions and access to AWS resources.
- **Jenkins**: For automating the CI/CD pipeline and deploying applications.
- **GitHub**: Hosting code repositories and version control.
- **Docker**: For containerizing applications to simplify deployment and scaling.
- **Docker Hub**: Platform to store and manage Docker images.
- **kubectl**: Command-line tool to interact with Kubernetes clusters
- **eksctl**: For creating and managing EKS clusters on AWS.
- **Kubernetes**: Orchestration platform for automating deployment, scaling, and management of containerized applications.
- **PROMETHEUS & GRAFANA**: Tools for monitoring and visualizing system metrics.



CI/CD Workflow for OnlineBoutique Deployment

STEP-1: Setting Up Your EC2 Instance

Launch two EC2 Instance:

First server (jenkins)

- **Storage:** Attach a 30 GB EBS volume.
- **Instance Type:** t2. large for better performance with 2 vCPUs and 8 GB RAM.
- **IAM Role:** Assign a role with full access to manage AWS resources.

Step 2: Install AWS CLI, kubectl, and eksctl

AWS CLI:

- Installation:

```
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
sudo apt install unzip
unzip awscliv2.zip
sudo ./aws/install
```

kubectl:

- Installation:

```
curl -o kubectl https://amazon-eks.s3.us-west-2.amazonaws.com/1.19.6/2021-01-05/bin/linux/amd64/kubectl
chmod +x ./kubectl
sudo mv ./kubectl /usr/local/bin
kubectl version --short --client
```

eksctl:

- Installation:

```
curl --silent --location
"https://github.com/weaveworks/eksctl/releases/latest/download/eksctl_$(uname -
s)_amd64.tar.gz" | tar xz -C /tmp
sudo mv /tmp/eksctl /usr/local/bin
eksctl version
```

Step 3: Configure IAM User Credentials

- Run **aws configure** and enter:
 - AWS Access Key ID -
 - AWS Secret Access Key -
 - Default region name - **us-east-1**
 - Default output format –

Note - Enter only the region name as us-east-1. Not required others details because access is already provided through the IAM role

Step 4: Create Cluster -

- **Cluster Creation:**

```
eksctl create cluster --name=EKS-1 --region= us-east-1 --zones= us-east-1a, us-east-1b  
--without-nodesgroup
```

- **OIDC Provider:**

```
eksctl utils associate-iam-oidc-provider --region us-east-1 --cluster EKS-1 --approve
```

- **Node Group Creation:**

```
eksctl create nodesgroup --cluster=EKS-1 --region= us-east-1 --name=node2 --node-  
type=t3.medium --nodes=3 --nodes-min=2 --nodes-max=4 --node-volume-size=20 --ssh-  
access --ssh-public-key=DevOps --managed --asg-access --external-dns-access --full-ecr-  
access --appmesh-access --alb-ingress-access
```

Step 5: Install Jenkins & Docker

Install Java:

```
sudo apt install openjdk-17-jre-headless -y
```

Install Jenkins:

```
sudo wget -O /usr/share/keyrings/jenkins-keyring.asc https://pkg.jenkins.io/debian-  
stable/jenkins.io-2023.key  
echo "deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc]  
https://pkg.jenkins.io/debian-stable binary/" | sudo tee /etc/apt/sources.list.d/jenkins.list >  
/dev/null  
sudo apt-get update  
sudo apt-get install Jenkins -y
```

Install Docker:

```
sudo apt install docker.io -y
```

```
sudo chmod 777 /var/run/docker.sock
```

Configure Jenkins for Docker:

- In Jenkins Dashboard: Manage Jenkins -> Tools -> Docker installations -> Name: docker -> Install automatically -> Docker version: latest.

Step 6: Download Plugins

- Required Plugins:
 - Docker
 - Docker Pipeline
 - Kubernetes
 - Kubernetes CLI

Step 7: Add Credentials for Docker & GitHub

Add DockerHub Credentials:

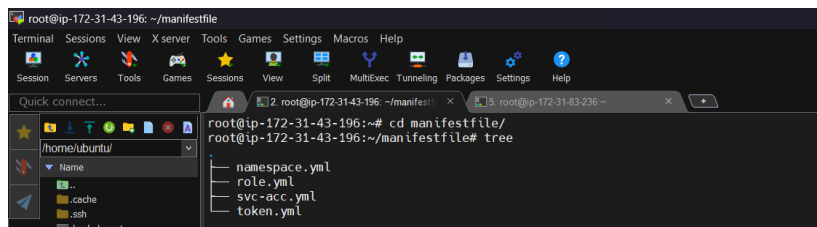
- Dashboard -> Manage Jenkins -> Credentials -> System -> Global credentials (unrestricted).
- Add: Username: bhagyeshpatil99, Password: [Docker password], ID: docker.

Add GitHub Credentials:

Repo - <https://github.com/bhagyesh98/microservices-project.git> (take code form github)

- Add: Username: bhagyesh98, Password: [GitHub token].

Step 8: Create Service Account, Role, and Role Binding for webapps Namespace



vim namespace.yml

```
apiVersion: v1
kind: Namespace
metadata:
  name: webapps
```

Create for Service Account:

Vim svc-acc.yml

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: jenkins
  namespace: webapps
```

Create Role and Role Binding:

role.yml:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: app-role
  namespace: webapps
rules:
- apiGroups:
  - ""
  - apps
  - autoscaling
  - batch
  - extensions
  - policy
  - rbac.authorization.k8s.io
resources:
- pods
- componentstatuses
- configmaps
- daemonsets
- deployments
- events
- endpoints
- horizontalpodautoscalers
- ingress
- jobs
- limitranges
- namespaces
- nodes
- pods
- persistentvolumes
- persistentvolumeclaims
- resourcequotas
- replicaset
- replicationcontrollers
- serviceaccounts
- services
verbs: ["get", "list", "watch", "create", "update", "patch", "delete"]
---
```

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: app-rolebinding
  namespace: webapps
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: app-role
subjects:
- namespace: webapps
  kind: ServiceAccount
  name: jenkins
```

Generate Token:

Vim token.yml

```
apiVersion: v1
kind: Secret
type: kubernetes.io/service-account-token
metadata:
  name: bhagyesh
  namespace: webapps
  annotations:
    kubernetes.io/service-account.name: Jenkins
```

Get Token:

- **Command:**

```
kubectrl describe secret bhagyesh -n webapps
```

Step 9: Add Token to Jenkins Credentials

- Add Token:
 - Dashboard -> Manage Jenkins -> Credentials -> System -> Global credentials (unrestricted).
 - Add: Secret: [Generated token], ID: k8s-secret.

Step 10: Change DockerHub ID and Token in All Services and Main Branch

- Update DockerHub ID: Update your DockerHub credentials in your Jenkins pipelines and services.
- Update Token and Endpoint: Ensure the token and endpoint are correctly set in the main branch of your project repository.

Step 11: Monitoring Prometheus with Grafana

1. Install Components:
 - Install Grafana, Prometheus, and Node Exporter on the monitoring server.
2. Access Grafana:
 - Port: 3000, Username/Password: admin/admin.
3. Connect Prometheus to Grafana:
 - Navigate to Data Sources → Add Prometheus → Enter Prometheus URL → Save & Test.
4. Import Dashboard:
 - Click + → Import → Enter Dashboard ID 1860 → Load → Select Prometheus → Import.

Snapshot:

EKS CLUSTER:

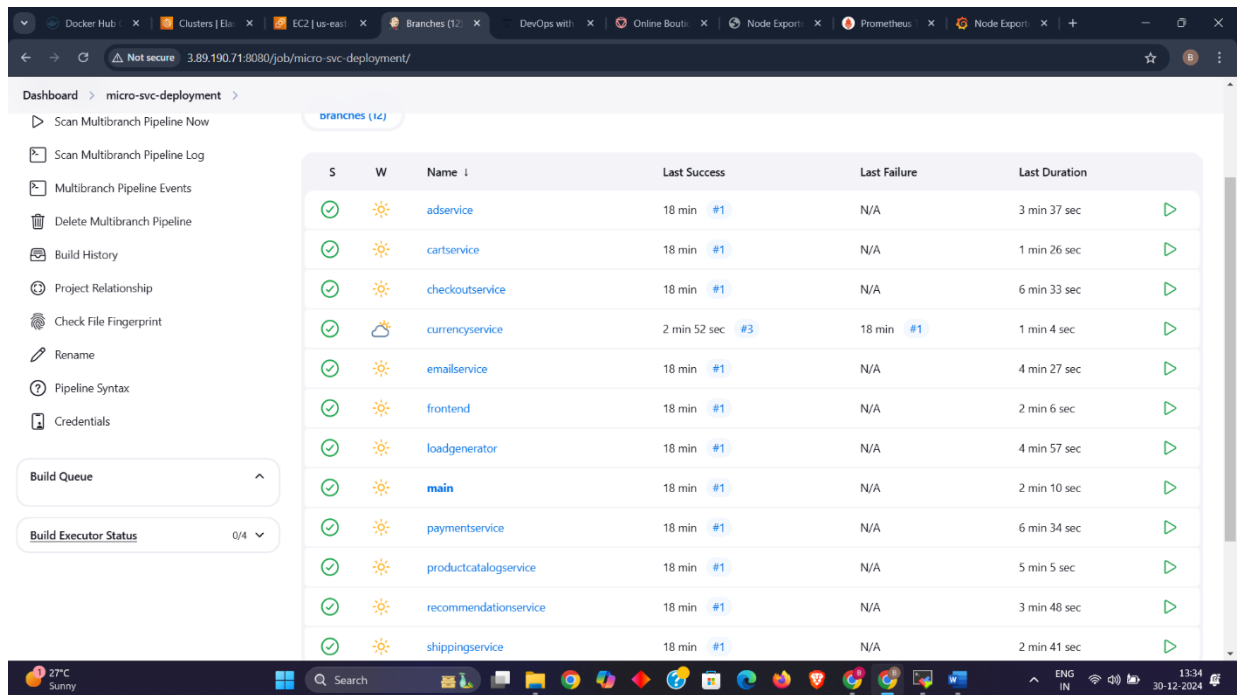
The screenshot displays the AWS Management Console interface. The top navigation bar shows the AWS logo, a search bar, and the user's profile (bhagyesh99) in N. Virginia. The left sidebar contains the 'VPC dashboard' and a 'Virtual private cloud' section with links to Subnets, Route tables, Internet gateways, Egress-only internet gateways, Carrier gateways, DHCP option sets, Elastic IPs, Managed prefix lists, NAT gateways, and Peering connections. The main content area is titled 'Your VPCs (1/2)' and shows a table of VPCs. The table has columns for Name, VPC ID, State, Block Public..., IPv4 CIDR, and IPv6 CIDR. Two VPCs are listed: 'vpc-070c7cc81b5a19907' (Available, 172.31.0.0/16) and 'eksctl-EKS-1-cluster/VPC' (Available, 192.168.0.0/16). Below the table is a 'Resource map' section showing the network architecture. It includes 'Subnets (4)' with 'us-east-1a' and 'us-east-1b' subnets, 'Route tables (4)' with private and public route tables, and 'Network connections (2)' with an Internet Gateway and a NAT Gateway. Arrows indicate the connections between these resources.

Name	VPC ID	State	Block Public...	IPv4 CIDR	IPv6 CIDR
-	vpc-070c7cc81b5a19907	Available	Off	172.31.0.0/16	-
eksctl-EKS-1-cluster/VPC	vpc-03e16b097cffe69d5	Available	Off	192.168.0.0/16	-

EKS Nodes and Servers:

```
root@ip-172-31-43-196:~/manifestfile# cat namespace.yml
apiVersion: v1
kind: Namespace
metadata:
  name: webapps
root@ip-172-31-43-196:~/manifestfile# cat role.yml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: app-role
  namespace: webapps
rules:
- apiGroups:
  - ""
  - apps
  - autoscaling
  - batch
  - extensions
  - policy
  - rbac.authorization.k8s.io
resources:
  - pods
  - componentstatuses
  - configmaps
  - daemonsets
  - deployments
  - events
  - endpoints
  - horizontalpodautoscalers
  - ingress
  - jobs
  - limitranges
  - namespaces
  - nodes
  - persistentvolumes
  - persistentvolumeclaims
  - resourcequotas
  - replicaset
  - replicationcontrollers
  - serviceaccounts
  - services
verbs: ["get", "list", "watch", "create", "update", "patch", "delete"]
```

ROLE CREATION:

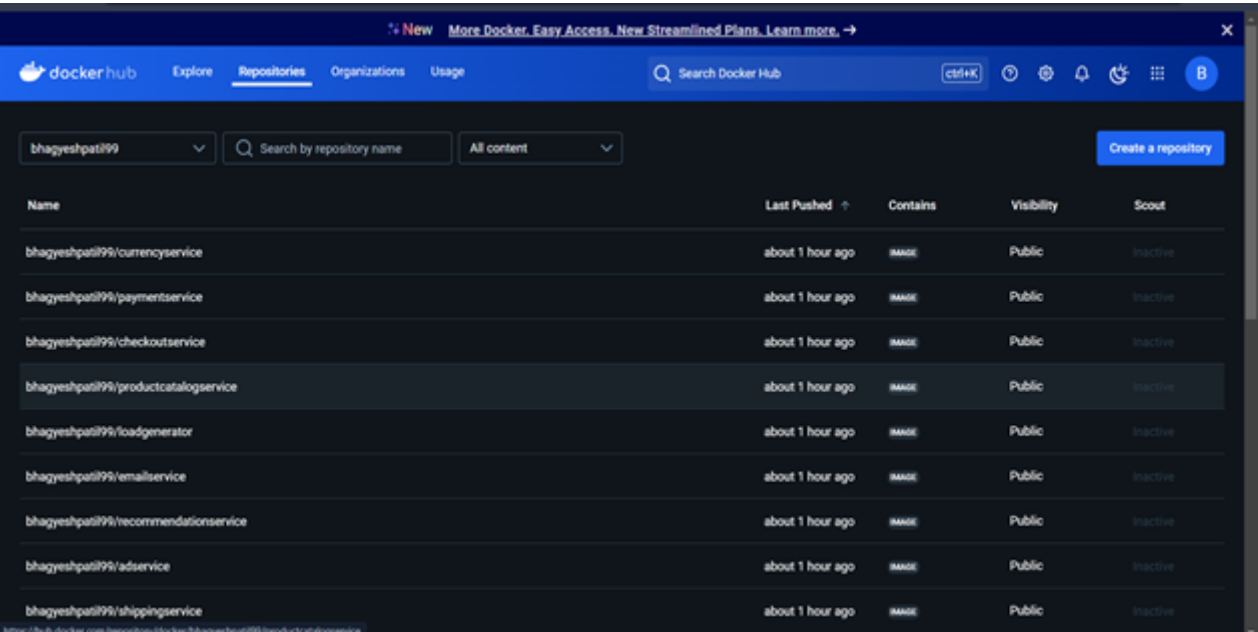


The screenshot shows the Jenkins Dashboard for a pipeline named 'micro-svc-deployment'. The dashboard includes a sidebar with navigation options like 'Scan Multibranch Pipeline Now', 'Build History', and 'Project Relationship'. The main area displays a table of pipeline runs with columns for status, name, last success, last failure, and last duration. All runs are successful, indicated by green checkmarks.

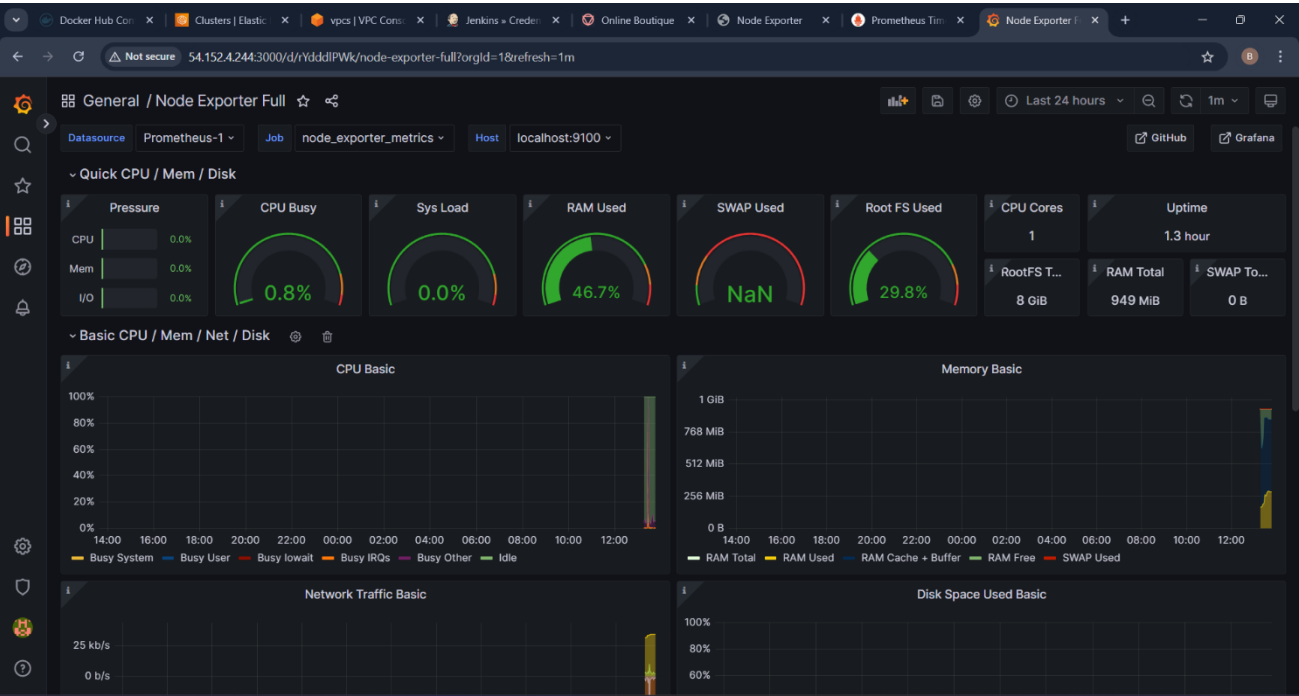
S	W	Name	Last Success	Last Failure	Last Duration
✓	☀	adservice	18 min #1	N/A	3 min 37 sec
✓	☀	cartservice	18 min #1	N/A	1 min 26 sec
✓	☀	checkoutservice	18 min #1	N/A	6 min 33 sec
✓	☁	currencyservice	2 min 52 sec #3	18 min #1	1 min 4 sec
✓	☀	emailservice	18 min #1	N/A	4 min 27 sec
✓	☀	frontend	18 min #1	N/A	2 min 6 sec
✓	☀	loadgenerator	18 min #1	N/A	4 min 57 sec
✓	☀	main	18 min #1	N/A	2 min 10 sec
✓	☀	paymentservice	18 min #1	N/A	6 min 34 sec
✓	☀	productcatalogservice	18 min #1	N/A	5 min 5 sec
✓	☀	recommendationservice	18 min #1	N/A	3 min 48 sec
✓	☀	shippingservice	18 min #1	N/A	2 min 41 sec

THE PIPELINE CODE IS TAKEN FROM GITHUB IN THE JENKINSFILE:

DOCKERHUB IAMGE:



Monitoring Prometheus with Grafana:



WEBSITE OUTPUT:

