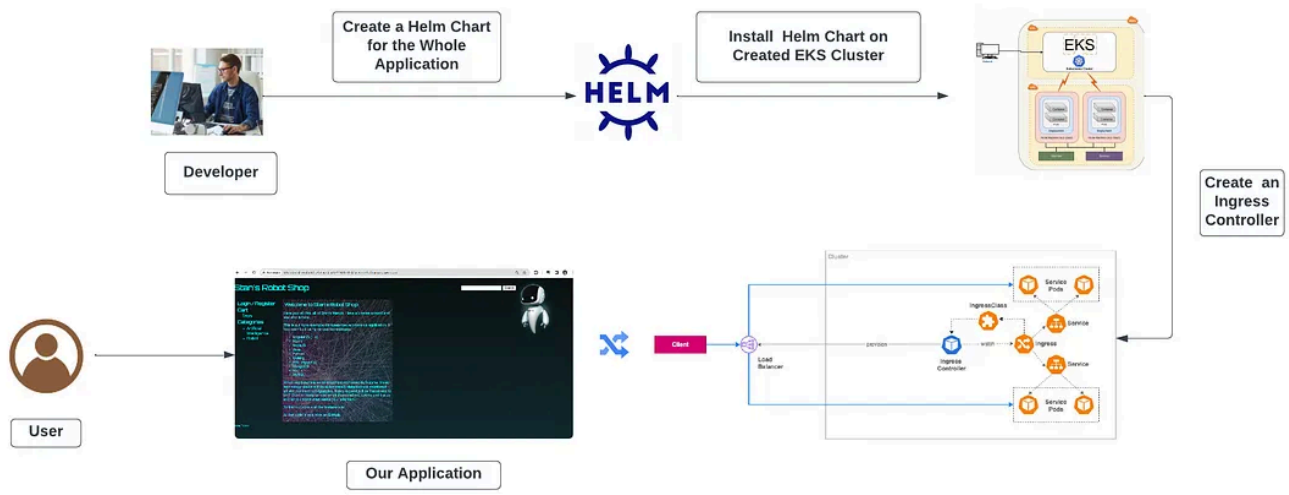
Deploy an E -Commerce Three Tier application on AWS EKS with Helm

6 min read · Feb 5, 2024

Sreedhar Reddy Madithati Follow



Sample Three Tier Application

Introduction:

In the dynamic landscape of software development, architects and developers constantly seek robust design patterns that ensure scalability, maintainability, and efficient resource utilization. One such time-tested approach is the 3-tier architecture, a well-structured model that divides an application into three interconnected layers. This architectural style has been a cornerstone for building scalable and resilient applications for decades.

Understanding the Basics:

The 3-tier architecture is composed of three primary layers, each with distinct responsibilities:

1. **Presentation Layer:**

- Also known as the user interface layer, this tier is responsible for interacting with end-users.
- It encompasses the user interface components, such as web pages, mobile apps, or any other interface through which users interact with the application.
- The goal is to provide a seamless and intuitive user experience while keeping the presentation logic separate from the business logic.

2. **Application (or Business Logic) Layer:**

- Positioned between the presentation and data layers, the application layer contains the business logic that processes and manages user requests.
- It acts as the brain of the application, handling tasks such as data validation, business rules implementation, and decision-making.
- Separating the business logic from the presentation layer promotes code reusability, maintainability, and adaptability to changes.

3. **Data Layer:**

- The data layer is responsible for managing and storing the application's data.
- It includes databases, data warehouses, or any other data storage solutions.
- This layer ensures data integrity, security, and efficient data retrieval for the application.
- By isolating data-related operations, developers can optimize data access and storage mechanisms independently of the rest of the application.

Benefits of 3-Tier Architecture:

1. **Scalability:**

- The modular nature of 3-tier architecture allows for independent scaling of each layer.
- This enables efficient resource allocation, ensuring that specific components can be scaled based on demand without affecting the entire application.

2. **Maintainability:**

- With clear separation of concerns, developers can make changes to one layer without impacting others.
- This facilitates easier debugging, updates, and maintenance, as modifications can be confined to the relevant layer.

3. **Flexibility and Adaptability:**

- The architecture accommodates technology changes and updates without disrupting the entire system.
- New technologies can be integrated into specific layers, allowing the application to evolve over time.

Prerequisites:

1. kubectl — A command line tool for working with Kubernetes clusters. For more information, see Installing or updating kubectl. https://docs.aws.amazon.com/eks/latest/userguide/install-kubectl.html
2. eksctl — A command line tool for working with EKS clusters that automates many individual tasks. For more information, see Installing or updating. https://docs.aws.amazon.com/eks/latest/userguide/eksctl.html
3. AWS CLI — A command line tool for working with AWS services, including Amazon EKS. For more information, see Installing, updating, and uninstalling the AWS CLI https://docs.aws.amazon.com/cli/latest/userguide/cli-chap-install.html in the AWS Command Line Interface User Guide.
4. After installing the AWS CLI, I recommend that you also configure it. For more information, see Quick configuration with aws configure in the AWS Command Line Interface User Guide. https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-quickstart.html#cli-configure-quickstart-config

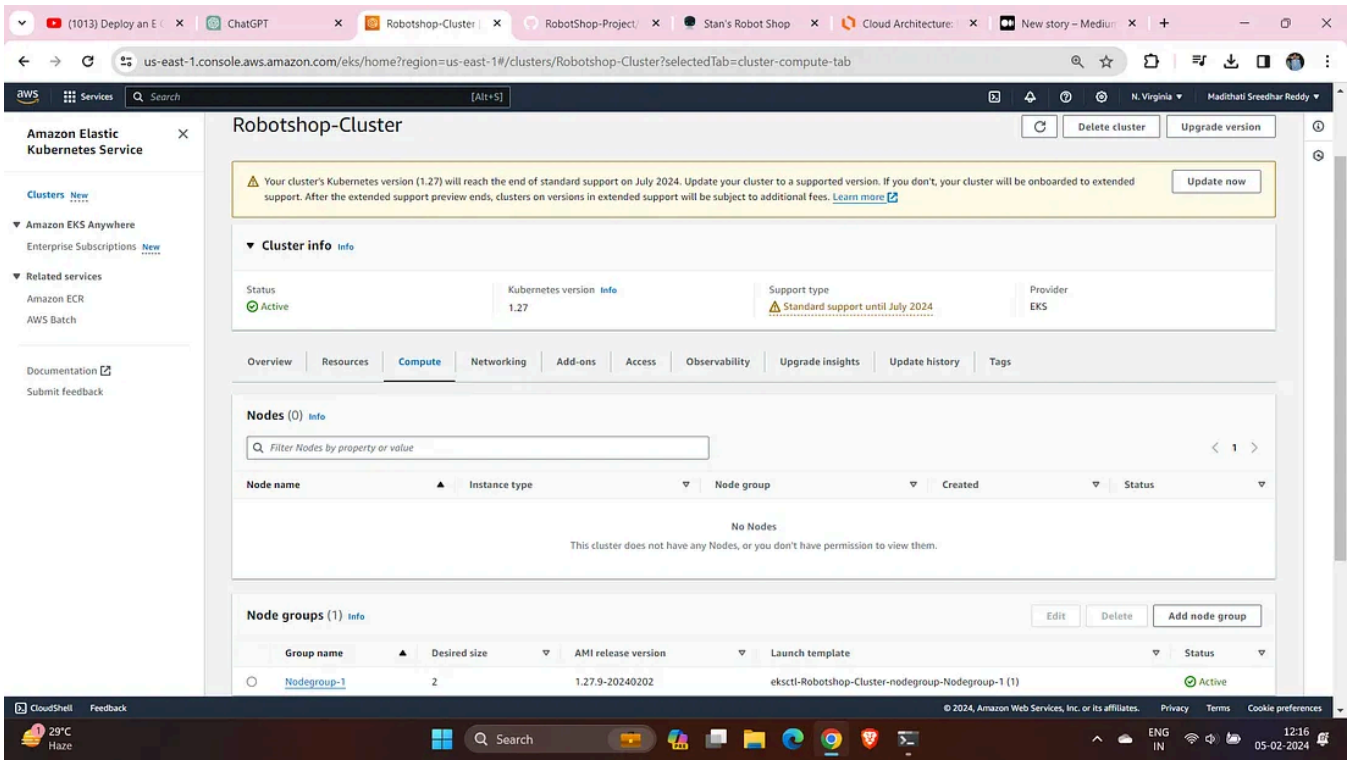
Steps:

Step-1: Create an EKS Cluster

```
eksctl create cluster \
  --name your-cluster-name \
```

```
    --region your-region \
    --nodegroup-name your-nodegroup-name \
    --node-type t3.small \
    --nodes 2 \
    --nodes-min 1 \
    --nodes-max 3
```

This will create an EKS Cluster along with the node-group



Step-2: Configure IAM OIDC provider

1. Export Cluster Name and assign oidc_id

```
export cluster_name=<CLUSTER-NAME>
oidc_id=$(aws eks describe-cluster --name $cluster_name --query "cluster.identi
```

2. Check if there is an IAM OIDC provider configured already

```
aws iam list-open-id-connect-providers | grep $oidc_id | cut -d "/" -f4
```

3. If not, run the below command.

```
eksctl utils associate-iam-oidc-provider --cluster $cluster_name --approve
```

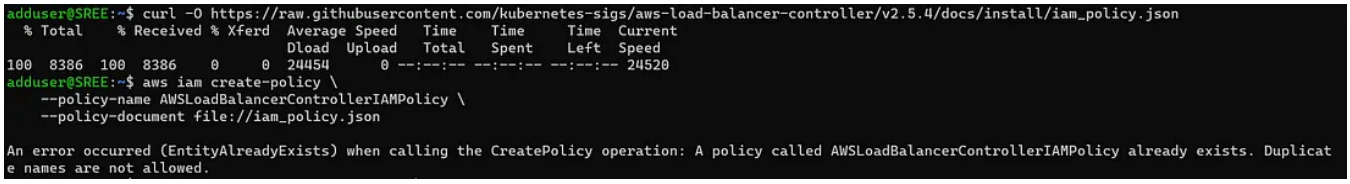
Step-3: Setup ALB Add-On

1. Download IAM policy.

```
curl -O https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-cor
```

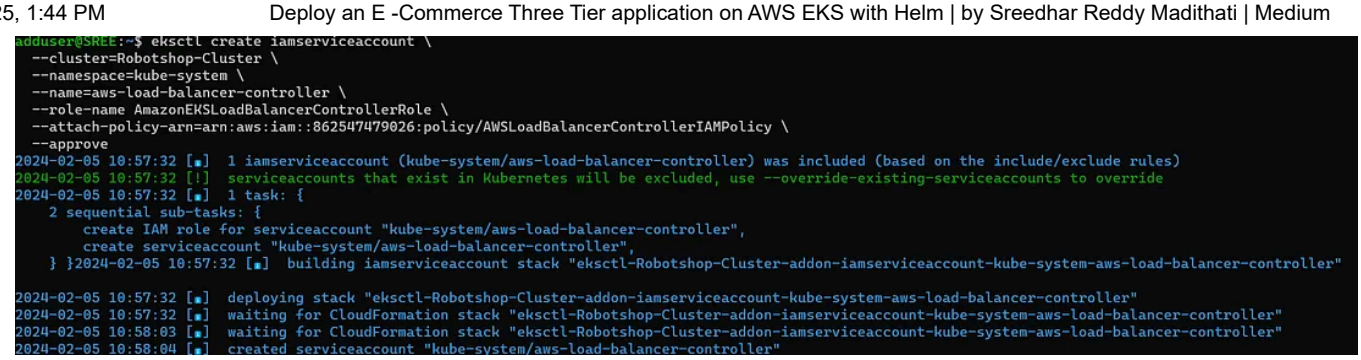
2. Create IAM Policy.

```
aws iam create-policy \
    --policy-name AWSLoadBalancerControllerIAMPolicy \
    --policy-document file://iam_policy.json
```



3. Create IAM Role.

```
eksctl create iamserviceaccount \
  --cluster=<your-cluster-name> \
  --namespace=kube-system \
  --name=aws-load-balancer-controller \
  --role-name AmazonEKSLoadBalancerControllerRole \
  --attach-policy-arn=arn:aws:iam::<your-aws-account-id>:policy/AWSLoadBalancer
  --approve
```



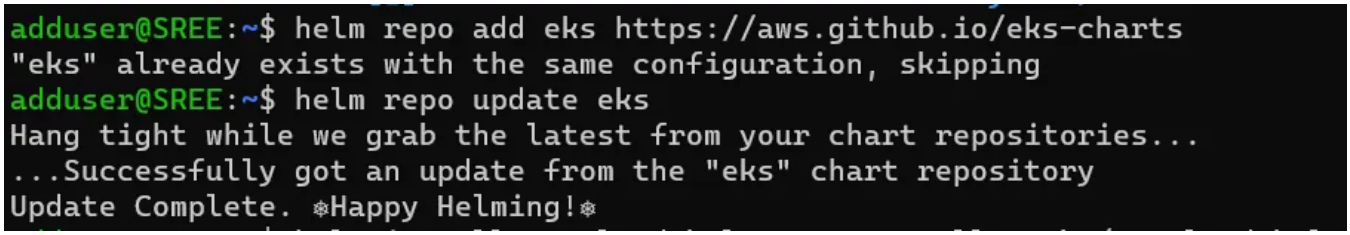
Step-4: Deploy ALB controller

1. Add helm repo.

```
helm repo add eks https://aws.github.io/eks-charts
```

2. Update the repo.

```
helm repo update eks
```



3. Install the chart.

```
helm install aws-load-balancer-controller eks/aws-load-balancer-controller -n k
```

4. Verify that the deployments are running.

```
kubectl get deployment -n kube-system aws-load-balancer-controller
```

```
adduser@SREE:~$  helm install aws-load-balancer-controller eks/aws-load-balancer-controller -n kube-system --set clusterName=Robotshop-Cluster --set service
Account.create=false --set serviceAccount.name=aws-load-balancer-controller --set region=us-east-1 --set vpcId=vpc-0bee82e8dbe06b03c
NAME: aws-load-balancer-controller
LAST DEPLOYED: Mon Feb  5 11:04:41 2024
NAMESPACE: kube-system
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
AWS Load Balancer controller installed!
adduser@SREE:~$ kubectl get deployment -n kube-system aws-load-balancer-controller
NAME                           READY   UP-TO-DATE   AVAILABLE   AGE
aws-load-balancer-controller   2/2     2            2           46s
```

Step-5: EBS CSI Plugin configuration.

1. The Amazon EBS CSI plugin requires IAM permissions to make calls to AWS APIs on your behalf.

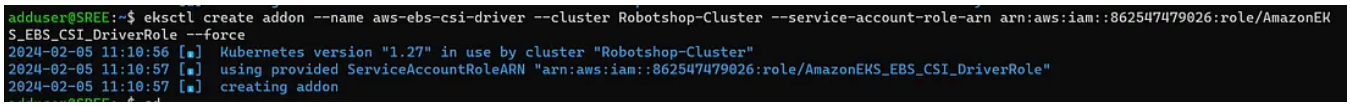
2. Create an IAM role and attach a policy. AWS maintains an AWS managed policy or you can create your own custom policy. You can create an IAM role and attach the AWS managed policy with the following command. Replace my-cluster with the name of your cluster. The command deploys an AWS CloudFormation stack that creates an IAM role and attaches the IAM policy to it.

```
eksctl create iamserviceaccount \
    --name ebs-csi-controller-sa \
    --namespace kube-system \
    --cluster <YOUR-CLUSTER-NAME> \
    --role-name AmazonEKS_EBS_CSI_DriverRole \
    --role-only \
    --attach-policy-arn arn:aws:iam::aws:policy/service-role/AmazonEBSCSIDriver
    --approve
```

```
adduser@SREE:~$ eksctl create iamserviceaccount \
    --name ebs-csi-controller-sa \
    --namespace kube-system \
    --cluster Robotshop-Cluster \
    --role-name AmazonEKS_EBS_CSI_DriverRole \
    --role-only \
    --attach-policy-arn arn:aws:iam::aws:policy/service-role/AmazonEBSCSIDriverPolicy \
    --approve
2024-02-05 11:08:57 [ℹ]  1 existing iamserviceaccount(s) (kube-system/aws-load-balancer-controller) will be excluded
2024-02-05 11:08:57 [ℹ]  1 iamserviceaccount (kube-system/ebs-csi-controller-sa) was included (based on the include/exclude rules)
2024-02-05 11:08:57 [!]  serviceaccounts in Kubernetes will not be created or modified, since the option --role-only is used
2024-02-05 11:08:57 [ℹ]  1 task: { create IAM role for serviceaccount "kube-system/ebs-csi-controller-sa" }
2024-02-05 11:08:57 [ℹ]  building iamserviceaccount stack "eksctl-Robotshop-Cluster-addon-iamserviceaccount-kube-system-ebs-csi-controller-sa"
2024-02-05 11:08:57 [ℹ]  deploying stack "eksctl-Robotshop-Cluster-addon-iamserviceaccount-kube-system-ebs-csi-controller-sa"
2024-02-05 11:08:58 [ℹ]  waiting for CloudFormation stack "eksctl-Robotshop-Cluster-addon-iamserviceaccount-kube-system-ebs-csi-controller-sa"
2024-02-05 11:09:29 [ℹ]  waiting for CloudFormation stack "eksctl-Robotshop-Cluster-addon-iamserviceaccount-kube-system-ebs-csi-controller-sa"
```

3. Run the following command. Replace with the name of your cluster, with your account ID.

```
eksctl create addon --name aws-ebs-csi-driver --cluster <YOUR-CLUSTER-NAME> --s
```



Step-6: Install the Helm Chart:

1. Initially Clone the GitHub Repo:

Github URL :https://github.com/uniquesreedhar/RobotShop-Project.git

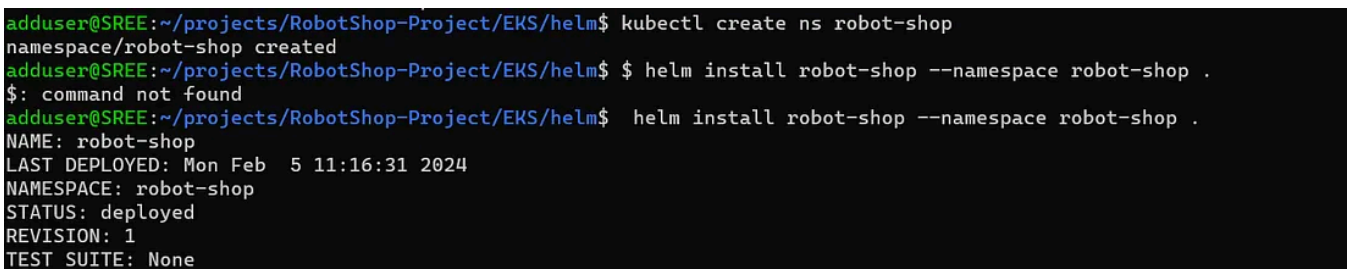
```
git clone https://github.com/uniquesreedhar/RobotShop-Project.git
```

2. Then Navigate to the path where chart.yaml exists

```
cd RobotShop-Project/EKS/helm
```

3. Create a namespace and then install the helm chart.

```
kubectl create ns robot-shop
helm install robot-shop --namespace robot-shop .
```



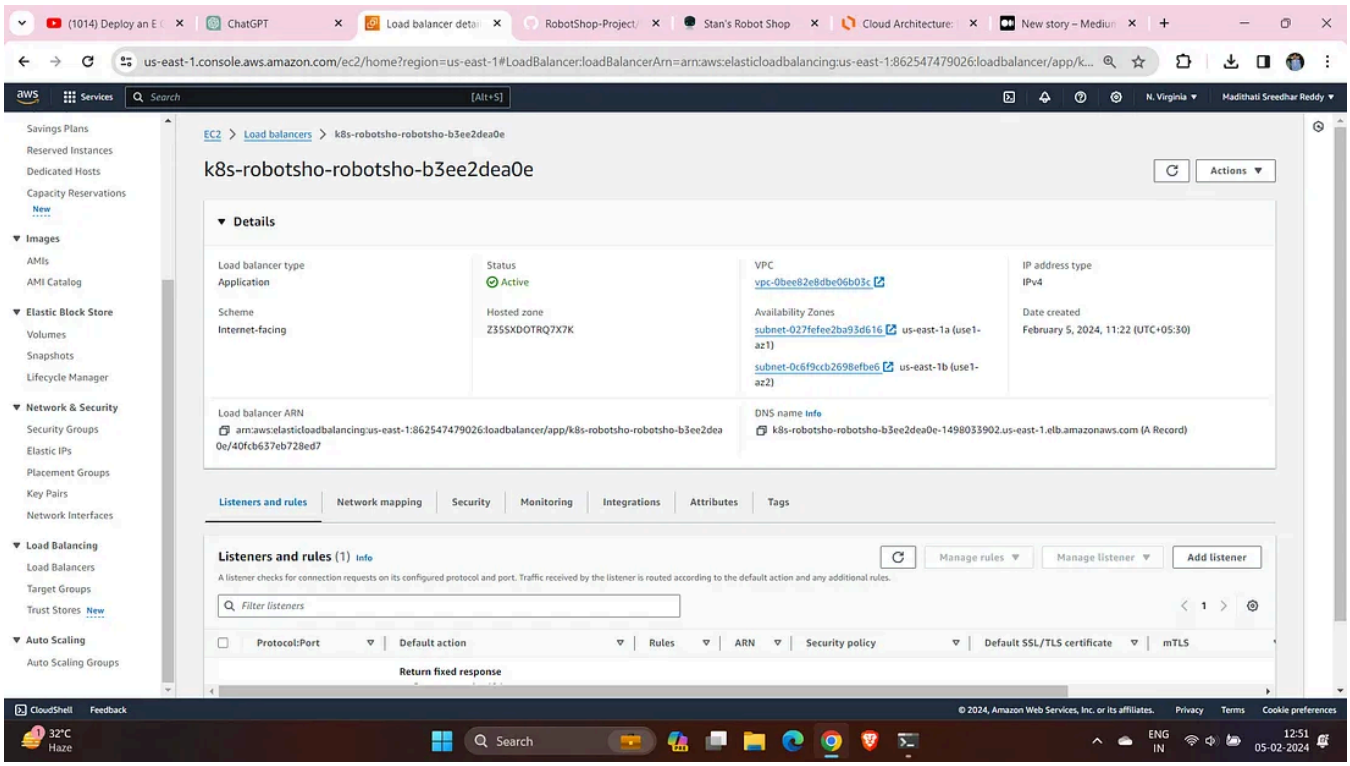
4. Ensure all the pods are running if not troubleshoot the issues.

```
adduser@SREE:~/projects/RobotShop-Project/EKS/helm$ kubectl get pods -n robot-shop
NAME                          READY   STATUS    RESTARTS   AGE
cart-dd947f945-xsnks          1/1     Running   0          101m
catalogue-7fd84d6c48-trvdm    1/1     Running   0          101m
dispatch-66cb67494d-6j5dp     1/1     Running   0          101m
mongodb-db95c57c5-q99qf       1/1     Running   0          101m
mysql-8cbc4749d-ptfbv         1/1     Running   0          101m
payment-85f9dcf964-7jdq8      1/1     Running   0          101m
rabbitmq-79488b58b5-hmbfr     1/1     Running   0          101m
ratings-5c79749c7b-rmlvz      1/1     Running   0          101m
redis-0                       1/1     Running   0          101m
shipping-75f7cf6b6d-v4phr     1/1     Running   0          101m
user-59745f7ccb-cvtrt         1/1     Running   0          101m
web-656685795d-wn6jx          1/1     Running   0          101m
adduser@SREE:~/projects/RobotShop-Project/EKS/helm$
```

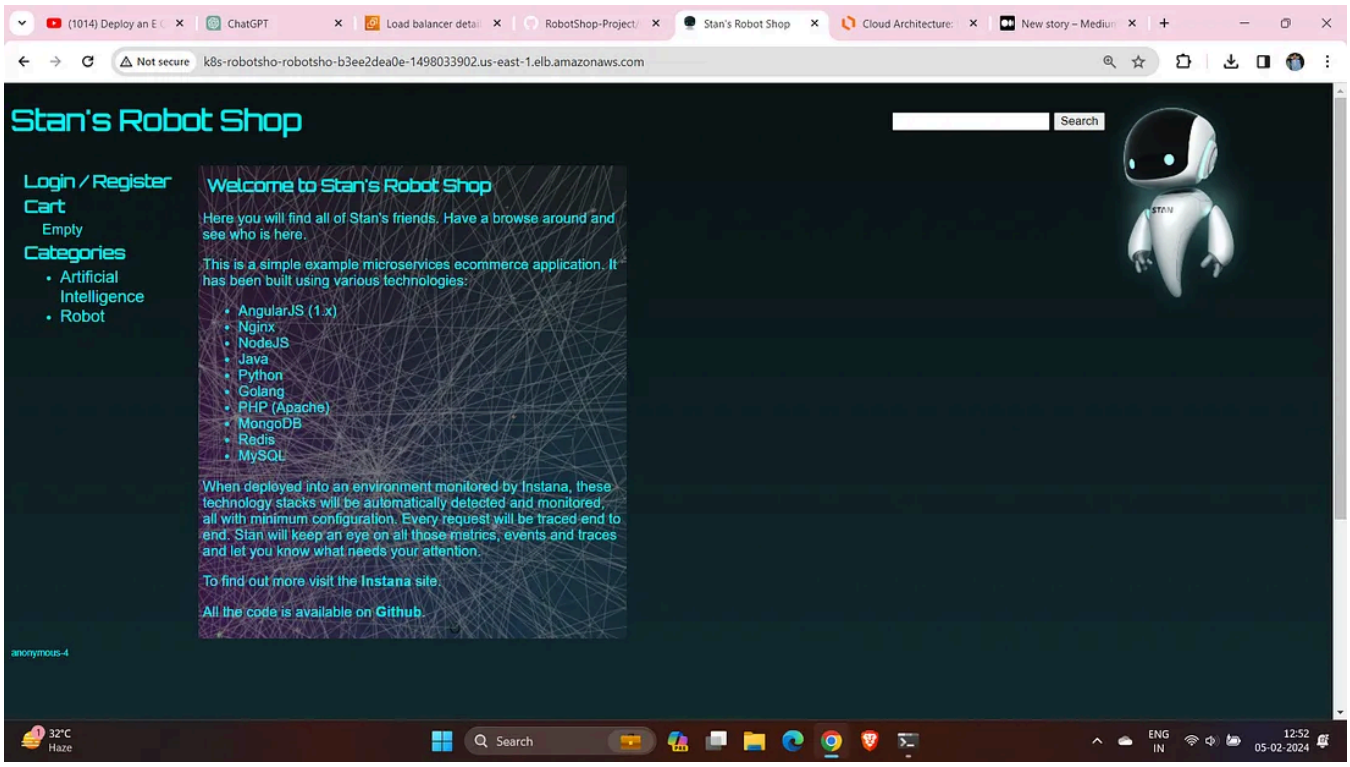
Step 7: Create Ingress

```
cd /RobotShop-Project/EKS/helm
 kubectl apply -f ingress.yaml
```

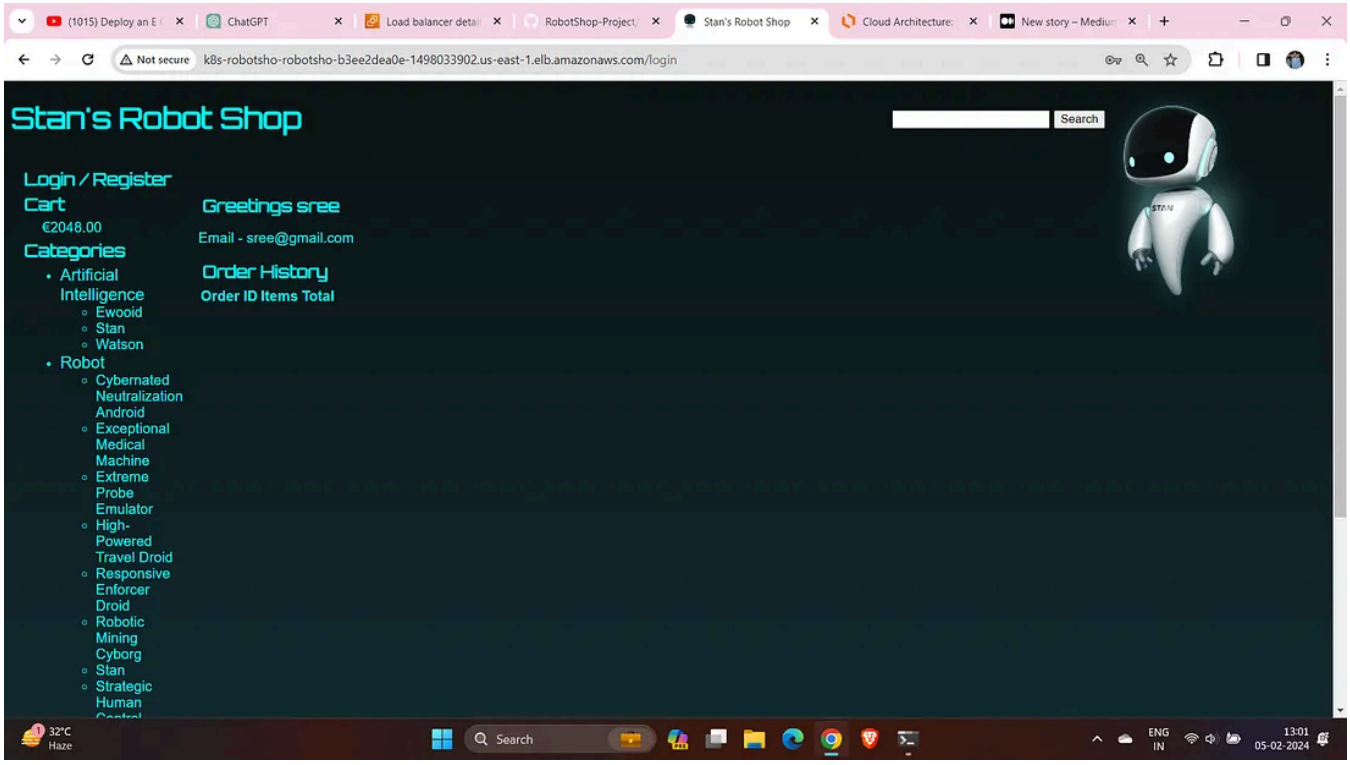
This will create a Load Balancer on the AWS console.



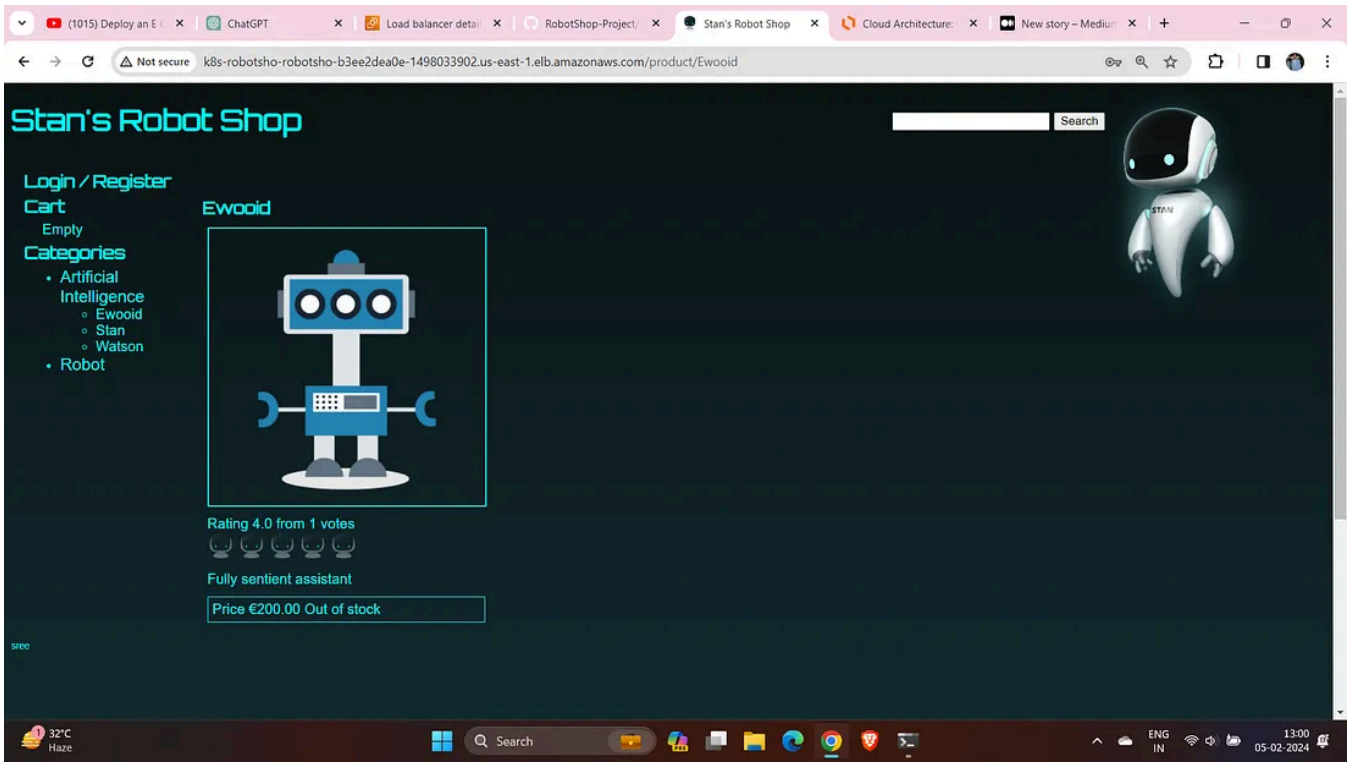
Paste the DNS-name on your favorite browser and access the application.



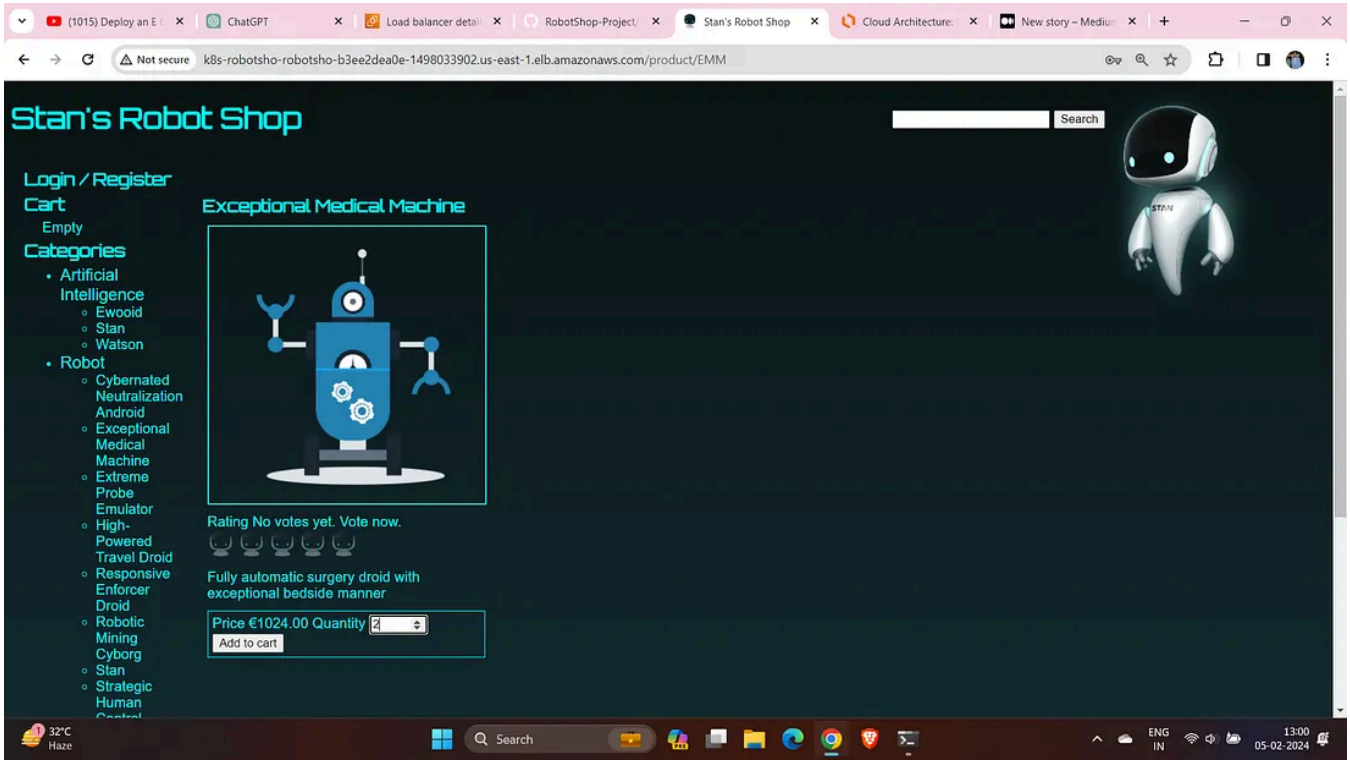
Get Register and login into the application.



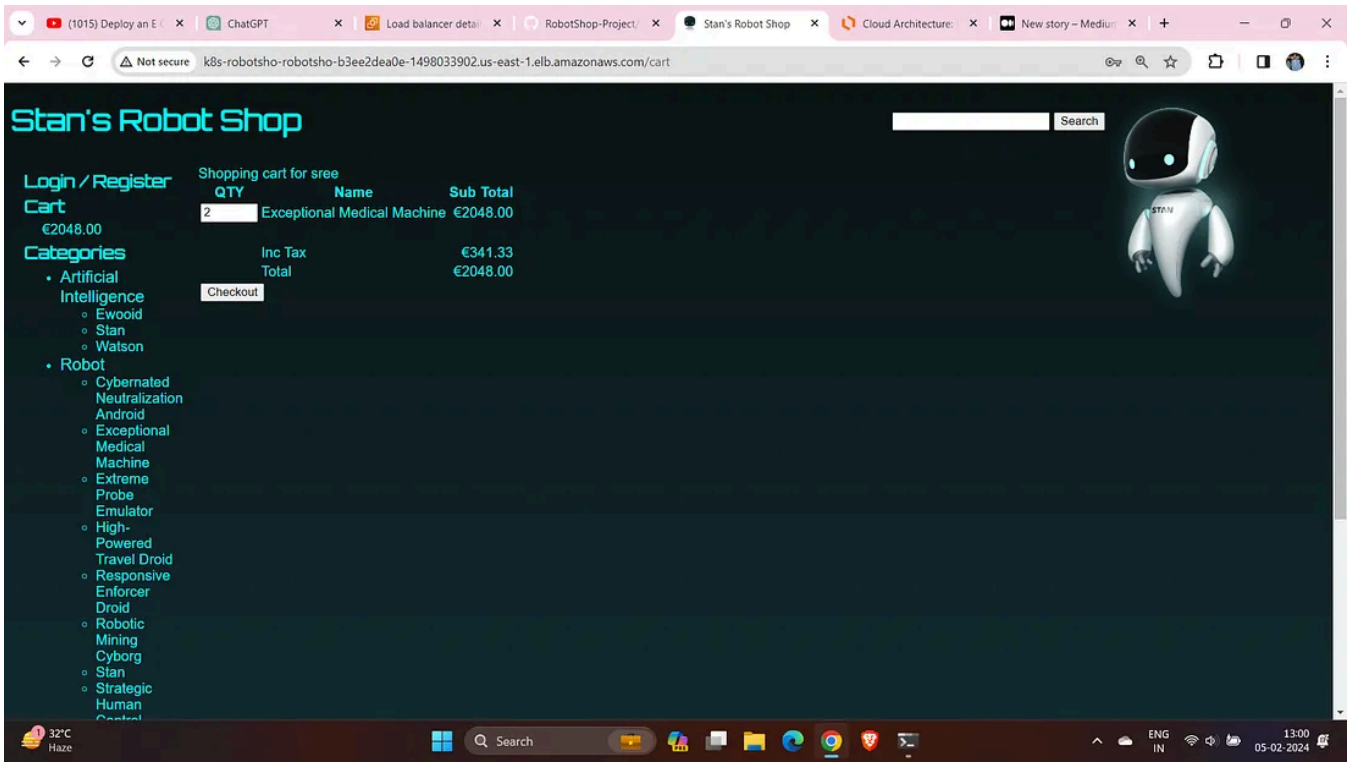
Rate the Artificial Intelligence.



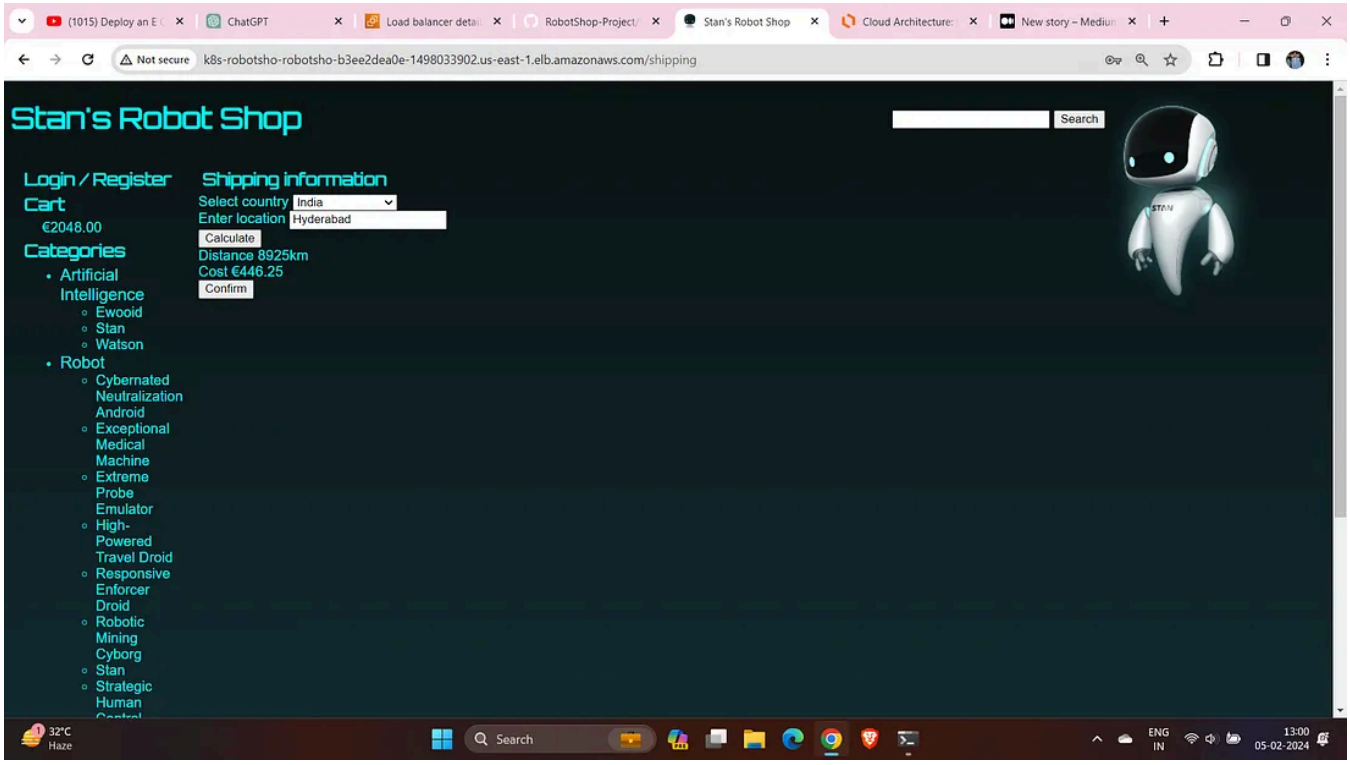
Add to cart the robots.



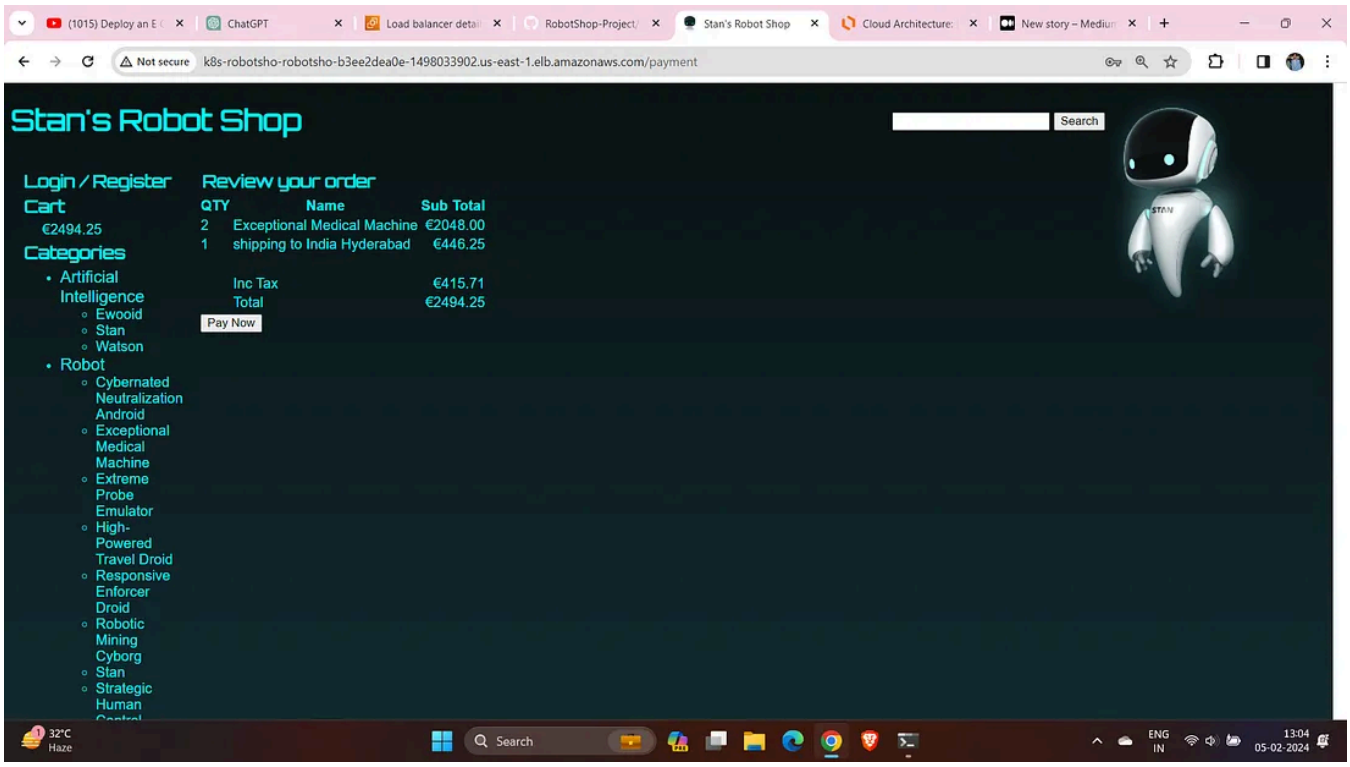
Checkout them.



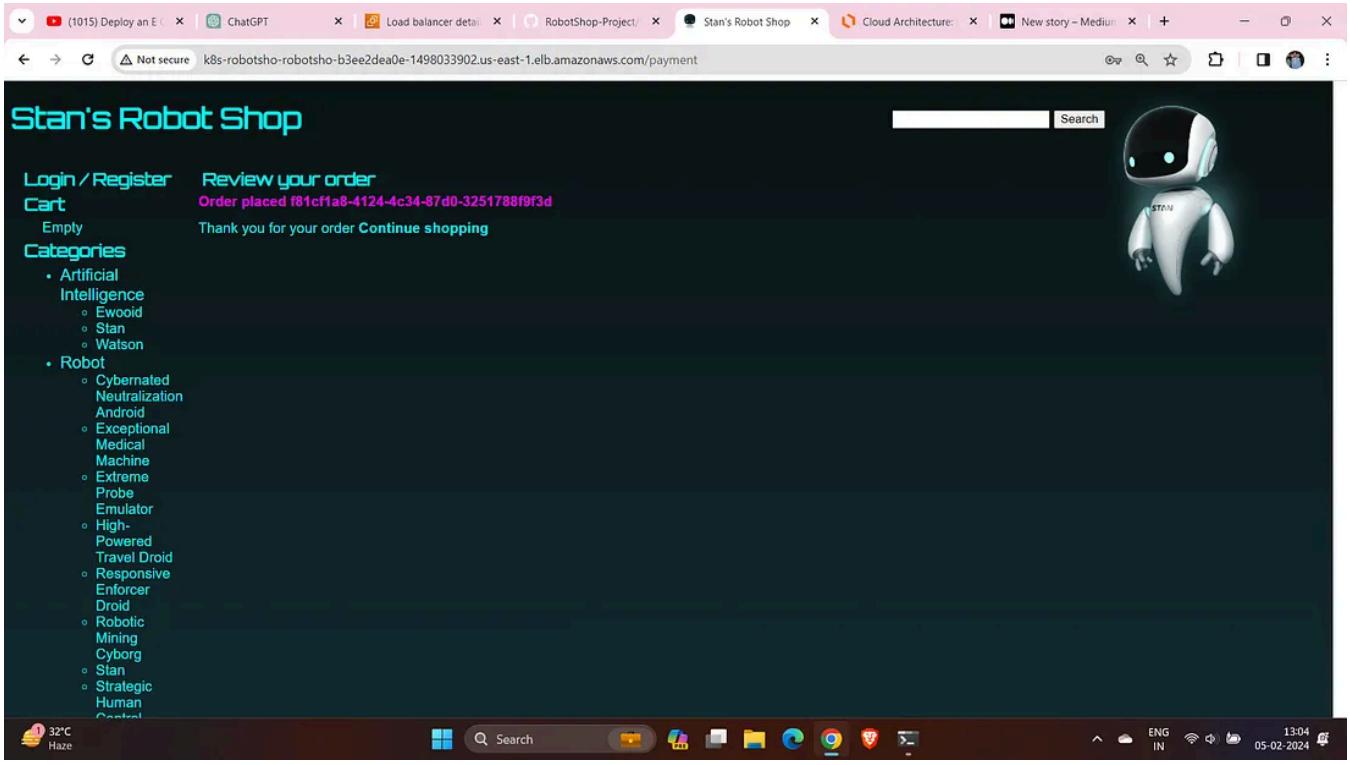
Select your Location and then calculate the price and confirm the order.



Then place the Order.



And pay the amount .



Congratulations... Your Order has been Successfully placed.. :)

DevOps | Devops Practice | Devops Project | AWS | Aws Eks

Follow

Written by Sreedhar Reddy Madithati

339 followers · 3 following

SRE at Autorabit , Hyderabad

Responses (1)

Write a response

What are your thoughts?

md

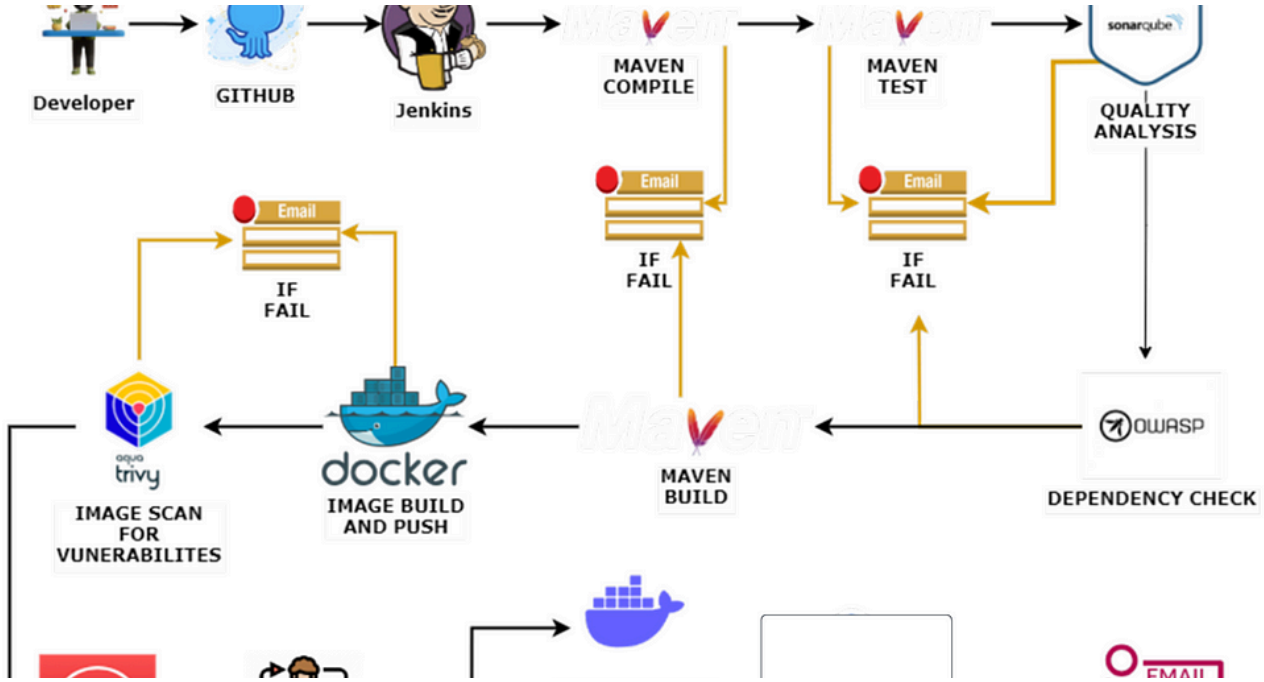
Mkx

Dec 30, 2024

bro can you add Jenkins jenkins pipeline syntax or Jenkins file, it will help me a lot.

Reply

More from Sreedhar Reddy Madithati

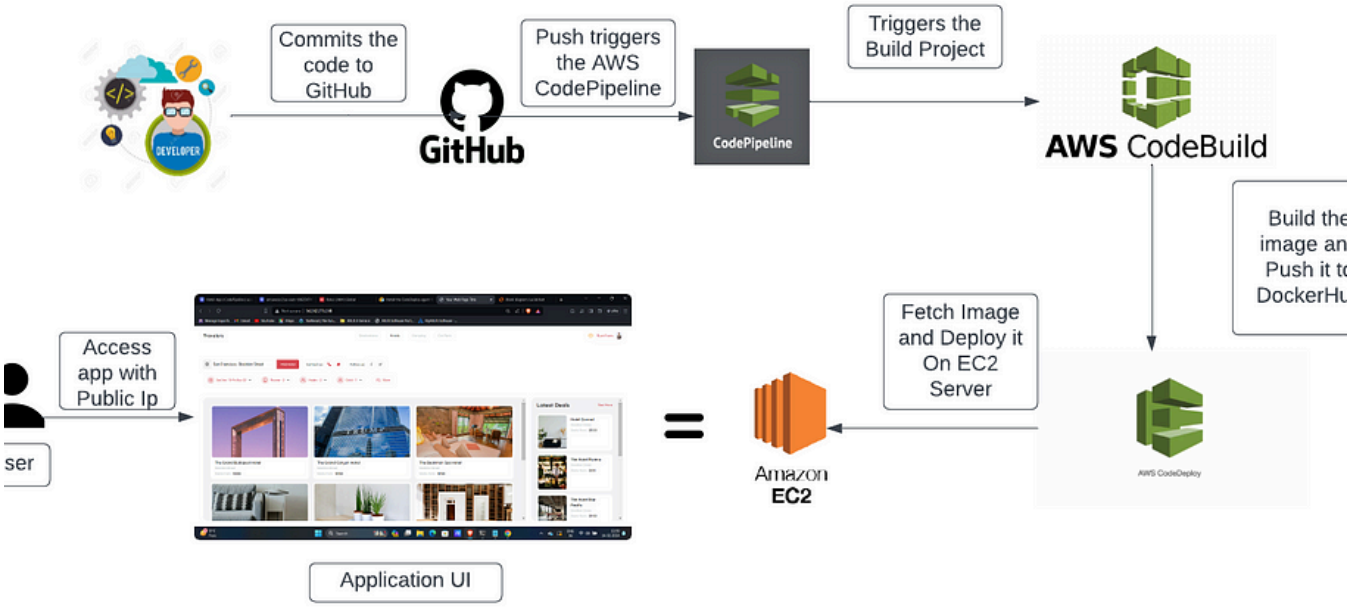


Sreedhar Reddy Madithati

Real Time CI/CD Pipeline for Java Application to Deploy on Apache Server.

Introduction:

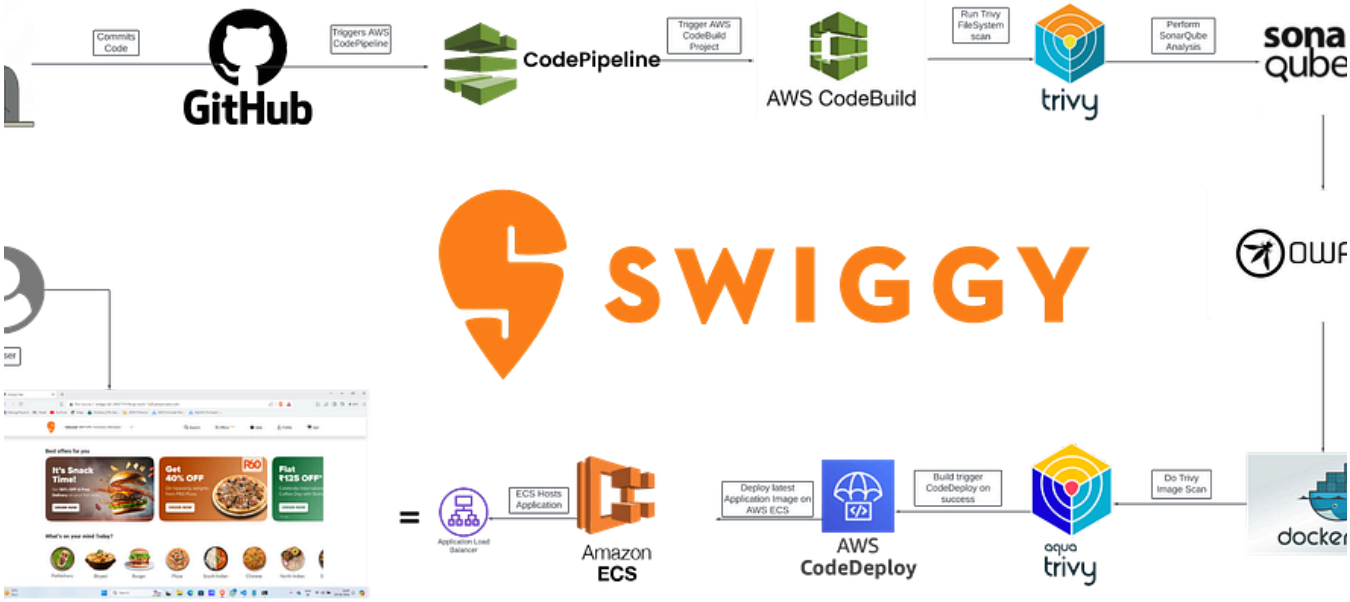
Feb 18, 2024 138 2



Sreedhar Reddy Madithati

AWS CICD Pipeline: Deploy Application on EC2 with AWS Code Pipeline

Feb 14, 2024 84 3



Sreedhar Reddy Madithati

DevSecOps: Blue-Green Deployment of Swiggy-Clone on AWS ECS with AWS Code Pipeline

Introduction:

Apr 10, 2024 22 1

Sreedhar Reddy Madithati

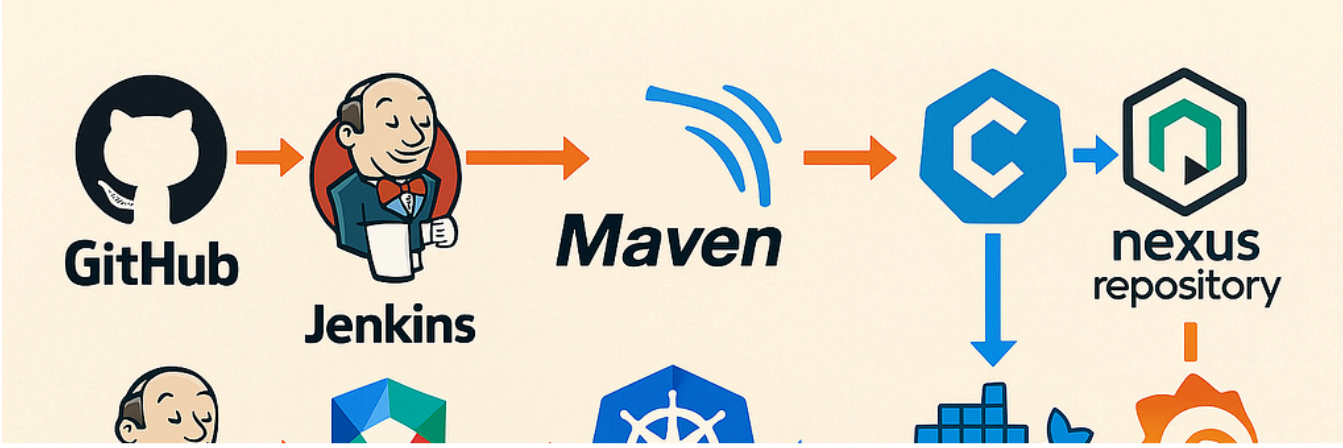
DevSecOps: Zomato Clone Deployment in EKS with GitHub actions and Terraform

INTRODUCTION:

Mar 12, 2024 9

See all from Sreedhar Reddy Madithati

Recommended from Medium

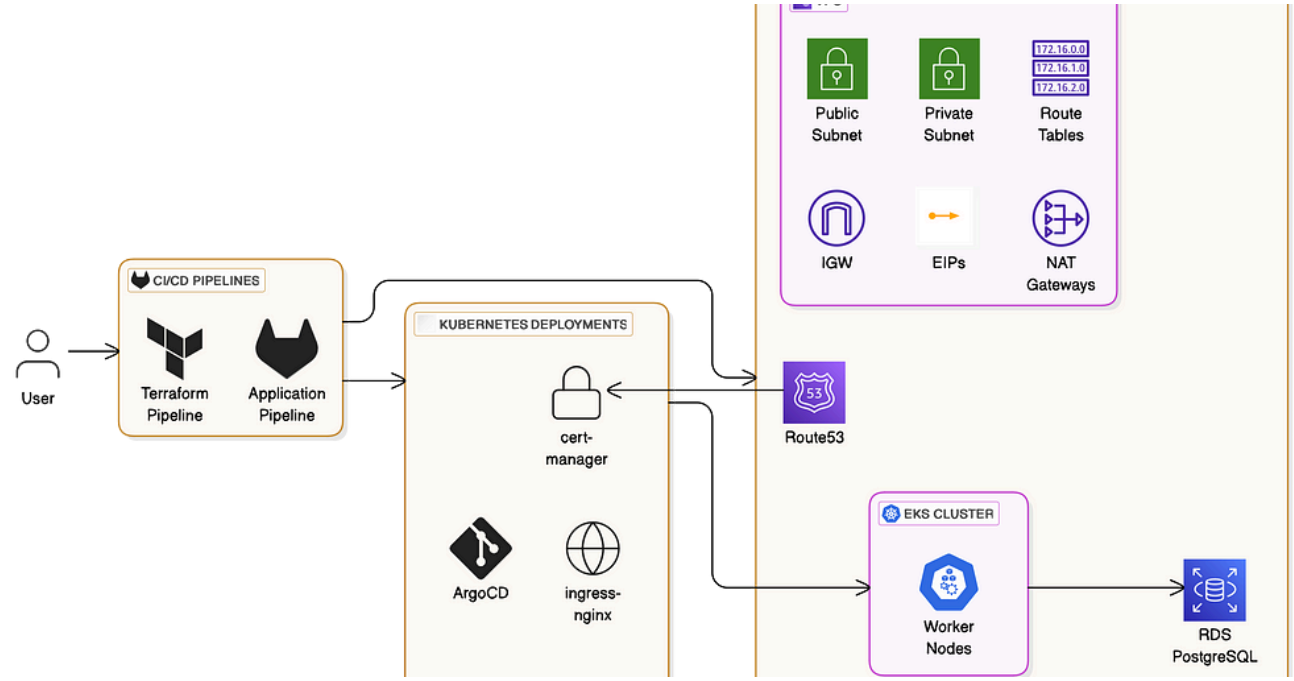


In AWS in Plain English by Master raj

How I Built an End-to-End CI/CD Pipeline on AWS Using Jenkins, Docker, Kubernetes, SonarQube & More

This isn't just another tutorial. It's the exact pipeline I built—fully automated, security-checked, monitored, and deployed in a real...

Jul 8 6

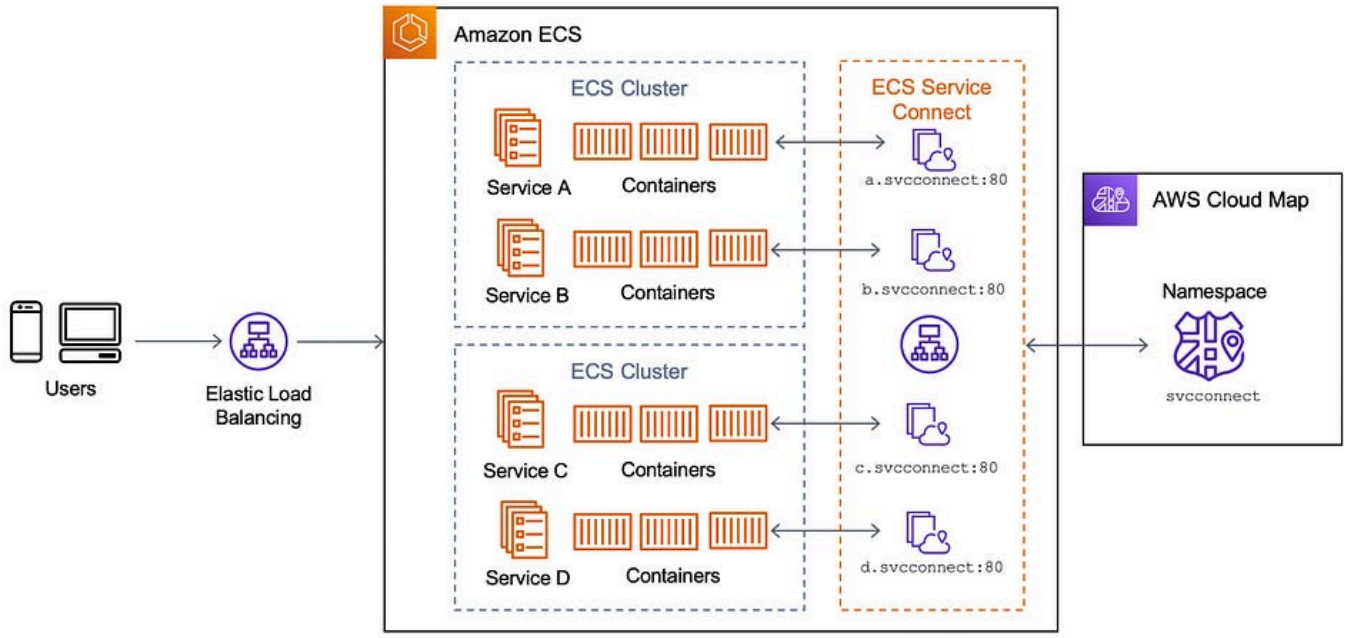


In FAUN.dev — Developer Community by Mohamed ElEmam

Hands-On DevOps Project: Practice Full-Stack Deployment on AWS EKS with Terraform, Helm and more

Hands-On DevOps Project: Practice Full-Stack Deployment on AWS EKS with Terraform, Helm, SonarQube & GitLab CI/CD

Feb 18 103 1



Bhanu Reddy

ECS Best Practices — Cluster Design

1) Introduction :

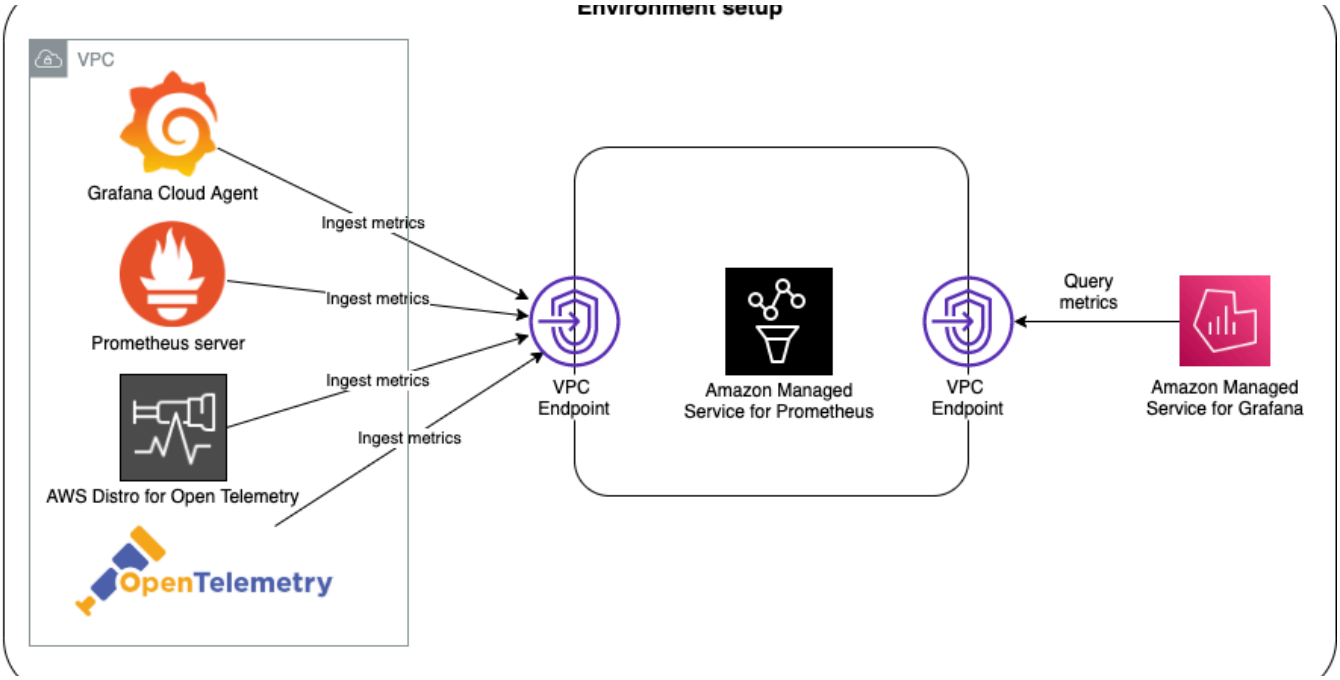
Apr 2 10

In Towards AWS by Ravi Kyada

Enhance EKS Load Balancing: Enable IPVS Mode in AWS EKS

If you're running Kubernetes in production, you already know that performance bottlenecks are the stuff of nightmares.

Apr 7 18

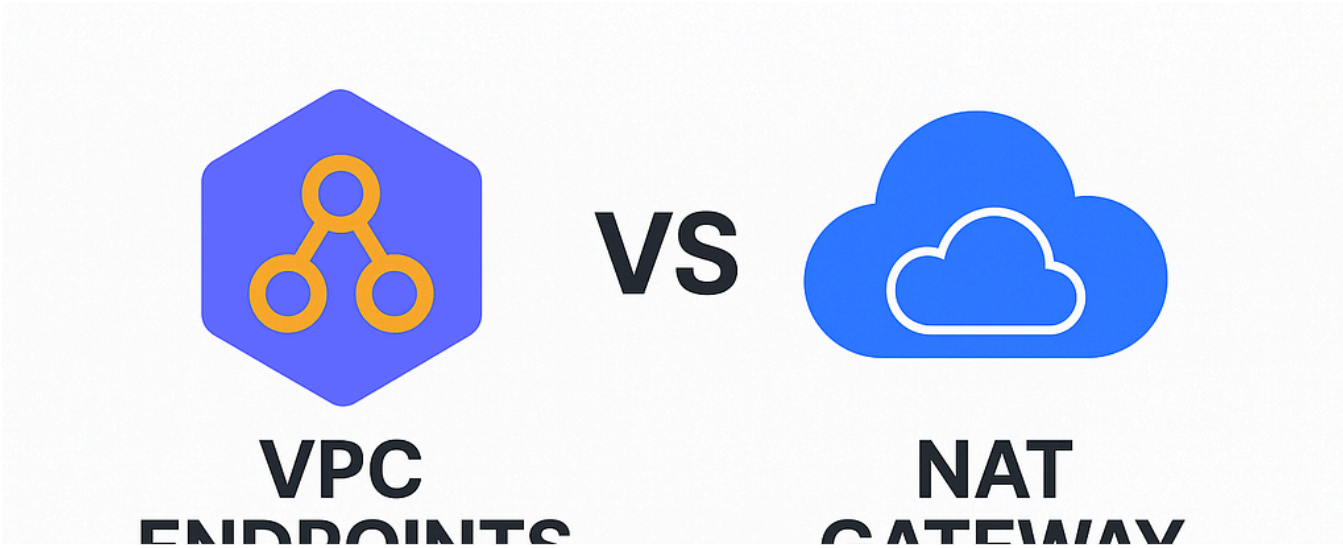


Nagarjun (Arjun) Nagesh

Monitoring and Logging in AWS: CloudWatch vs. Prometheus vs. ELK

Monitoring and logging are critical for ensuring application performance, security, and reliability in cloud environments. AWS offers...

Mar 11



Kinjal Thakkar

NAT Gateway or VPC Endpoints? AWS Cost Battle in Real ECS Deployments

Before we dive in, here's some context. In my previous blog, I discussed how VPC Endpoints could be a great alternative to NAT Gateways for...

Jul 8 5

See more recommendations