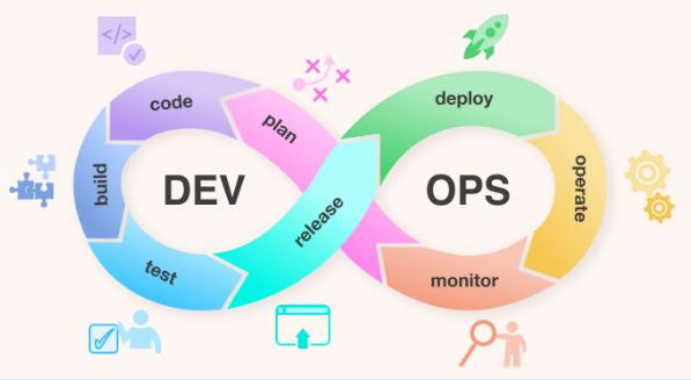
Project – End to End DevOps Deployment for Expense Tracker Application.

Presenter - Navin Kumar

Date – 31-01-2025



Project Scope : To Implement DevOps for an Application Deployment (Expense tracker Application).

- Automate application delivery with industry best practises.
- Focus on operational efficiency , scalability, and reliability.

Goals:

- Reduce the Deployment by 40%.
- Ensure 99.9% uptime.
- Support up to 10,000 concurrent users.
- Cost savings.
- Higher quality software.
- Increased Efficiency.
- Enhanced customer satisfaction.

Inputs ,prerequisites and Initial Setup :

1. A Cloud Account (AWS ,Azure, or Google)
2. Docker Hub Account – To host containerized images of the microservices.
3. Git Repository:

   The cloned repository of the Expense Tracker application, which includes Frontend built in Next.js Backend built in Node.js
4. Estimated Costs: Utilize cost –saving practises like spot instances.

Web Application overview

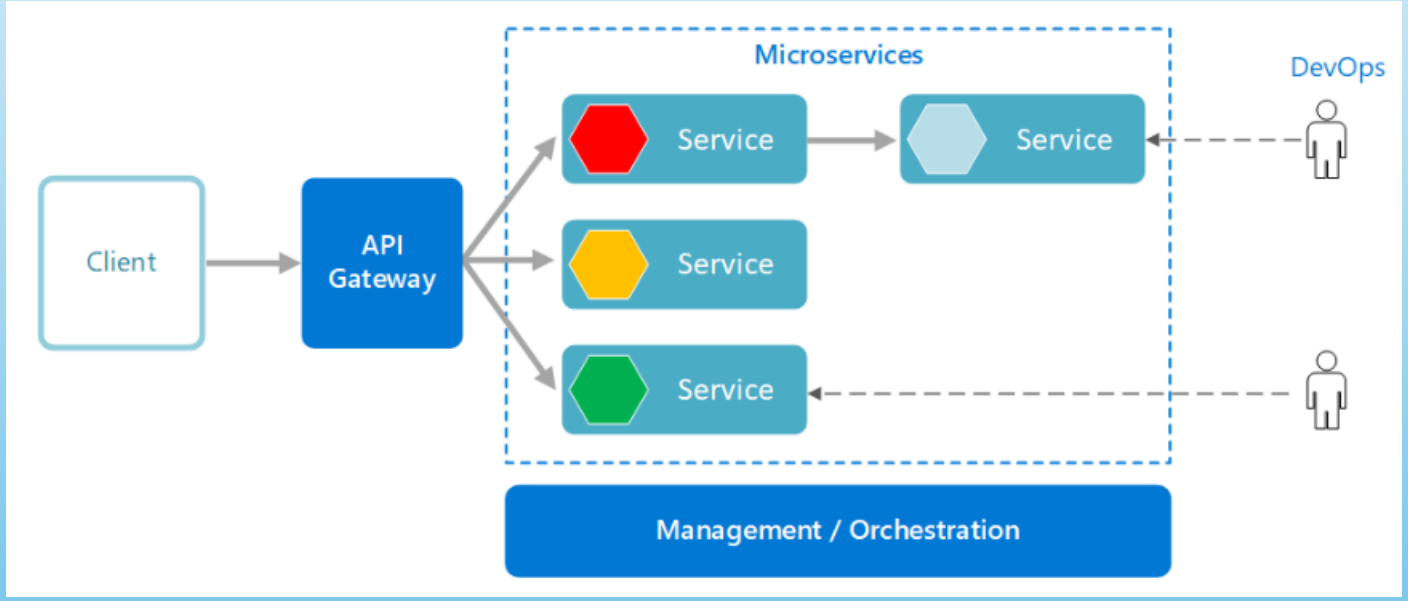
Application: Expense Tracker

Backend: Node.js

Frontend: Next.js

Database: MongoDB

Cache: Redis



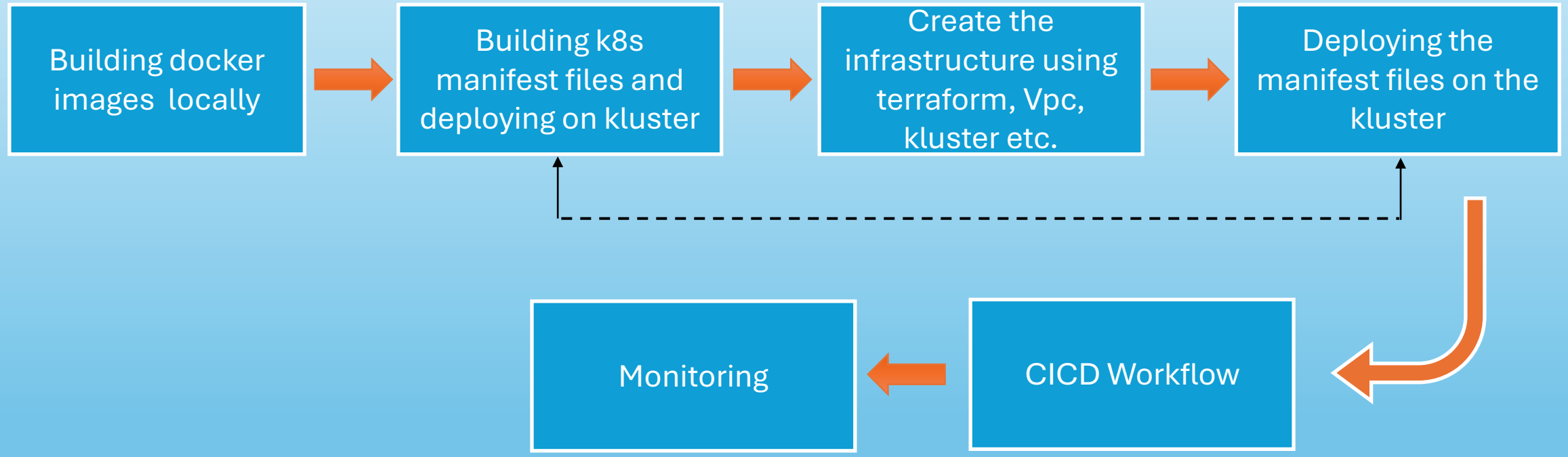
Goal: Build a scalable solution to support zero to thousands of users.

Architecture: Stateful microservices, horizontal & Vertical scaling.

7-day DevOps sprint plan

Product Backlog	Day 1	Day 2	Day 3	Day 4	Day 5	Day 6	Day 7
Project Study & Planning	Project Study	Creating Docker Files	Creating Kubernetes Deployment Files	Creating Terraform Infrastructure	Deployment of Kubernetes Files on Cluster	Project Presentation	Final Project Presentation
Project Plan	Project Plan	Building Docker Images	Creating Cluster & Cluster Config	Creating VPC	Creating CI/CD Workflow (Build & Deploy)	Readme File in GitHub	
Project Estimation	Project Estimation	Docker Compose	Deploying Manifest Files on Cluster	Creating EKS Cluster	Testing Deployment in Self-Hosted Runner		
Project Scope	Project Scope	Pushing Docker Images to Docker Hub	Accessing Microservices on Cluster	Creating Variables	Resolving Errors in CI/CD Pipeline		
Dockerization		Accessing Microservices after Docker		Creating Outputs	Accessing Services After CI/CD Setup		
Kubernetes Setup				Creating S3 Bucket	Monitoring using Prometheus & Grafana		
Infrastructure Provisioning				Creating DynamoDB Table	Installing Prometheus & Grafana in Runner		
CI/CD Pipeline Setup					Accessing Prometheus & Grafana		
Monitoring & Observability					Setting Up Dashboards in Grafana		
Kubernetes Metrics Visibility					Ensuring Kubernetes Metrics Visibility		

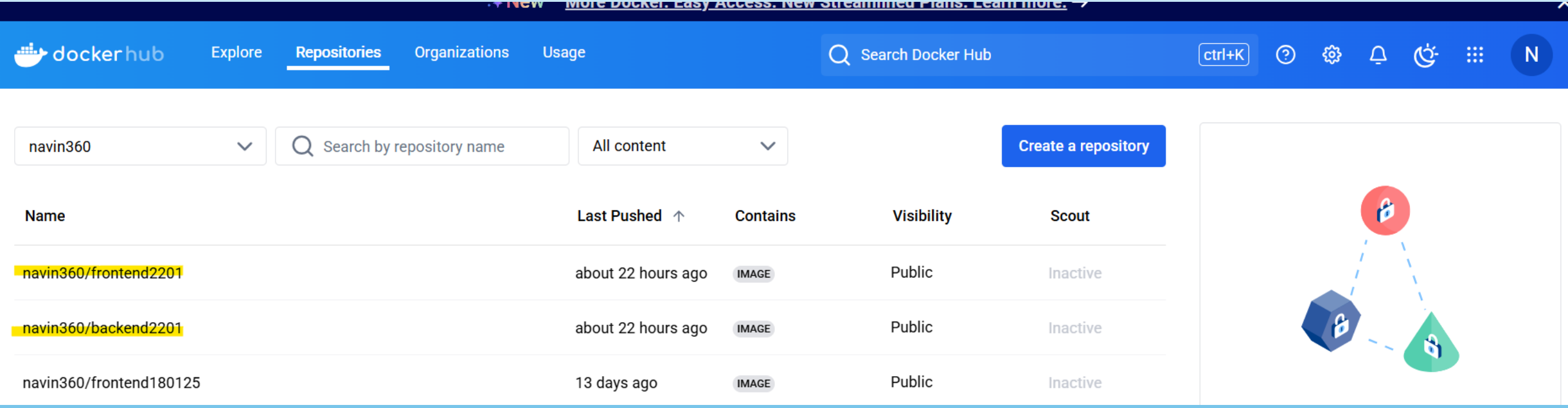
Execution Architecture



Execution Architecture

- Cloned the repository to the local system.
- Created Docker files for the frontend and backend microservices.
- Developed a Docker Compose file with four microservices: frontend, backend, Redis, and MongoDB.
- Verified that the containers are running and confirmed frontend and backend accessibility at their respective ports in the web browser.

Docker images



Execution Architecture

- Developed Kubernetes manifest files for the frontend deployment and service, backend deployment and service, Redis deployment, service, and persistent volume, as well as MongoDB deployment, service, and persistent volume.
- Created a Kubernetes cluster in AWS and deployed the manifest files.
- Verified that all deployments, services, and pods are running successfully.
- confirmed frontend and backend accessibility at their respective ports in the web browser.

K8s manifest files

EXPLORER

OPEN EDITORS
- redis-deployment.yml
- Dockerfile expensy_backend M
- Dockerfile expensy_frontend M
- docker-compose.yml M
- frontend-deploy.yaml

FINAL-PROJECT-IH
- expensy_backend
- expensy_frontend
- backend-deploy.yaml
- backend-service.yml
- commands.txt
- database-deploy.yaml
- database-pv.yml
- database-pvc.yml
- database-service.yml
- docker-compose.yml M
- frontend-deploy.yaml
- frontend-service.yml
- README.md
- redis-deployment.yml
- redis-pv.yml
- redis-pvc.yml
- redis-service.yml

redis-deployment.yml | Dockerfile expensy_backend M | Dockerfile expensy_frontend M | docker-compose.yml M | frontend-deploy.yaml

frontend-deploy.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend2301-deployment
  labels:
    app: frontend2301
spec:
  replicas: 1
  selector:
    matchLabels:
      app: frontend2301
  template:
    metadata:
      labels:
        app: frontend2301
    spec:
      containers:
      - name: frontend2301
        image: navin360/frontend2201
        ports:
        - containerPort: 3000
        env:
        - name: NEXT_PUBLIC_API_URL
          value: "http://a0159d2048f1449839db0f5fb5e14229-1820137776.eu-central-2.elb.amazonaws.com:8706"
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

powershell

```
PS C:\Deveops\Final Project_IH_22\final-project-ih>
```

Execution Architecture

- The infrastructure includes a VPC, Kubernetes cluster, security groups, subnets, S3, and other necessary components.
- State management – terraform backend state stored in S3 with DynamoDB locking.
- After executing the Terraform code and successfully provisioning the infrastructure, verified the setup in AWS, adjusted access permissions, and confirmed the control plane and worker nodes.
- Re-executed all Kubernetes manifest files on the Terraform-provisioned cluster and verified that all deployments, services, and pods are running successfully.
- confirmed frontend and backend accessibility at their respective ports in the web browser.

Terraform infrastructure files

File Edit Selection View Go Run Terminal Help terraform-final-project

EXPLORER

OPEN EDITORS
- vpc.tf final-project
- variables.tf final-project
- EKS.tf final-project
- outputs.tf final-project
- securitygroup.tf final-project

TERRAFORM-FINAL-PROJECT
- final-project
  - .terraform
    - modules
    - providers
  - .terraform.lock.hcl
  - dynamodb.tf
  - EKS.tf
  - outputs.tf
  - S3.tf 1
  - securitygroup.tf
  - terraform.tfstate
  - terraform.tfstate.backup
  - variables.tf
  - vpc.tf

OUTLINE

TIMELINE

vpc.tf | variables.tf | EKS.tf | outputs.tf | securitygroup.tf

final-project > vpc.tf > resource "random_string" "suffix"

```
provider "aws" {
  region = var.aws_region
}

data "aws_availability_zones" "available" {}
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS

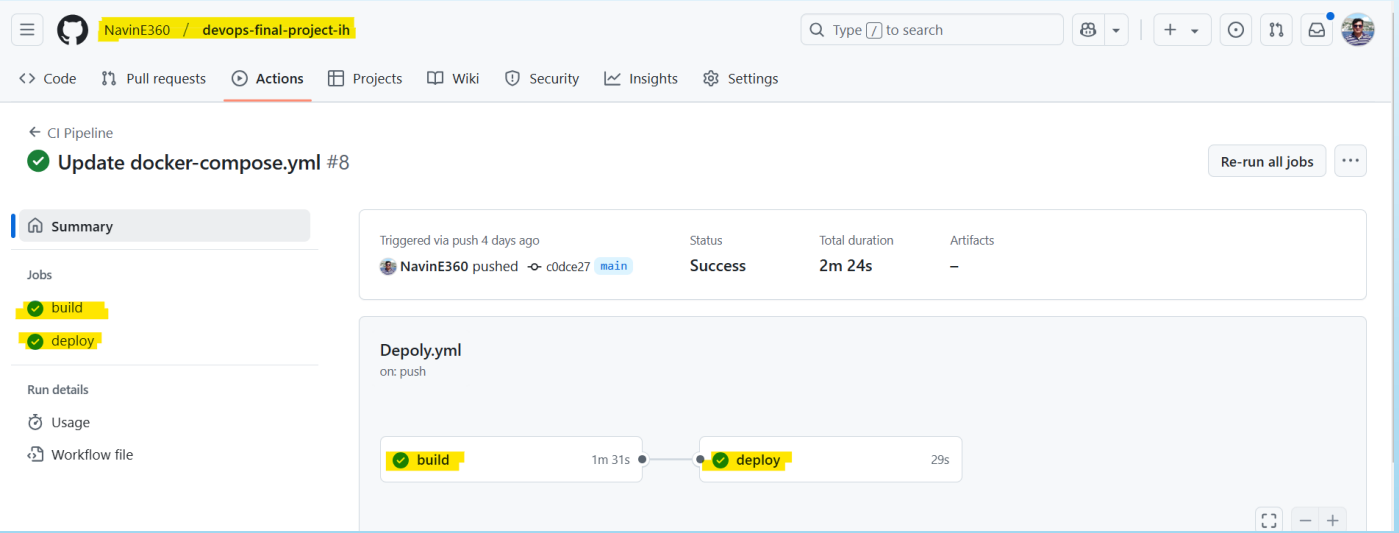
```
PS C:\Deveops\Final Project_IH_22\terraform-final-project\final-project> terraform state list
data.aws_availability_zones.available
aws_dynamodb_table.terraform_lock_table
aws_kms_key.mykey
aws_s3_bucket.mybucket
aws_s3_bucket_policy.mybucket_policy
aws_s3_bucket_public_access_block.mybucket_access
aws_s3_bucket_versioning.mybucket_versioning
aws_security_group.Final_project
random_string.suffix
module.eks.data.aws_caller_identity.current[0]
module.eks.data.aws_iam_policy_document.assume_role_policy[0]
module.eks.data.aws_iam_policy_document.custom[0]
module.eks.data.aws_iam_session_context.current[0]
module.eks.data.aws_partition.current[0]
module.eks.data.tls_certificate.this[0]
module.eks.aws_cloudwatch_log_group.this[0]
module.eks.aws_ec2_tag.cluster_primary_security_group["cluster"]
module.eks.aws_eks_cluster.this[0]
module.eks.aws_iam_openid_connect_provider.oidc_provider[0]
module.eks.aws_iam_policy.cluster_encryption[0]
module.eks.aws_iam_policy.custom[0]
module.eks.aws_iam_role.this[0]
module.eks.aws_iam_role_policy_attachment.cluster_encryption[0]
module.eks.aws_iam_role_policy_attachment.custom[0]
module.eks.aws_iam_role_policy_attachment.this["AmazonEKSClusterPolicy"]
module.eks.aws_iam_role_policy_attachment.this["AmazonEKSVPCResourceController"]
module.eks.aws_security_group.cluster[0]
module.eks.aws_security_group.node[0]
module.eks.aws_security_group_rule.cluster["ingress_nodes_443"]
module.eks.aws_security_group_rule.node["egress_all"]
module.eks.aws_security_group_rule.node["ingress_cluster_443"]
```

Create CI/CD pipelines with Github Actions

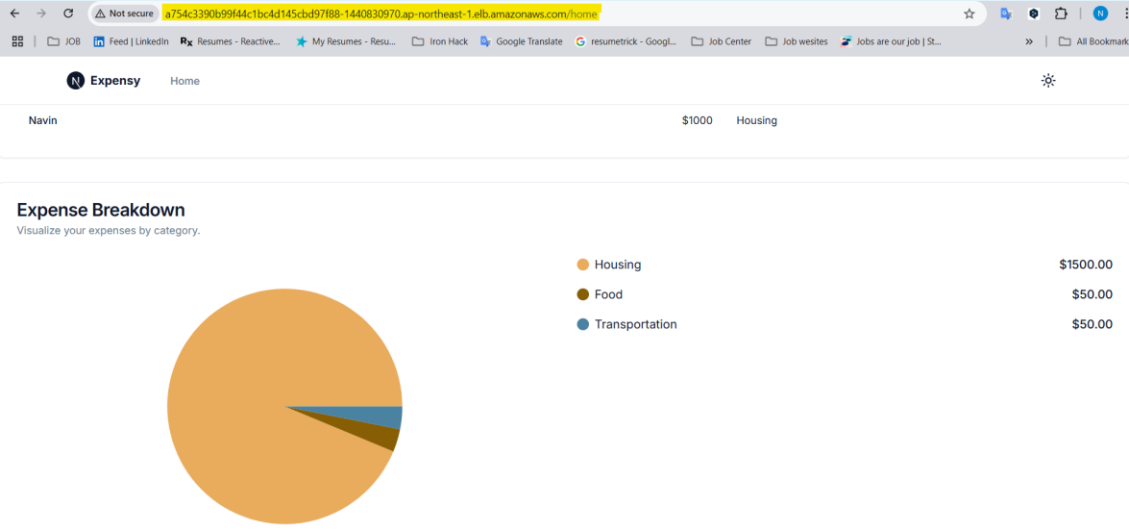
Execution Architecture

- Automated build, test and deployment.
- Tested workflows using a self-hosted runner.
- Verified that all deployments, services, and pods are running successfully.
- Confirmed frontend and backend accessibility at their respective ports in the web browser.

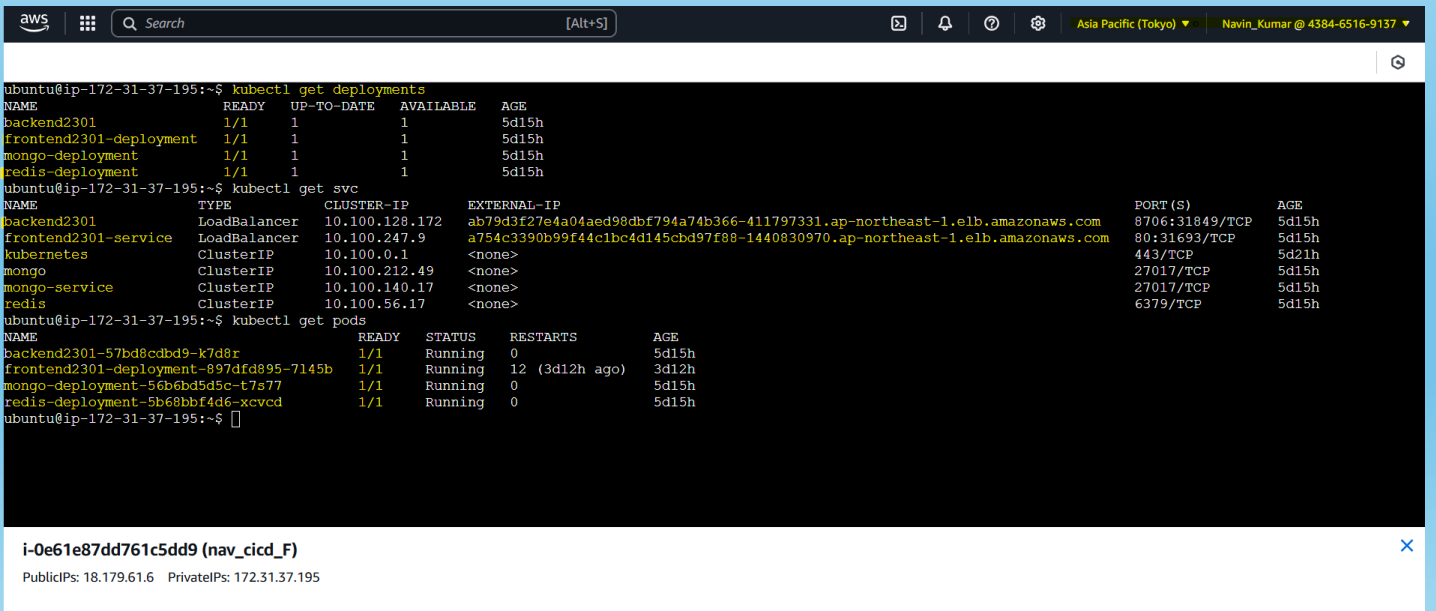
CICD workflow



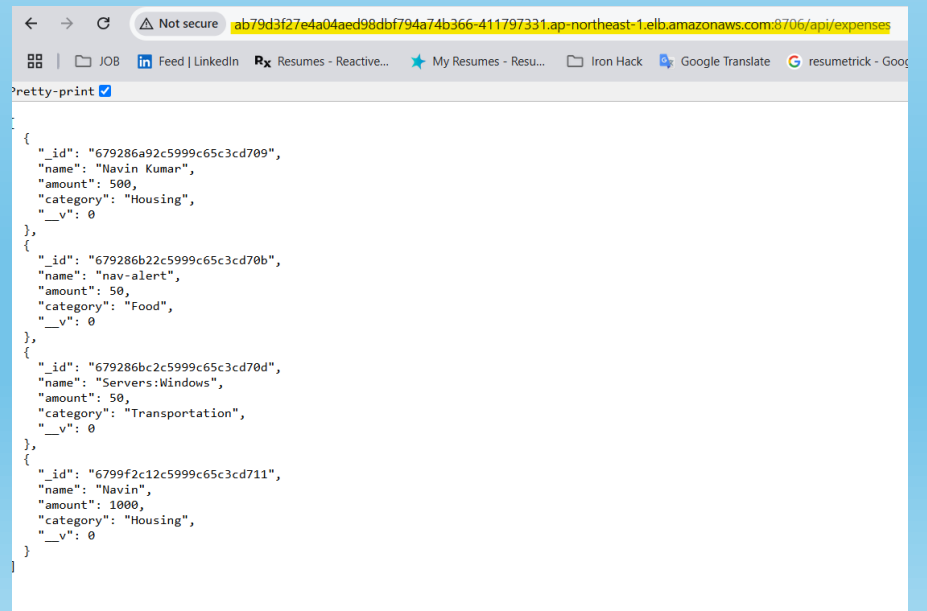
Frontend



EC2 – Self hosted runner



Backend



Prometheus
+
Grafana

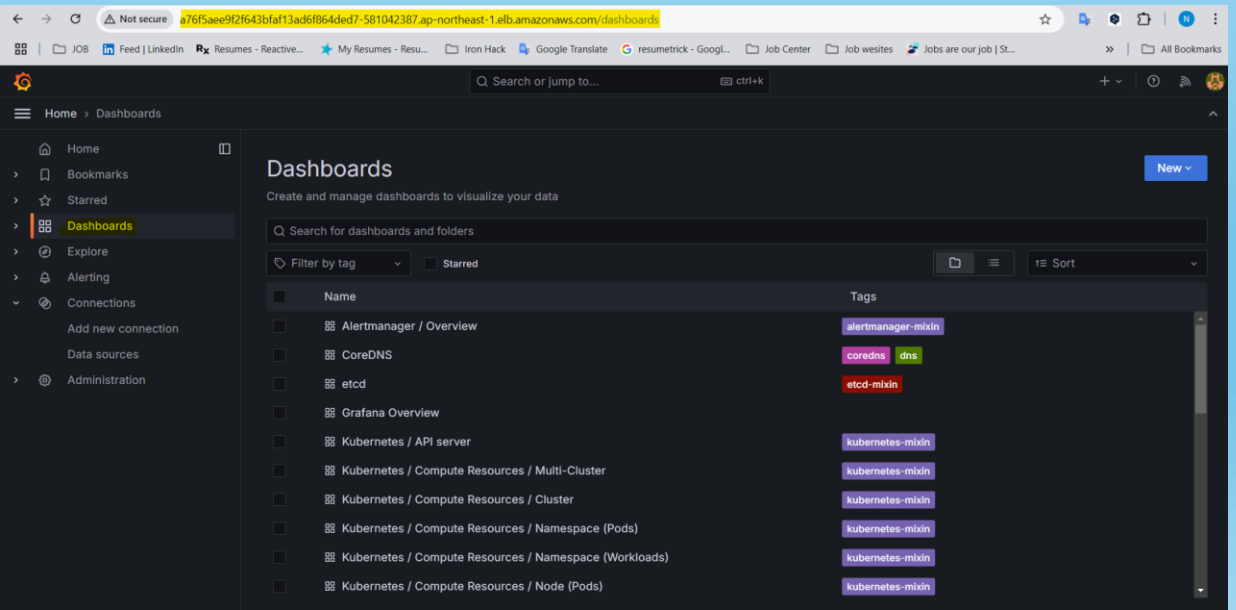
Execution Architecture

- Installed using Helm on a self-hosted EC2 instance.
- Visualized Kubernetes metrics: CPU usage, memory, pod statuses.
- Configured alerting for Slack/Email notifications.
- Ensured secure access with encrypted connections.

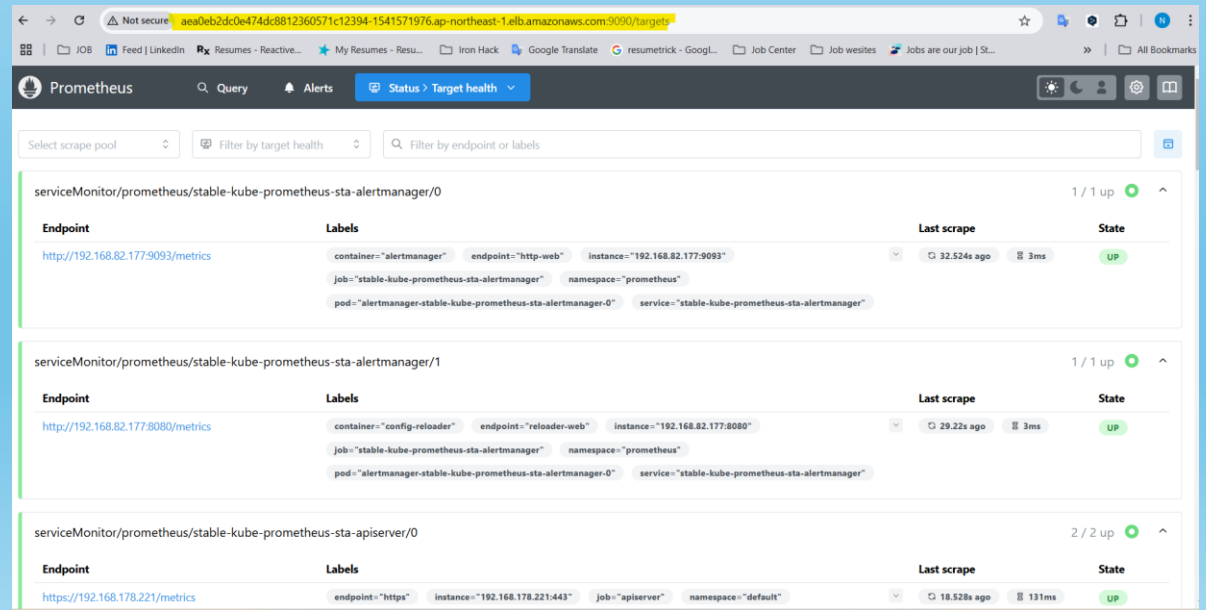
EC2 – Self hosted runner, installed Prometheus and Grafana using Helm package.

```
redis                   ClusterIP      10.100.56.17     <none>                                                                    6379/TCP         5d15h
ubuntu@ip-172-31-37-195:~$ kubectl get pods
NAME                                      READY   STATUS    RESTARTS        AGE
backend2301-57bd8cdbd9-k7d8r              1/1     Running   0               5d15h
frontend2301-deployment-897dfd895-7145b   1/1     Running   12 (3d12h ago)  3d12h
mongo-deployment-56b6bd5d5c-t7s77         1/1     Running   0               5d15h
redis-deployment-5b68bbf4d6-xcvcd         1/1     Running   0               5d15h
ubuntu@ip-172-31-37-195:~$ kubectl get svc -n prometheus
NAME                                      TYPE           CLUSTER-IP       EXTERNAL-IP                                                                PORT(S)
                    AGE
alertmanager-operated                     ClusterIP      None             <none>                                                                     9093/TCP,9094/
TCP,9094/UDP        2d18h
prometheus-operated                       ClusterIP      None             <none>                                                                     9090/TCP
                    2d18h
stable-grafana                            LoadBalancer   10.100.91.38     a76f5aee9f2f643bfaf13ad6f864ded7-581042387.ap-northeast-1.elb.amazonaws.com   80:30659/TCP
                    2d18h
stable-kube-prometheus-sta-alertmanager   ClusterIP      10.100.24.22     <none>                                                                     9093/TCP,8080/
TCP                 2d18h
stable-kube-prometheus-sta-operator       ClusterIP      10.100.235.120   <none>                                                                     443/TCP
                    2d18h
stable-kube-prometheus-sta-prometheus     LoadBalancer   10.100.255.48    aea0eb2dc0e474dc8812360571c12394-1541571976.ap-northeast-1.elb.amazonaws.com   9090:32491/TCP
,8080:30202/TCP     2d18h
stable-kube-state-metrics                 ClusterIP      10.100.97.3      <none>                                                                     8080/TCP
                    2d18h
stable-prometheus-node-exporter           ClusterIP      10.100.57.173    <none>                                                                     9100/TCP
                    2d18h
ubuntu@ip-172-31-37-195:~$
```

Grafana



Prometheus



Deliverables

1. Infrastructure Code:
    - Terraform scripts.
    - Documentation for setup and deployment.
2. Application Deployment:
    - Docker files, Kubernetes manifests.
    - Documentation for building and deploying containers.
3. CI/CD Pipeline Configuration:
    - GitHub Actions workflows.
    - Detailed setup instructions.
4. Monitoring and Logging:
    - Prometheus configuration, Grafana dashboards.
    - Setup and usage documentation.

Thank you !