> This section provides a detailed overview of the infrastructure, application
> deployment process, CI/CD pipeline, and monitoring systems used in the
> project.

Table of Contents

1. Summary

This document introduces the overall architecture, infrastructure setup, application deployment strategy, CI/CD workflow, and monitoring system used in this project.

The goal of this system is to provide a scalable, maintainable, and observable platform for deploying modern applications. It leverages cloud-native tools and best practices to ensure reliability and automation.

Key components include:

- **Infrastructure**: Provisioned using Infrastructure as Code (e.g., Terraform, Ansible), hosted on [e.g., AWS/GCP/Azure].
- **Application Deployment**: Containerized using Docker and orchestrated with Kubernetes.
- **CI/CD Pipeline**: Automates scan, and deployment using [AWS CodePipeline, ArgoCD].
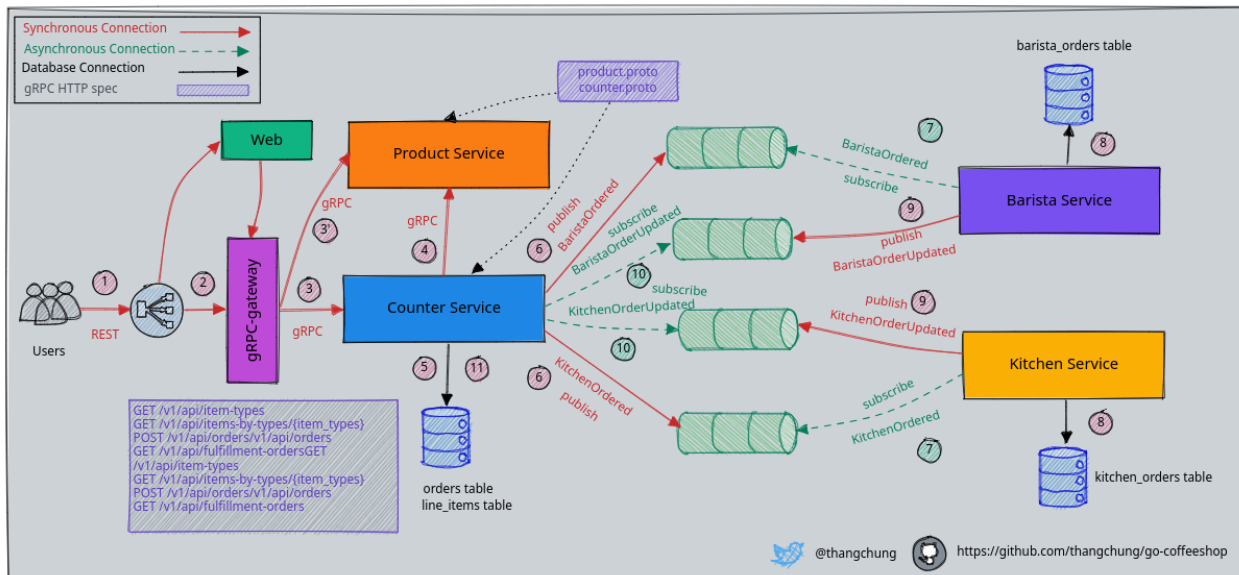- **Monitoring System**: Provides observability through tools likes CloudWatch

This guide is intended for developers, DevOps engineers, and platform engineers who need to understand, deploy, or maintain the system.

2. Architecture

2.1. AWS Infrastructure Architecture
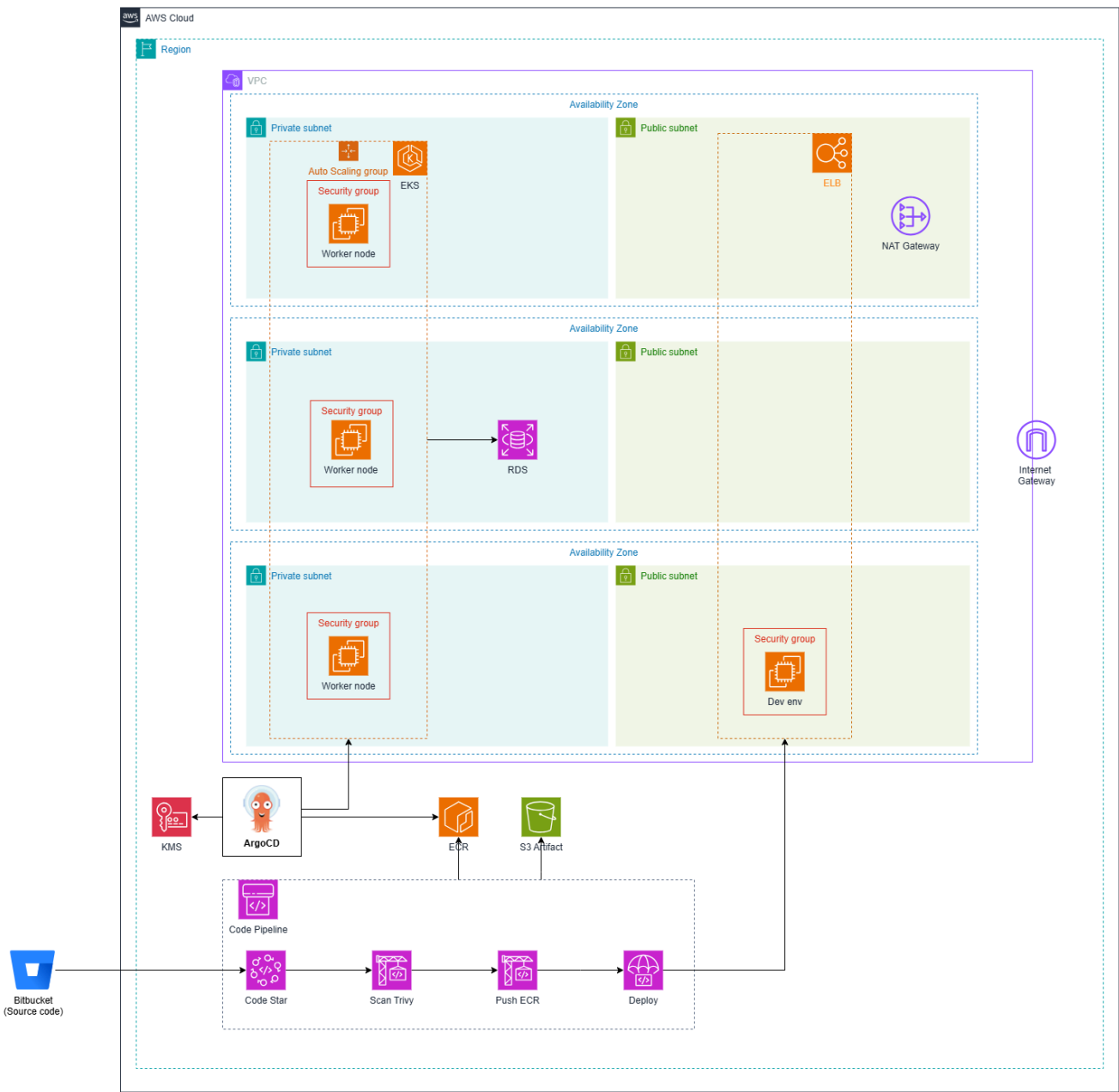
2.2. Application Services Architecture

2.3 Monitoring & Observability Architecture

3. User guideline

3.1. Structure Overview

The root directory contains all necessary components to provision infrastructure, deploy applications, configure monitoring, manage secrets

Here is an overview of each top-level folder and file:

```
├── argo
├── coffeeshop
├── docker-compose.yaml
├── docs
├── eks-setup
├── infrastructure
├── monitor-setup
├── README.md
└── secrets
```

3.2. `infrastructure/`

This directory contains all Terraform files to provision the necessary resources. It includes both custom and public modules.

```
├── backend.tf
├── common_variables.tf
├── locals.tf
├── main.tf
├── provider.tf
├── README.md
├── trainee.tfvars
├── variables.tf
├── modules
│   ├── ci_cd_pipeline
│   │   ├── main.tf
│   │   ├── outputs.tf
│   │   └── variables.tf
│   ├── eks
│   │   ├── main.tf
│   │   ├── outputs.tf
│   │   └── vairables.tf
│   ├── eks_iam
│   │   ├── main.tf
│   │   ├── outputs.tf
│   │   └── variables.tf
│   ├── elasticache
│   │   ├── main.tf
│   │   ├── outputs.tf
│   │   └── variables.tf
│   ├── rds
│   │   ├── main.tf
│   │   ├── outputs.tf
│   │   └── variables.tf
│   └── vpc
│       ├── main.tf
│       ├── outputs.tf
│       └── vairables.tf
└──
```

There are two main workspaces:

- dev: EC2, VPC
- prod: EKS, RDS, VPC, EKS, KMS, CICD

Preparation:

- terraform
- awscli
- eksctl
- Create a CodeStar connection to your Git repo and get `codestar_connection_arn`

Before creating infrastructure, update the `trainee.tfvars` file with your information:

```
region = "<YOUR-REGION>"

name = "<YOUR-PROJECT-NAME>"

availability_zones = [
  "<YOUR-AVAILABILITY-ZONE-1>",
  "<YOUR-AVAILABILITY-ZONE-2>"
]

db_name = "<YOUR-DB-NAME>"

enable_nat_gateway = true

github_owner = "<YOUR-GITHUB-OWNER>"

github_repo = "<YOUR-GITHUB-REPO>"

codestar_connection_arn = "<YOUR-CODESTAR-CONNECTION-ARN>"

account_id = "<YOUR-AWS-ACCOUNT-ID>"

services = {
  product = {
    image = "baominh/go-coffeeshop-product:latest"
  },
  counter = {
    image = "baominh/go-coffeeshop-counter:latest"
  },
  barista = {
    image = "baominh/go-coffeeshop-barista:latest"
  },
  kitchen = {
    image = "baominh/go-coffeeshop-kitchen:latest"
  },
  proxy = {
    image = "baominh/go-coffeeshop-proxy:latest"
```

```
    },
    web = {
      image = "baominh/go-coffeeshop-web:latest"
    },
    vulnerables = {
      image = "vulnerables/web-dvwa:latest"
    },
    rabbitmq = {
      image = "rabbitmq:3.11-management-alpine"
    }
  }
}
```

Update the S3 backend in `backend.tf`:

```
terraform {
  backend "s3" {
    bucket = "<YOUR-BUCKETNAME>"
    key    = "terraform.tfstate"
    region = "<YOUR-REGION>"
  }
}
```

Create Terraform workspaces:

```
terraform workspace create prod
terraform workspace create dev
```

Switch to `prod` workspace and create resources:

```
# Switch to prod workspace
terraform workspace select dev

terraform init
terraform apply -varfile=trainee.tfvars

terraform workspace select prod

terraform init
terraform apply -varfile=trainee.tfvars
```

After creation, you can check your infrastructure:

Dev environment

Instance summary for i-072ff45a61b329cea (opwat-trainee-project-dev) Info

Updated 6 minutes ago

Instance ID	**Public IPv4 address**	**Private IPv4 addresses**
i-072ff45a61b329cea	13.213.6.223 | open address	10.0.1.150
IPv6 address	**Instance state**	**Public IPv4 DNS**
–	⊘ Running	ec2-13-213-6-223.ap-southeast-1.compute.amazonaws.com | open address
Hostname type	**Private IP DNS name (IPv4 only)**	
IP name: ip-10-0-1-150.ap-southeast-1.compute.internal	ip-10-0-1-150.ap-southeast-1.compute.internal	
Answer private resource DNS name	**Instance type**	**Elastic IP addresses**
–	t2.medium	–
Auto-assigned IP address	**VPC ID**	**AWS Compute Optimizer finding**
13.213.6.223 [Public IP]	vpc-06d43e61122506797 (opwat-trainee-project-vpc-vpc)	ⓘ Opt-in to AWS Compute Optimizer for recommendations. | Learn more
IAM Role	**Subnet ID**	**Auto Scaling Group name**
opwat-trainee-project-dev-20250510091519026800000001	subnet-0c98ec8c32f021433 (opwat-trainee-project-vpc-public-1)	–
IMDSv2	**Instance ARN**	**Managed**
Required	arn:aws:ec2:ap-southeast-1:026090549419:instance/i-072ff45a61b329cea	false
Operator		
–		

Details | Status and alarms | Monitoring | Security | Networking | Storage | Tags

▼ Instance details Info

AMI ID	**Monitoring**	**Platform details**
ami-02f7b163d79aae0cb	detailed	Linux/UNIX
AMI name	**Allowed image**	**Termination protection**
amzn2-ami-hvm-2.0.20250512.0-x86_64-gp2	–	Disabled
Stop protection	**Launch time**	**AMI location**
Disabled	Sat May 17 2025 13:23:30 GMT+0700 (Indochina Time) (30 minutes)	amazon/amzn2-ami-hvm-2.0.20250512.0-x86_64-gp2

Prod environment

Amazon Elastic Kubernetes Service > Clusters > opwat-trainee-project-cluster

Amazon Elastic Kubernetes Service

- Clusters
- ▼ **Settings**
  - Console settings
- ▼ **Amazon EKS Anywhere**
  - Enterprise Subscriptions
- ▼ **Related services**
  - Amazon ECR
  - AWS Batch

Documentation

opwat-trainee-project-cluster

Delete cluster | View dashboard

▼ Cluster info Info

Status	**Kubernetes version** Info	**Support period**	**Provider**
⊘ Active	1.32	ⓘ Standard support until March 21, 2026	EKS
Cluster health issues	**Upgrade insights**	**Node health issues**	
⊘ 0	⊘ 4	⊘ 0	

Overview | Resources | Compute | Networking | Add-ons | Access | Observability | Update history | Tags

Details

API server endpoint	**OpenID Connect provider URL**	**Created**
https://9AF37B05E19EE4037D34F2D79B4F2017.gr7.ap-southeast-1.eks.amazonaws.com	https://oidc.eks.ap-southeast-1.amazonaws.com/id/9AF37B05E19EE4037D34F2D79B4F2017	a day ago
Certificate authority	**Cluster IAM role ARN**	**Cluster ARN**
LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSURCVENDQWUyZ0F3SUJBZ0lQTRWYU9wU59rbzh3RFFZSktvWklodmNNQQVFFTEJRQXdGVEVVTUJGR0ExVUUKQXhNS2EzVmlhV0p1WlhSbGN6QWV	arn:aws:iam::026090549419:role/eks-cluster-role View in IAM	arn:aws:eks:ap-southeast-1:026090549419:cluster/opwat-trainee-project-cluster
		Platform version Info
		eks.10

EKS Auto Mode Info

EKS automates routine cluster tasks for compute, storage, and networking to meet application compute needs.

Manage

EKS Auto Mode
Disabled

3.3. Get kube config file

```
eksctl utils write-kubeconfig --cluster opwat-trainee-project-cluster --region ap-southeast-1
```

3.4. Create AWS cred use for SOPS

```
kubectl create secret generic argocd-aws-credentials \
    --from-literal=accesskey=XXXX \
```

```
    --from-literal=secretkey=XXXX \
    -n default
```

3.5. ArgoCD

3.5.1 Install ArgoCD

We will install ArgoCD to manage all resources using GitOps. In the `argo/` directory, run:

```
helm repo add argo https://argoproj.github.io/argo-helm
helm install argocd argo/argo-cd -f values.yaml -n default
```

Forward port to local:

```
kubectl port-forward svc/argocd-server 8080:443
```

Get ArgoCD initial password:
username: admin
password:

```
kubectl -n default get secret argocd-initial-admin-secret -o jsonpath="
{.data.password}" | base64 -d
```

Change default password:

```
argocd login localhost:8080 --username admin --password <OLD_PASSWORD>
argocd account update-password
```

3.5.2 Create argoCD application

Ở phần này chúng ta quan tâm với 2 folder chính là `argo/` và `iam-role/`

```
.
├── application
│   ├── app-project.yaml
│   ├── aws-cloudwatch-metrics.yaml
│   ├── aws-load-balancer-controller.yaml
│   ├── coffeeshop-barista-app.yaml
│   ├── coffeeshop-counter-app.yaml
│   ├── coffeeshop-kitchen-app.yaml
```

```
|       ├── coffeeshop-product-app.yaml
|       ├── coffeeshop-proxy-app.yaml
|       ├── coffeeshop-rabbitmq.yaml
|       ├── coffeeshop-secret.yaml
|       ├── coffeeshop-web-app.yaml
|       ├── external-dns.yaml
|       └── image-updater.yaml
├── README.md
└── values.yaml
...
.
├── aws-load-balancer-controller
|       ├── iam-policy.json
|       └── trust-policy.json
├── cloudwatch-agnet
|       ├── iam-policy.json
|       ├── README.md
|       └── trust-policy.json
├── external-dns
|       ├── iam-policy.json
|       ├── README.md
|       └── trust-policy.json
└── image-updater
        ├── iam-policy.json
        └── trust-policy.json
```

Step to deploy 1 application:

- Create policy, trust-policy
- Create Role
- Add policy to created role
- Add role arn to serviceaccount

argocd bitbucket key

```
sops -e argocd-bitbucket-key.yaml > argocd-bitbucket-key.enc.yaml
```

aws-load-balancer-controller

- Create role in `iam-role/`

- Make sure you update OIDC of your cluster

```
eksctl utils associate-iam-oidc-provider \
  --region ap-southeast-1 \
  --cluster opwat-trainee-project-cluster \
  --approve
```

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

```
aws iam attach-role-policy \
  --policy-arn
arn:aws:iam::026090549419:policy/AWSLoadBalancerControllerIAMPolicy \
  --role-name AmazonEKSLoadBalancerControllerRole
```

```
aws iam update-assume-role-policy \
  --role-name AmazonEKSLoadBalancerControllerRole \
  --policy-document file://trust-policy.json
```

```
k apply -g aws-load-balancer-controller.yaml
```

CloudWatch agent

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

```
aws iam create-role \
  --role-name CloudwatchAgentRole \
  --assume-role-policy-document file://trust-policy.json
```

```
aws iam attach-role-policy \
  --role-name CloudwatchAgentRole \
  --policy-arn arn:aws:iam::026090549419:policy/CloudwatchAgentPolicy
```

external-dns

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

```
aws iam create-role \
  --role-name ExternalDNSRole \
  --assume-role-policy-document file://trust-policy.json
```

```
aws iam update-assume-role-policy \
  --role-name ExternalDNSRole \
  --policy-document file://trust-policy.json
```

```
aws iam attach-role-policy \
  --role-name ExternalDNSRole \
  --policy-arn arn:aws:iam::026090549419:policy/ExternalDNSPolicy
```

```
k apply -g external-dns.yaml
```

argocd image updater

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

```
aws iam create-role \
  --role-name argoCDImageUpdaterRole \
  --assume-role-policy-document file://trust-policy.json
```

```
aws iam attach-role-policy \
  --role-name argoCDImageUpdaterRole \
  --policy-arn arn:aws:iam::026090549419:policy/argoCDImageUpdaterPolicy
```

```
k apply -g image-updater.yaml
```

sops

```
k apply -g coffeeshop-secret.yaml
```

application

```
kubectl apply -f app-project.yaml \
   -f aws-load-balancer-controller.yaml \
   -f coffeeshop-barista-app.yaml \
   -f coffeeshop-counter-app.yaml \
   -f coffeeshop-kitchen-app.yaml \
   -f coffeeshop-product-app.yaml \
   -f coffeeshop-proxy-app.yaml \
   -f coffeeshop-rabbitmq.yaml \
   -f coffeeshop-secret.yaml \
   -f coffeeshop-web-app.yaml \
   -f external-dns.yaml \
   -f image-updater.yaml
```

3.6 Result

You can verify after the pipelines complete
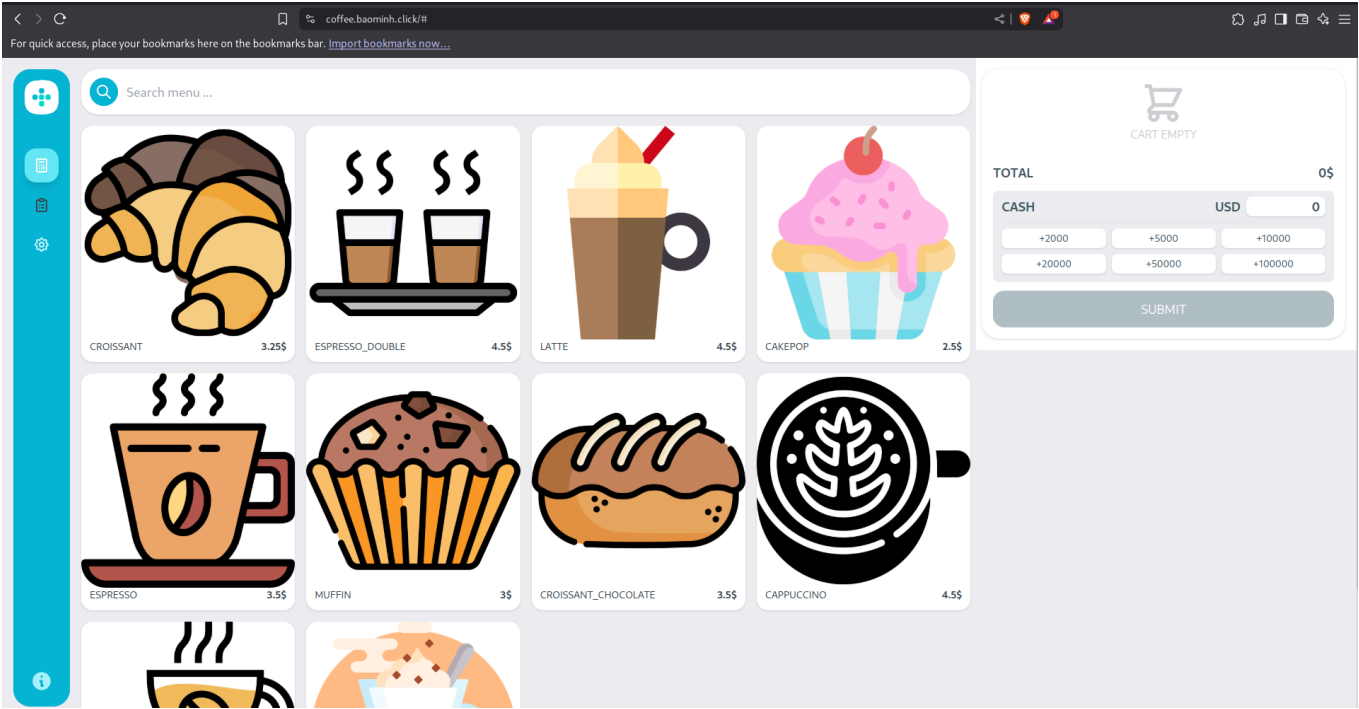
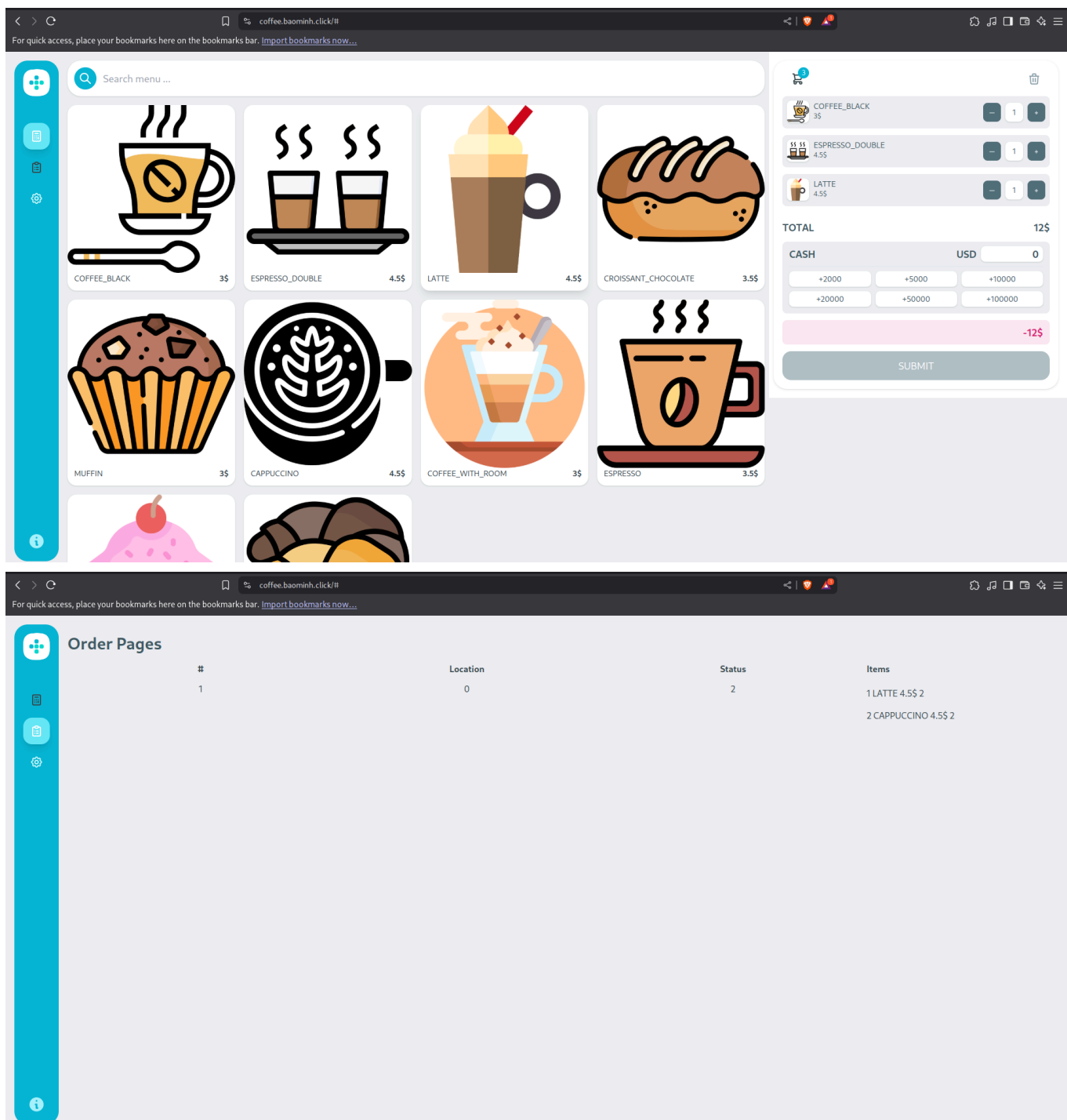You can access the dev environment via the EC2 IP
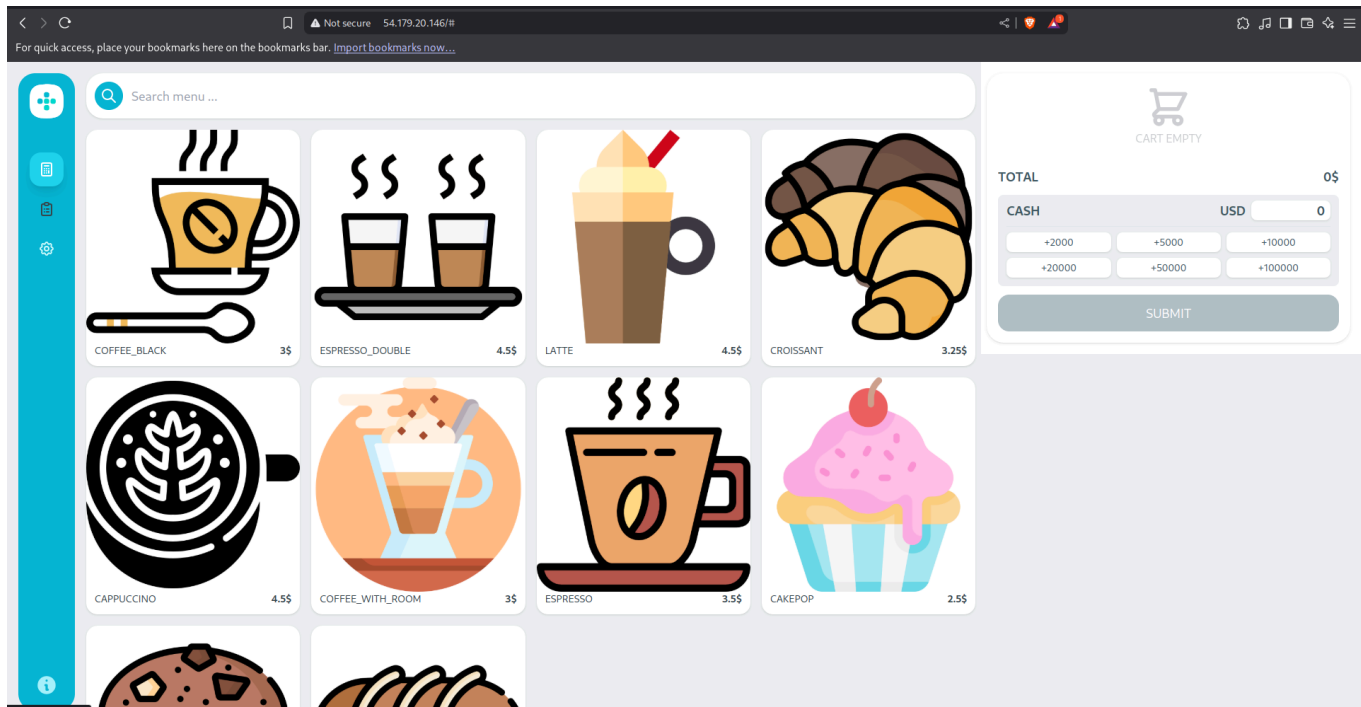
And you will see all applications running successfully
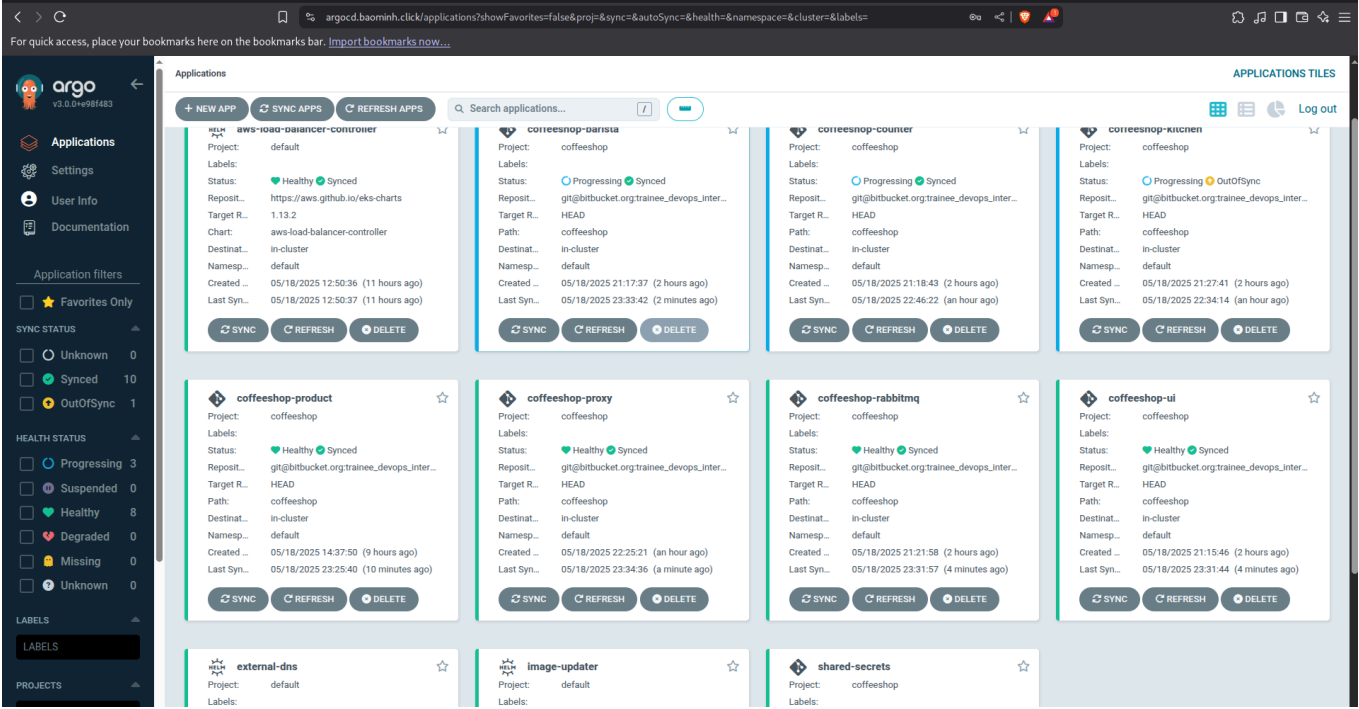
You can access:

4. The homepage of the application

5. Component Description

```
.
├── argo
│   ├── application
│   │   ├── app-project.yaml
│   │   ├── coffeeshop-barista-app.yaml
│   │   ├── coffeeshop-counter-app.yaml
│   │   ├── coffeeshop-fluent-bit.yaml
```

```
│   │   ├── coffeeshop-kitchen-app.yaml
│   │   ├── coffeeshop-product-app.yaml
│   │   ├── coffeeshop-proxy-app.yaml
│   │   ├── coffeeshop-rabbitmq.yaml
│   │   ├── coffeeshop-secret.yaml
│   │   ├── coffeeshop-web-app.yaml
│   │   └── test-app.yaml
│   ├── argocd-image-updater
│   │   ├── argocd-image-updater-policy.json
│   │   ├── argocd-image-updater-trust-policy.json
│   │   └── argocd-image-updater-values.yaml
│   ├── argocd-install
│   │   └── values.yaml
│   ├── README.md
│   └── sops
│       ├── argocd-aws-credentials.yaml
│       ├── kms-decrypt-policy.json
│       └── sops-kms-key-policy.json
├── busybox.yaml
├── coffeeshop
│   ├── Chart.yaml
│   ├── README.md
│   ├── templates
│   │   ├── configmap.yaml
│   │   ├── deployment.yaml
│   │   ├── _helpers.tpl
│   │   ├── hpa.yaml
│   │   └── service.yaml
│   ├── values
│   │   ├── barista
│   │   │   └── values.yaml
│   │   ├── counter
│   │   │   └── values.yaml
│   │   ├── kitchen
│   │   │   └── values.yaml
│   │   ├── product
│   │   │   └── values.yaml
│   │   ├── proxy
│   │   │   └── values.yaml
│   │   ├── rabbitmq
```

```
│   │   │       └── values.yaml
│   │   └── web
│   │       └── values.yaml
│   └── values.yaml
├── docker-compose.yaml
├── docs
│   └── trainee-project.drawio
├── eks-setup
│   ├── aws-load-balancer-controller
│   │   ├── iam-policy.json
│   │   ├── README.md
│   │   ├── service-account.yaml
│   │   └── trust-policy.json
│   ├── external-dns
│   │   ├── externaldns-policy.json
│   │   └── README.md
│   └── README.md
├── infrastructure
│   ├── backend.tf
│   ├── common_variables.tf
│   ├── environments
│   │   └── prod.tfvars
│   ├── locals.tf
│   ├── main.tf
│   ├── modules
│   │   ├── ci_cd_pipeline
│   │   │   ├── main.tf
│   │   │   ├── outputs.tf
│   │   │   └── variables.tf
│   │   ├── eks
│   │   │   ├── main.tf
│   │   │   ├── outputs.tf
│   │   │   └── vairables.tf
│   │   ├── eks_iam
│   │   │   ├── main.tf
│   │   │   ├── outputs.tf
│   │   │   └── variables.tf
│   │   ├── elasticache
│   │   │   ├── main.tf
│   │   │   ├── outputs.tf
```

```
|   |   |   └── variables.tf
|   |   ├── rds
|   |   |   ├── main.tf
|   |   |   ├── outputs.tf
|   |   |   └── variables.tf
|   |   └── vpc
|   |       ├── main.tf
|   |       ├── outputs.tf
|   |       └── vairables.tf
|   ├── provider.tf
|   ├── README.md
|   └── variables.tf
├── ingress
|   └── ingress.yaml
├── monitoring
|   ├── fluent-bit-config.yaml
|   ├── README.md
|   └── trust-policy.json
├── nginx.conf
├── README.md
├── remove-tanit.sh
└── secrets
    ├── agocd-cm-plugin.yaml
    ├── kops-secret.yaml
    ├── kustomization.yaml
    ├── secret.enc.yaml
    └── secret.yaml
```

`argo/`

Contains all **ArgoCD-related configurations**:

- `application/` : ArgoCD `Application` manifests used to deploy services.
- `argocd-image-updater/` : Config files for ArgoCD Image Updater, including image policies and trust policies.
- `argocd-install/` : Values file to install ArgoCD via Helm.
- `sops/` : Contains KMS policies and encrypted AWS credentials for use with SOPS.

`coffeeshop/`

This folder includes the **Helm chart** and values used to deploy the microservices:

- `templates/` : Helm templates for deploying ConfigMaps, Deployments, Services, HPAs, etc.
- `values/` : Subdirectories with environment-specific `values.yaml` for each microservice (e.g., barista, counter, product).
- `values.yaml` : Base values file.

`infrastructure/`

Contains **Terraform modules and configuration** for provisioning cloud infrastructure:

- `modules/` : Reusable Terraform modules for EKS, RDS, VPC, IAM roles, CI/CD pipelines, etc.
- `environments/` : Environment-specific variables (`prod.tfvars`).
- Root-level files (`main.tf` , `provider.tf` , etc.) orchestrate module usage.

`eks-setup/`

Includes **IAM roles and trust policies** required to create Kubernetes service accounts:

- Subfolders like `aws-load-balancer-controller` and `external-dns` contain JSON policy files and Helm configurations.

`ingress/`

Contains the **Ingress resources** to expose services to the internet, including routing rules and annotations.

`secrets/`

Stores **SOPS-encrypted Kubernetes secrets** and ArgoCD plugin configurations:

- Includes both encrypted (`*.enc.yaml`) and decrypted versions for reference.
- Used for securely managing sensitive data like access tokens or passwords.

`monitoring/`

Includes configuration files for **Fluent Bit** and trust policies for setting up observability and log forwarding.

Other top-level files:

- `docker-compose.yaml` : For local development or testing.

- `nginx.conf` : NGINX configuration file (likely used in a reverse proxy or ingress context).
- `remove-tanit.sh` : Script file (purpose assumed to be cleanup or deprovisioning).
- `docs/` : Contains diagrams or documentation files (e.g., `trainee-project.drawio`).