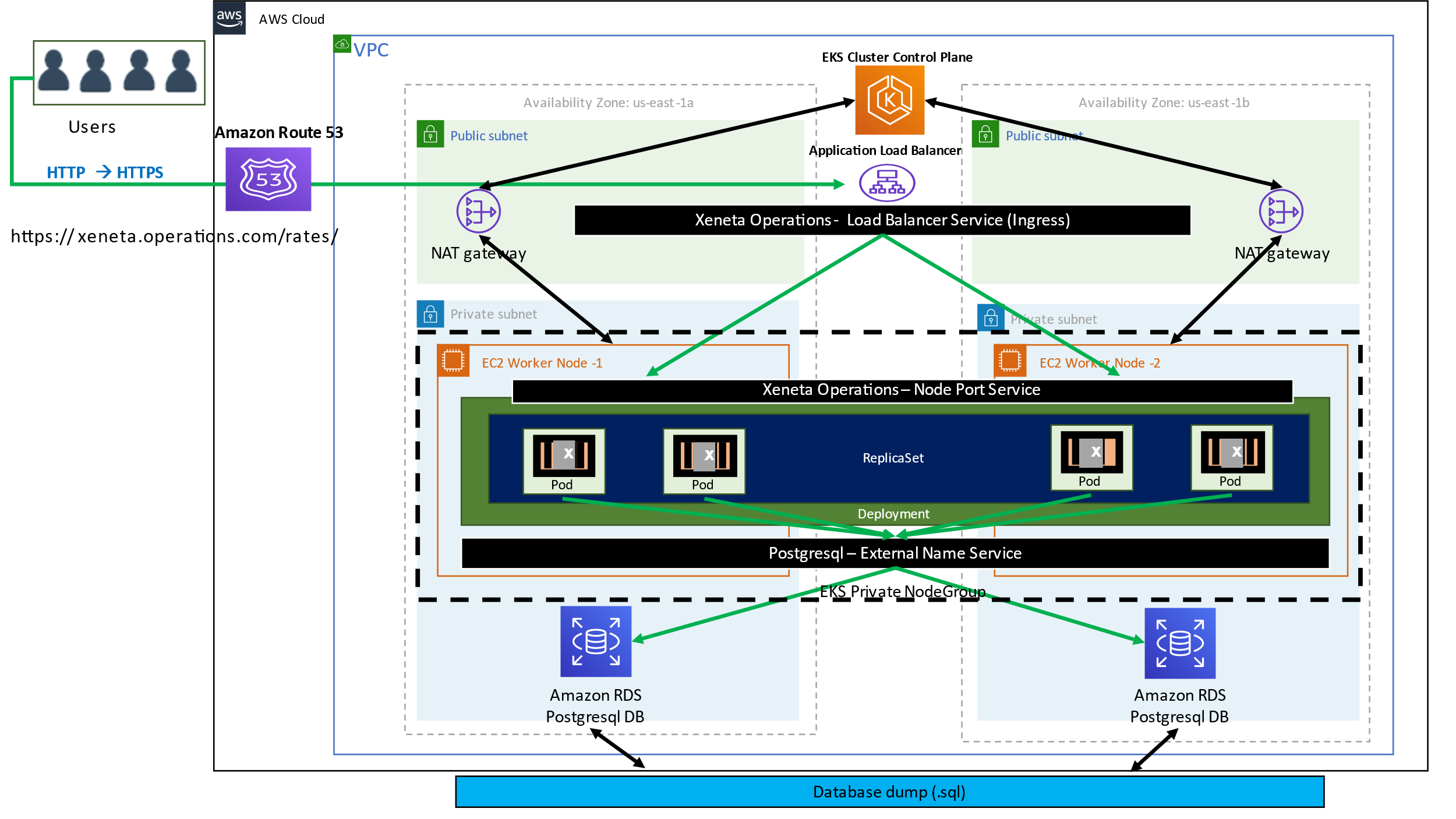
**Architecture**



# Implementation Overview

* AWS is chosen as the cloud vendor to deploy the environment.
* Terraform is the IaC tool used. Terraform is cloud-agnostic. It also enables state management of resources and simplifies the management of infrastructure.
* API service is packaged into Docker images to be deployed as Docker containers. Docker provides the ability to run anywhere, in a consistent and isolated environment, which is also cost-effective. Deployment is also fast.
* Kubernetes (AWS EKS) is the container orchestration service used. It is the best container orchestration tool for several reasons. It is proven and battle-tested with a large community for support and development. It increases developer productivity by reducing the time and complexity of deployment to environments.

# Service Usage

Terraform Backends:

* Terraform state files are stored in s3 buckets.
* DynamoDB table locks the state file from concurrent executions.

Networking:

* VPC – VPC is created with 3 subnets each of public, private, and database (private) subnets category.
* Public subnets are used to deploy the Application ALB (ingress), NAT Gateways, and Database agent EC2.
* Private subnets host the worker nodes as we do not want nodes to be exposed to the internet and RDS Instance.
* Other networking components like Route tables, NAT Gateways, etc are also deployed.

Database:

* RDS Postgresql 13 is the database chosen as per the requirements.
* RDS is deployed in the private subnet.

S3 Bucket:

* S3 bucket stores the data file.

Data Ingestion– EC2:

* Data file provided is ingested to RDS via EC2 instance running in a public subnet.
* EC2 user-data installs psql client, SSM Agent and other softwares like AWS CLI,
* During the launch, the EC2 will fetch the DB password from SSM and also gets the data file from s3. It then loads the data onto the RDS DB using the psql client.
* EC2 also comes bundled with an SSM agent for the user to login to EC2 to troubleshoot the database.

Docker Image and ECR:

* Since the docker container is used for app deployment, the image has to be built and uploaded to the docker repo. ECR is chosen as the docker repo for it is seamless and is readily available to use.

EKS Infra:

* EKS, Worker nodes, OIDC Configuration for the service accounts, etc are deployed after the data is successfully loaded.
* OIDC configuration allows ALB ingress controller service account to authenticate with AWS and create the required resource.
* Kubernetes deployments are made using Helm. Helm charts are created and deployed through terraform.

Helm Charts:

* There are 2 charts. The first one deploys the Kubernetes alb ingress controller and the other one deploys the application deployment and services along with the ingress object.
* Post-deployment, the application can be accessed using the ALB DNS name and subsequent path context.
* For Helm charts, Refer <https://github.com/Aravind04101996/x-ops-helm-final> repository.

# Repository Information

|  |  |
| --- | --- |
| .github/workflows/\* | GitHub CI workflow to deploy infra in AWS. |
| backend-terraform/\* | Creation of S3 Bucket for state file storage, Dynamo DB for state locking, OIDC IAM Identity provider, IAM roles and policies for GitHub ci |
| database-infra-terraform/\* | Creation of network components (vpc, subnets, natgw etc. . .), DB agent with Userdata to load data into RDS, RDS DB Instance, Security Groups, IAM Policies & Roles. |
| eks-cluster-terraform/\* | Creation of EKS Cluster, Node Groups, ALB ingress role and policies for the ingress controller service account, Security Groups. |
| Dockerfile | API service is packaged into Docker images to be deployed as Docker containers |