Visão Geral do Projeto

Neste projeto, construí uma infraestrutura escalável e segura na AWS para uma aplicação de logística usando **AWS CDK (Cloud Development Kit) em Java**. O ambiente foi projetado seguindo **boas práticas de arquitetura na nuvem**, garantindo escalabilidade, segurança e facilidade de deploy.

A infraestrutura é organizada em diferentes **stacks**, cada uma responsável por um serviço específico.

Componentes Principais da Infraestrutura

VpcStack → Rede e Comunicação

A primeira camada da infraestrutura é a **VPC (Virtual Private Cloud)**, que define a rede onde os recursos da AWS vão rodar.

- Possui **subnets públicas e privadas** para manter segurança e performance.
- Garante que o banco de dados e outros serviços críticos **não fiquem expostos** à internet.

SecurityGroupStack → Segurança

Criamos **grupos de segurança (Security Groups)** para controlar quem pode acessar cada serviço.

- Apenas o **ECS pode acessar o RDS (PostgreSQL)**.
- O Load Balancer aceita conexões da internet, mas a aplicação em si fica protegida.

RdsStack → Banco de Dados PostgreSQL

Utilizamos **Amazon RDS** para armazenar os dados da aplicação de forma confiável.

- As credenciais do banco são gerenciadas pelo **AWS Secrets Manager**, garantindo segurança.
- O banco está em **subnet privada**, impedindo acessos externos diretos.

MongoDbStack → Banco NoSQL (MongoDB Atlas)

O projeto usa **MongoDB Atlas**, que é um serviço gerenciado de MongoDB.

- As credenciais também são armazenadas no **Secrets Manager**.
- Configuração feita para permitir conexão apenas do ECS.

EcsClusterStack → Orquestração de Containers

Usamos **Amazon ECS (Elastic Container Service)** para rodar a aplicação dentro de **containers**.

- O ECS está configurado para rodar no modo **Fargate**, eliminando a necessidade de gerenciar servidores.

- O cluster permite escalabilidade automática conforme a demanda da aplicação.

EcrStack → Armazenamento de Imagens Docker

Para manter controle sobre as imagens Docker da aplicação, utilizamos o **Amazon ECR (Elastic Container Registry)**.

- A cada atualização de código, uma nova imagem é enviada para o ECR.
- O ECS baixa a imagem mais recente automaticamente durante a implantação.

EcsServiceStack → Serviço da Aplicação

A aplicação em si roda dentro do ECS, exposta ao público por meio de um **Load Balancer**.

- O serviço ECS se conecta ao PostgreSQL e ao MongoDB sem expor credenciais.
- O tráfego é roteado de forma eficiente para múltiplas instâncias do serviço.

Fluxo Completo do Deploy

O código da aplicação é **containerizado** com Docker e enviado para o **Amazon ECR**.
O CDK **gera e aplica a infraestrutura** automaticamente.
O ECS **baixa a nova imagem** e inicia a aplicação.
A aplicação **se comunica** com RDS e MongoDB **de forma segura** usando Secrets Manager.
O **Load Balancer distribui o tráfego** para as instâncias da aplicação.

Benefícios da Arquitetura

Alta Disponibilidade: ECS com Load Balancer garante que a aplicação sempre esteja no ar.
Escalabilidade: O ECS pode aumentar ou reduzir o número de instâncias conforme a demanda.
Segurança: Credenciais armazenadas de forma segura, banco de dados em subnet privada.
Automação: Infraestrutura gerenciada pelo CDK, reduzindo tarefas manuais.

Subnets (Rede)

As **subnets** são definidas na **VpcStack**, onde criamos uma **VPC customizada** para isolar os recursos.

Tipos de Subnets Criadas

- **Subnets Públicas**
    - São usadas pelo **Load Balancer** e outros serviços que precisam estar acessíveis pela internet.
    - O tráfego externo pode entrar, mas com controle via Security Groups.
- **Subnets Privadas**
    - São usadas para **o ECS (contêineres da aplicação) e o RDS (banco de dados PostgreSQL)**.
    - Elas **não têm acesso direto à internet**, garantindo maior segurança.
    - Para que os contêineres ECS possam acessar serviços externos (ex: MongoDB Atlas), usamos um **NAT Gateway**.

Por que essa separação?

- **O Load Balancer fica na subnet pública** porque ele precisa receber tráfego da internet.
- **O RDS e os serviços internos ficam na subnet privada** para evitar acessos não autorizados.
- O **ECS precisa acessar a internet** (para baixar imagens do Docker, por exemplo), então ele está em subnet privada com NAT Gateway.

Segurança (Security Groups e Controle de Acesso)

Para garantir que apenas os serviços corretos possam se comunicar entre si, criamos **Security Groups específicos** na **SecurityGroupStack**.

Security Group do Load Balancer

- Permite tráfego HTTP/HTTPS **da internet** para a aplicação (porta 80 ou 443).
- Encaminha as requisições para o ECS.

Security Group do ECS

- Permite tráfego **apenas do Load Balancer** para a aplicação (exemplo: porta 8081).
- Permite saída para a internet via NAT Gateway (para baixar imagens do ECR, por exemplo).
- Pode acessar o RDS e o MongoDB Atlas.

Security Group do RDS

- Permite conexões **apenas do ECS**, bloqueando qualquer tentativa externa.
- Apenas aceita tráfego na porta **5432 (PostgreSQL)**.

Security Group do MongoDB Atlas

- O MongoDB Atlas está na nuvem gerenciada, então configuramos permissões para aceitar conexões **somente do ECS**.
- No **MongoDbStack**, adicionamos a **IP Allowlist** para permitir conexões **apenas do cluster ECS**.

Proteção Adicional

Além de usar **Security Groups**, tomamos algumas medidas extras:

Uso de Secrets Manager: As credenciais do RDS e MongoDB Atlas não ficam hardcoded no código.

Subnets privadas: O RDS e ECS não estão acessíveis diretamente da internet.

Controle de tráfego no Load Balancer: Apenas ele recebe tráfego externo, protegendo os serviços internos.

NAT Gateway para o ECS: Permite saída para a internet sem expor os contêineres diretamente.

Resumo da Arquitetura de Segurança

O **Load Balancer recebe tráfego público** e redireciona para o ECS.

O **ECS só recebe tráfego do Load Balancer** e pode acessar o RDS e o MongoDB Atlas.

O **RDS só permite conexões do ECS** (porta 5432 fechada para o mundo).

O **MongoDB Atlas tem regras de IP Allowlist**, permitindo apenas conexões do ECS.

As **subnets privadas protegem os serviços críticos**, garantindo que fiquem inacessíveis externamente.

Com essa abordagem, temos **segurança, escalabilidade e performance** sem comprometer a acessibilidade da aplicação.