

# MVP - Tech challenge

---

Lucas Fernando das Neves Carvalho dos Santos

- Email: lfneves.dg@gmail.com
- RM: 350505
- Grupo 53

Repositorys:

- <https://github.com/lfneves/mvp>
  - <https://github.com/lfneves/infra-rds-terraform>
  - <https://github.com/lfneves/infra-eks-terraform>
  - <https://github.com/lfneves/infra-vpc-terraform>
- 

Infrastructure videos explaining:

Drive Part 1 and Part 2

<https://drive.google.com/drive/folders/1JQbQEVLdbQaywwlXT5w0eMuV1FTNScwk?usp=sharing>

Part 1 - AWS VPC, deploy AWS database RDS and AWS Cluster EKS

<https://youtu.be/FCwZ1W9Dc0s>

Part 2 - Application Deploy

<https://youtu.be/ZpGls-grP6I>

---

## Terraform AWS EKS Cluster Deployment

AWS Infra Terraform EKS

This project uses Terraform to automate the deployment of an Amazon Elastic Kubernetes Service (EKS) cluster on AWS. Amazon EKS is a managed Kubernetes service that simplifies the deployment, scaling, and operation of containerized applications using Kubernetes.

```
infra-eks-terraform
├── LICENSE
├── README.md
└── infra-eks-terraform
    ├── eks-cluster.tf
    ├── outputs.tf
    ├── providers.tf
    ├── variables.tf
    └── vpc.tf
```

```
└─ workstation-external-ip.tf
```

## Prerequisites

Before getting started, make sure you have the following prerequisites installed on your machine:

- [Terraform](#) (you can use `terraform --version` to check)
- [AWS CLI](#) configured with appropriate credentials
- [Kubectl](#) for interacting with the cluster
- [kubectl-aws-iam-authenticator](#) for authenticating with the EKS cluster
- Internet access

## Repository

1. Clone this repository:

```
git clone https://github.com/lfneves/infra-eks-terraform.git  
  
cd infra-eks-terraform
```

2. Automatically create a `delivery-eks-terraform.tfstate` file and deploy bucket on `delivery-terraform-s3` and provide the necessary variables:

```
region          = "us-east-1"  
cluster_name    = "delivery-cluster"  
node_instance_type = "t2.small"  
node_max_count   = 1  
node_min_count   = 1
```

---

## Terraform AWS RDS Database Deployment

### AWS Infra Terraform RDS

<https://github.com/lfneves/infra-rds-terraform>

**Terraform AWS RDS PostgreSQL Deployment** This project uses Terraform to automate the deployment of a single Amazon RDS instance with PostgreSQL on AWS. Amazon RDS (Relational Database Service) is a managed relational database service that makes it easy to deploy, operate, and scale databases.

```
infra-rds-terraform  
├─ LICENSE  
├─ README.md  
├─ infra-rds-terraform  
│  └─ main.tf
```

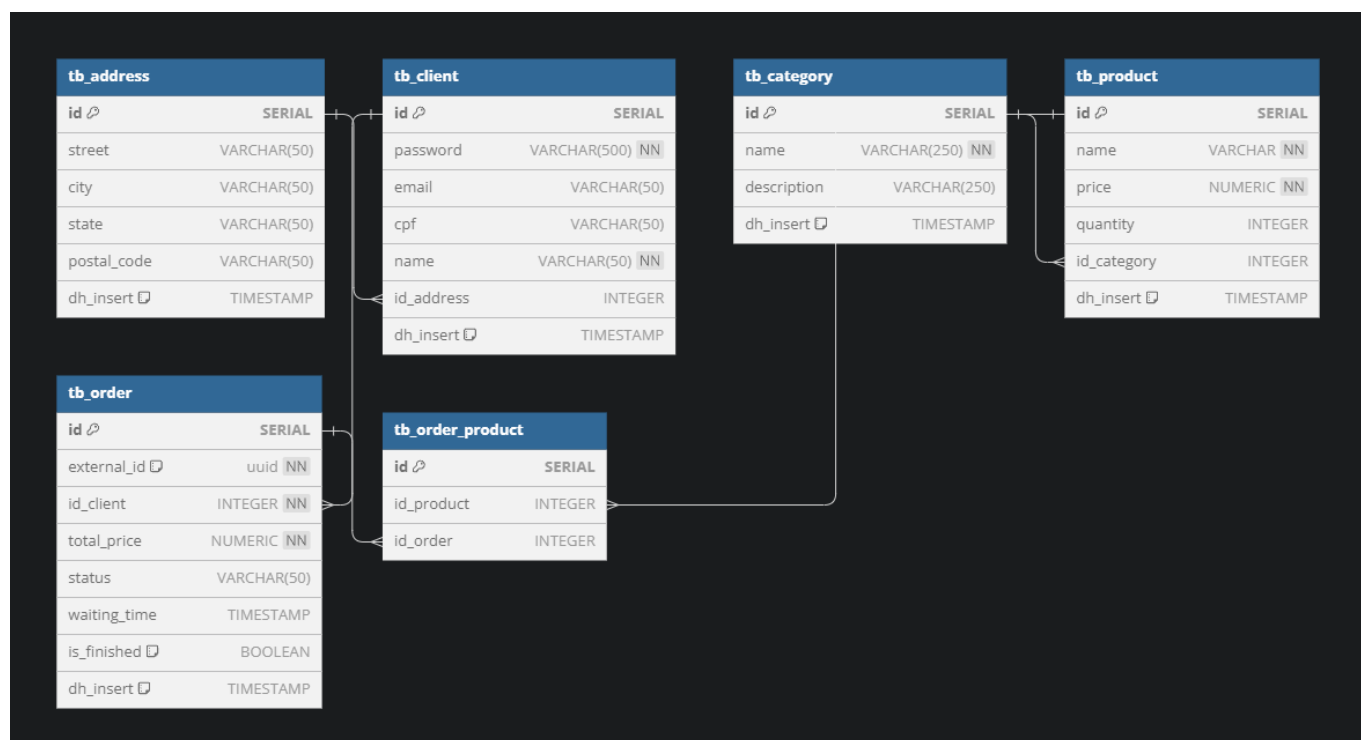
```
|— outputs.tf
|— table_schema.sql
|— variables.tf
|— versions.tf
```

## Justification for Choosing PostgreSQL for a Restaurant System

- **Robust Performance:** PostgreSQL is known for delivering solid performance, even in high transaction volume environments. This is crucial for a restaurant system where it's essential to process orders quickly and efficiently, ensuring a seamless customer experience.
- **Reliability and Stability:** PostgreSQL is renowned for its stability and reliability. Restaurant systems need a database that can handle continuous and critical workloads, minimizing the risk of unexpected failures that could disrupt business operations.
- **Flexible Data Model:** PostgreSQL supports a variety of data types and offers support for advanced features such as foreign keys, indexes, and stored procedures. This allows for flexible data modeling, covering everything from menus and orders to employee and customer information.
- **Advanced Query Capabilities:** PostgreSQL has a powerful SQL query engine that allows for efficient execution of complex queries. This is crucial for generating reports and data analysis that can help restaurant owners and managers make informed decisions.
- **Active Community and Support:** PostgreSQL has an active community of developers and users worldwide. This means you'll have access to ongoing technical support, updates, and security fixes. Additionally, many resources and plugins are available to customize the system to meet the restaurant's specific needs.
- **Cost-Effective:** PostgreSQL is an open-source database, which means it's a cost-effective option compared to many commercial database management systems. This can be especially advantageous for restaurants with budget constraints.
- **Integration and Scalability:** PostgreSQL is highly compatible with many programming languages and can be easily integrated with other parts of the restaurant system, such as online ordering apps, inventory management systems, and more. Furthermore, it is scalable, allowing the database to grow as the restaurant expands its operations.

In summary, PostgreSQL offers a solid combination of performance, reliability, flexibility, and cost-effectiveness, making it a sensible choice for a restaurant system that requires a robust and dependable database to meet critical business needs.

## Database Diagram



## Application mvp

<https://github.com/lfneves/mvp>

This is a **Spring Boot WebFlux** application using **Kotlin**.

Spring WebFlux utilizes the [Reactor](#) library, which is an implementation of Reactive Streams specs for building non-blocking applications.

This project:

- Uses [Reactor Netty as the default implementation](#) for testing purposes. To change to Apache Tomcat as the default Web container for Spring WebFlux, follow these steps.
- Utilizes functional endpoints.
- Employs the [PostgreSQL](#) database.

## Requirements

- Java 17 or later - [SDKMAN - Recommendation](#)
- Gradle 7.6.1 or later - [Gradle build tool Installation](#)
- Docker 24.0.2 or later - [How to install Docker](#)
- Docker Compose 1.29.2 or later - [Reference guide](#)
- Minikube v1.31.2 or later - [Get Started with Minikube](#)
- Helm v3.10.1 or later - [Installing Helm](#)
- The project runs on port 8099 (<http://localhost:8099>).

## Getting Started

```
# Get the latest version

git clone https://github.com/lfneves/mvp.git
```

## Project Structure

```
main
├── kotlin
│   ├── com
│   │   ├── mvp
│   │   │   ├── delivery
│   │   │   │   ├── DeliveryApplication.kt
│   │   │   │   ├── application
│   │   │   │   ├── domain
│   │   │   │   ├── infrastructure
│   │   │   │   └── utils
│   └── resources
│       ├── application.yml
│       ├── database
│       │   ├── 1_create_tables.sql
│       │   └── 2_inserts_category.sql
```

## Prerequisites

Check versions:

- Java 17+

```
java --version
```

- Docker

```
docker -v
```

- Docker Compose

```
docker-compose --version
```

## Installation

This is an example of how to use the software and how to install it.

## Docker

In the main project directory:

Docker build and start applications:

```
$ docker-compose up --build
```

Or use:

```
$ docker-compose up -d --build
```

To recreate the application in case of problems, use the command:

```
$ docker-compose down
```

---

## Kubernetes (k8s)

**To initiate Kubernetes applications, execute the commands found within the "k8s" folder.**

```
$ kubectl apply -f delivery/k8s/postgres/.
```

```
$ kubectl apply -f delivery/k8s/application/.
```

**o access the application URL, use the following command:**

```
$ minikube service delivery --url
```

**Example output:**

```
http://192.168.49.2:32000
```

Inside the "k8s" folder, you will discover ".yaml" files utilized to deploy databases and applications within Kubernetes.

```

/delivery/k8s
├── application
│   ├── 1-deployment.yaml
│   ├── 2-service-load-balancer.yaml
│   ├── 3-hpa.yaml
│   └── 4-ingress.yaml
└── postgres
    ├── 1-db-persistent-volume.yaml
    ├── 2-db-volume-claim.yaml
    ├── 3-db-configmap.yaml
    ├── 4-db-secret.yaml
    ├── 5-db-deployment.yaml
    └── 6-db-service.yaml

```

## Metric Server

```
$ minikube addons enable metrics-server
```

To monitor the Horizontal Pod Autoscaler, employ the following command:

```
$ kubectl get hpa
```

NAME	REFERENCE	TARGETS	MINPODS	MAXPODS	REPLICAS	AGE
horizontalpodautoscaler.autoscaling/delivery	Deployment/delivery	55%/80%	2	4	2	51m

## Kubernetes (k8s) - Install with Helm

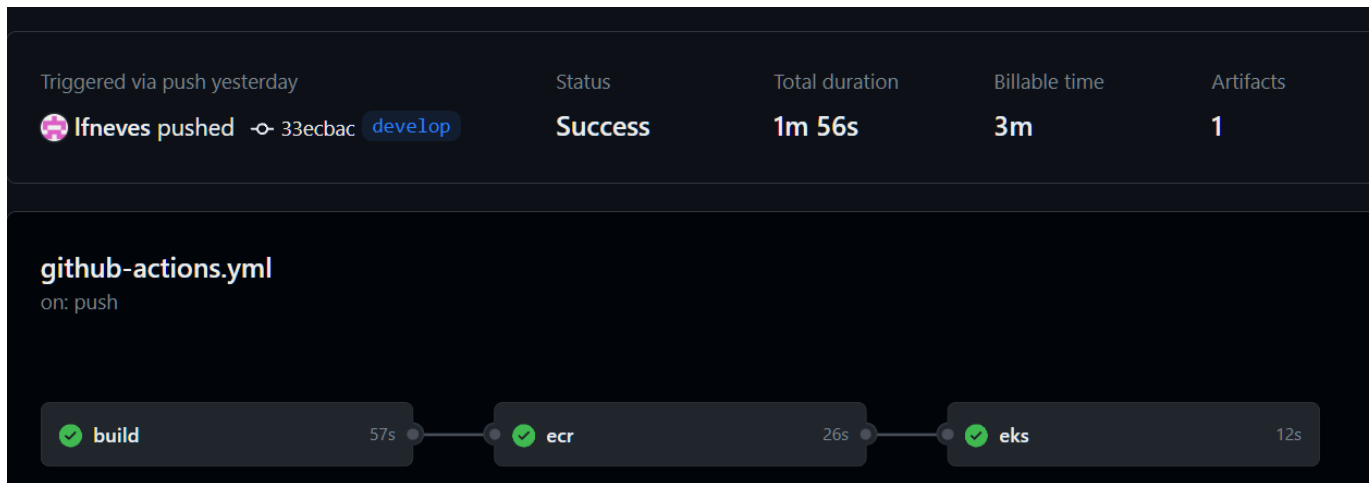
[BETA] Because this hasn't been implemented following best practices.

```
$ helm install deliveryhelm deliveryhelm/
```

## Helm uninstall

```
$ helm uninstall deliveryhelm deliveryhelm/
```

## Deploy Github-actions



Triggered via push yesterday	Status	Total duration	Billable time	Artifacts
lfneves pushed <a href="#">develop</a>	Success	1m 56s	3m	1

**github-actions.yml**  
on: push

build 57s — ecr 26s — eks 12s

---

## Integration Mercado Pago

For the webhook checkout process, generate a QR code.

For testing full process with Mercado Pago webhook, use [hookdeck.com](https://hookdeck.com) with CLI to change the order status in the localhost application.

Appllication path **/api/v1/mp-order/qr-code-checkout** creates a checkout with Mercado Pago.

Example:

```
{
  "in_store_order_id": "75ca8fe9-3b1a-4053-8f3e-49a62e91f8e8",
  "qr_data": "00020101021243650016COM.MERCADOLIBRE02013063675ca8fe9-3b1a-4053-8f3e-49a62e91f8e85204000053039865802BR5908delivery6009SA0PAUL062070503***63042BFA"
}
```

---

This project uses [CommandLineRunner](#)

- CommandLineRunner is used to create a default user, products and categories on start application startup.
- Default login :

**/api/auth/login-token**

```
{
  "username": "999999999999",
  "password": "123"
}
```

---

The best way to use it as a suggestion is by using [Postman](#)



## A collection is available preconfigured in the project root

MVP - Pos tech delivery application.postman\_collection.json

- This project uses user and session control for access
- Endpoints without control access ***"/api/auth/\*", "/api/v1/users/signup"***

### Create new user example:

http://localhost:8099/api/v1/users/signup

Body:

```
{
  "name": "Admin",
  "email": "admin@email.com",
  "cpf": "99999999999",
  "password": "admin",
  "address": {
    "street": "rua 1",
    "city": "sp",
    "state": "sp",
    "postalCode": "1234"
  }
}
```

**Login - Use the username (cpf) and password, then copy the token and use it in authenticated endpoints.**

http://localhost:8099/api/auth/login-token

```
{
  "username": "99999999999",
  "password": "admin"
}
```

Response:

```
{
  "token":
  "eyJhbGciOiJIUzI1NiJ9.eyJpZENSaWVudCI6IjAiLCJ1c2VybmFtZSI6IjEyMzQ0TEyIiwic3ViIjoiaMTIzNDU2Nzg5MTIiLCJpYXQiOiJlE2ODgwOTI1NTAsImF1ZCI6Im5vLWFwcGxpY2F0aW9uLW5hbWUiLCJleHAiOiJlE2ODgwOTQwMDB9.HagYPqukwOML30Yad8sRj lnE0Gsy-5tGUSC72S-xyfU"
}
```

💡 To make it easier use environment variables

Place the command in the test tab on `/api/auth/login-token`

```
pm.environment.set("token", pm.response.json().token);
```

Example:

The image shows two screenshots of the Postman interface. The top screenshot shows a POST request to `{{baseUrl}}/api/auth/login-token` with the 'Tests' tab selected. The test script contains the line: `pm.environment.set("token", pm.response.json().token);`. The bottom screenshot shows a GET request to `{{baseUrl}}/api/v1/users/get-by-username` with the 'Authorization' tab selected. The authorization type is 'Bearer Token' and the token value is `{{token}}`. A note states: 'The authorization header will be automatically generated when you send the request. Learn more about [authorization](#)'.

This project also uses OpenAPI Specification ([Swagger](#)).

To access swagger use the URL:

`http://localhost:8099/swagger-ui.html` or `http://localhost:8099/webjars/swagger-ui/index.html`

localhost:8099/webjars/swagger-ui/index.html

Swagger.  
Supported by SMARTBEAR

v3/api-docs

Explore

MVP - Pos tech delivery application

v3/api-docs

Servers

http://localhost:8099 - Generated server url

Authorize

Usuários

PUT

/api/v1/users/update-user/{id}

Atualiza Usuário por id

POST

/api/v1/users/signup

Cadastro de Usuário

GET

/api/v1/users/{id}

Busca Usuário por id

DELETE

/api/v1/users/{id}

Deleta Usuário por id

GET

/api/v1/users/get-by-username

Busca Usuário por username(CPF)

Produtos Administrador

PUT

/api/v1/product/update/{id}

Atualiza produto pelo id

POST

/api/v1/product/create

Cria produtos

GET

/api/v1/product

Busca todos produtos

Produtos Administrador		
PUT	/api/v1/product/update/{id}	Atualiza produto pelo id
POST	/api/v1/product/create	Cria produtos
GET	/api/v1/product	Busca todos produtos
Admin Pedidos		
PUT	/api/v1/order/update-order-status/{id}	Atualiza o status do pedido
PUT	/api/v1/order/finish-order	Finaliza o pedido atualizando os status
GET	/api/v1/order	Busca todos pedidos
Pedidos		
PUT	/api/v1/order/fake-checkout	Efetua o pagamento atualizando os status
PUT	/api/v1/order/add-new-product-to-order	Adiciona item(s) ao pedido
POST	/api/v1/order/create-order	Inicia um pedido
GET	/api/v1/order/{id}	Busca pedido id
DELETE	/api/v1/order/{id}	Remove pedido pelo id
GET	/api/v1/order/all-products-by-order-id/{id}	Busca todos os produtos id do pedido
DELETE	/api/v1/order/remove-product-order	Remove produto de um pedido
Autorização		
POST	/api/auth/token-by-application	Autenticação clientID + secretID
POST	/api/auth/login-token	Autenticação usuário + senha
Administrador de Usuários		
GET	/api/v1/users	Busca todos usuários
Performace Usuários		
GET	/api/v1/users/flux	Performace Usuários
Produtos		
GET	/api/v1/product/{id}	Busca produtos pelo id
DELETE	/api/v1/product/{id}	Remove produto pelo id
GET	/api/v1/product/get-by-category-name	Busca produtos por categoria
Produtos Performace		
GET	/api/v1/product/flux	Performace produtos
Pedidos Performace		
GET	/api/v1/order/flux	Performace Pedidos

## Roadmap

- ☒ Improve README.md
- ☒ Update order add paid status and adjusting service
- ☒ Implementation Helm
- Improvements

- ☒ Refactor admin services and repository to new package
  - ☒ Fix create order exceptions
  - ☒ Mercado Pago Qr code checkout
  - ☒ Refactor scripts database
- 

## License

Distributed under the MIT License. See LICENSE.txt for more information.