Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)
Escola Politécnica

Disciplina de Engenharia de Software II
Trabalho Final — Implementação e implantação de microsserviços

Aluno: Ricardo B. Süffert.
Professor: Júlio H. P. Machado.
Data de entrega: 26/06/2024.

1. **<u>Modelagem do banco de dados</u>:** Para o banco de dados, foi utilizado o SGBD em memória H2. O diagrama entidade-relacionamento (ER) dos bancos de dados utilizados pelos microsserviços de cadastramento e de pagamentos é disponibilizado na Imagem I.

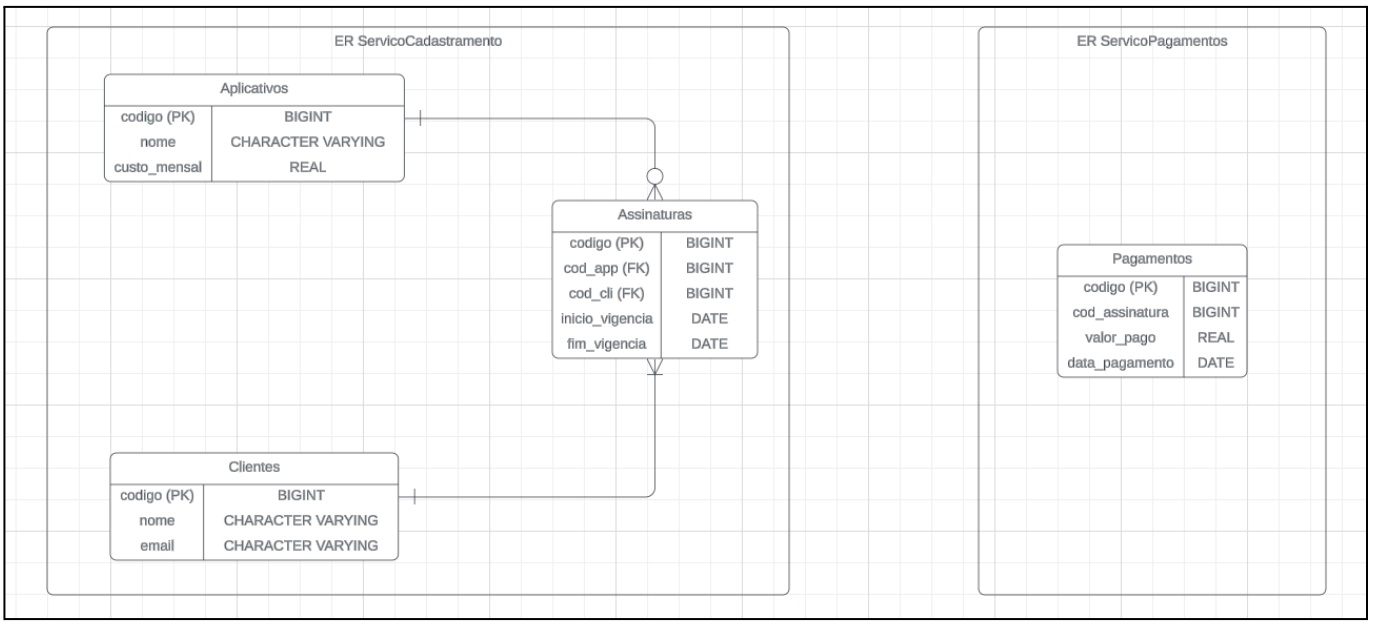

Imagem I – Diagrama ER dos bancos de dados do sistema

2. **<u>Implantação e componentes do sistema</u>:** O diagrama de implantação do sistema é disponibilizado na Imagem II, e a Imagem III ilustra o diagrama de componentes do sistema.

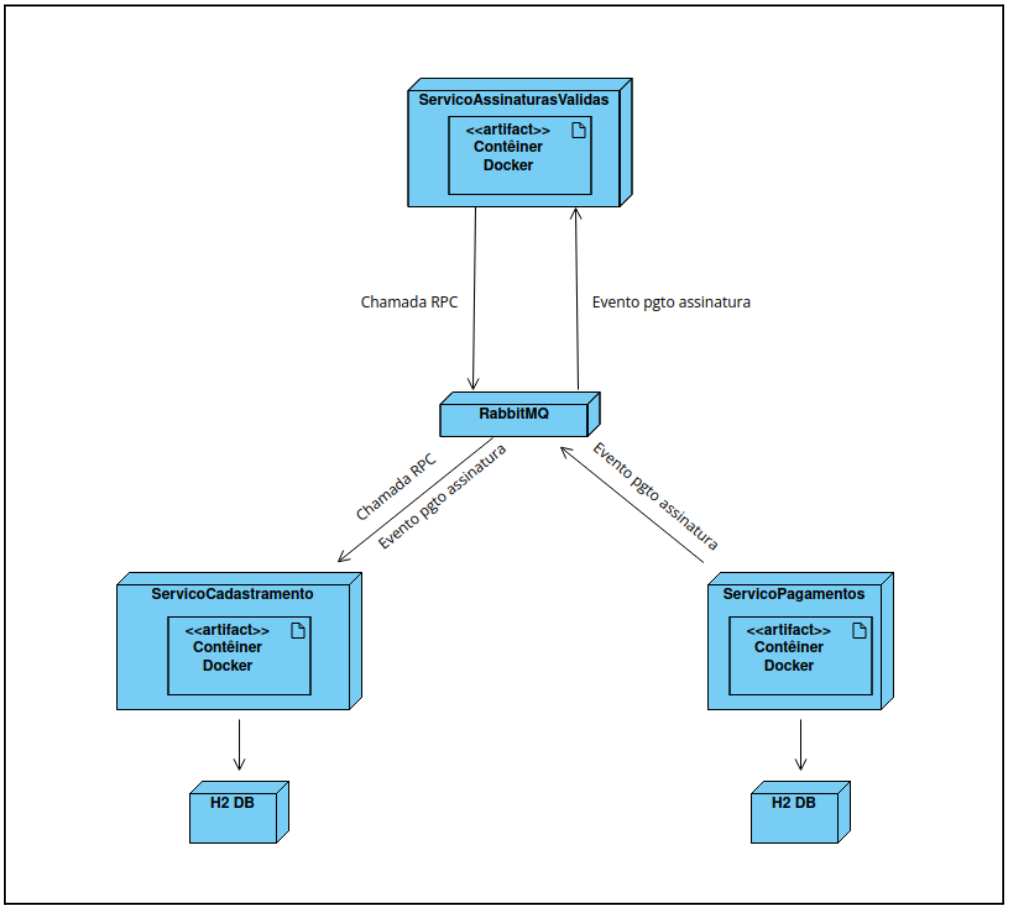

Imagem II – Diagrama de implantação do sistema

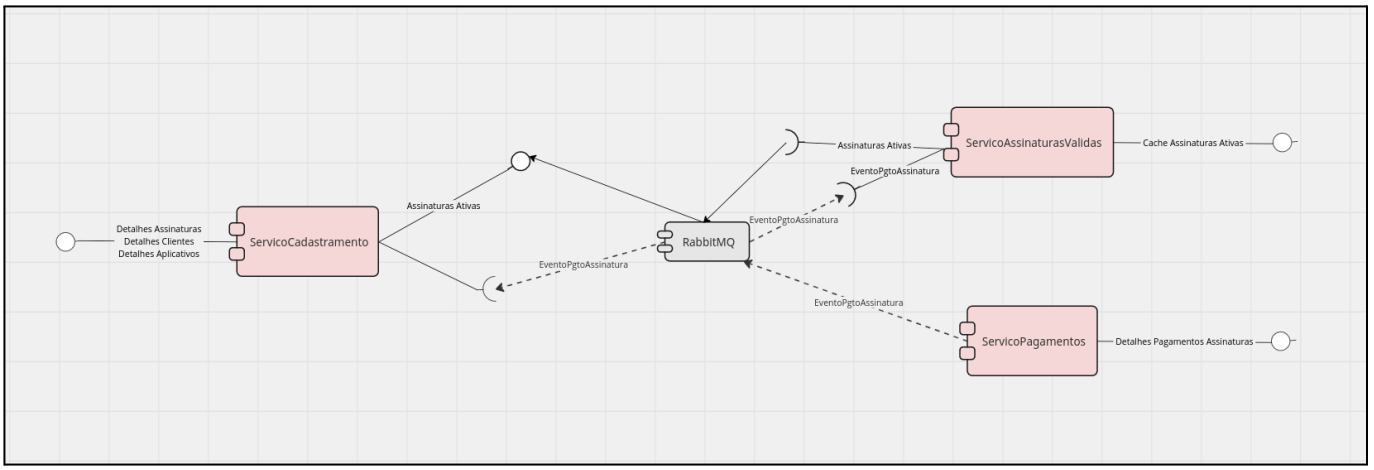

Imagem III – Diagrama de componentes do sistema

Ademais, o seguinte passo a passo em Markdown explica como o sistema foi implantado na AWS por meio da AWS CLI.

```
# Instructions for deploying the system to AWS

For deploying the application to AWS, we'll be using the **AWS CLI** on Linux. Make sure you have that installed before following the next steps.

## Pushing Docker images to the cloud

1. Build the Docker images of the three microsservices by running `docker build -t <image_name> .` at the root directory of each of the microsservices (or replace the `.` with the path to the Dockerfile).
2. Define the following environment variables so the AWS CLI knows your credentials to access your AWS account.

```bash
export AWS_ACCESS_KEY_ID=<your_access_key_id>
export AWS_SECRET_ACCESS_KEY=<your_secret_access_key>
export AWS_SESSION_TOKEN=<your_session_token>
```

3. Create an **Elastic Container Registry (ECR)** instance for each microsservice. The following command creates an ECR instance. After running it, a JSON response will be printed.

`aws ecr create-repository --repository-name <repository_name>`

4. On the **AWS Management Console**, open the ECR service, click on the name of the repository, click on "Permissions" on the left-hand panel, select "Edit JSON policy" and replace the lines with the JSON below. After that click "Save". Do that for each ECR instance created.
```

```json
{
 "Version": "2008-10-17",
 "Statement": [
   {
     "Effect": "Allow",
     "Principal": "*",
     "Action": "ecr:*"
   }
 ]
}
```

5. Back to your terminal, set a temporary variable with your Account ID:

`account_id=$(aws sts get-caller-identity | grep Account | cut -d '"' -f4)`

6. Tag the Docker image of each microsservice with the ECR repository URI. The following command does that for one image.

`docker tag <image_name>:latest
$account_id.dkr.ecr.<region>.amazonaws.com/<repository-name>:latest`

7. Log Docker into your AWS account with the following command:

`aws ecr get-login-password --region <region> | docker login --username AWS
--password-stdin <aws_account_id>.dkr.ecr.<region>.amazonaws.com`

8. Push the Docker images of each microsservice to its respective ECR repository. The following command does that for one image.

`docker push
$account_id.dkr.ecr.<region>.amazonaws.com/<repository_name>:latest`

Running the Docker images in the cloud

By now, you should have the Docker images of your microsservices in the cloud, each in a separate ECR container. Now, in order to execute them, follow the instructions below.

1. Create an **Elastic Container Service (ECS)** cluster by running the command below.

`aws ecs create-cluster --cluster-name <cluster_name>`

2. Create a task definition for each of the microsservices of the system with the command below.

```bash
aws ecs register-task-definition --cli-input-json '{
 "family": "<any_name_for_your_task_definition>",
 "containerDefinitions": [
   {
     "name": "<any_name_for_the_container_the_task_will_run>",
     "image": "<the_uri_of_the_image_in_the_ecr_instance>",
     "memory": <memory_in_MiB>,
     "cpu": <vCPU_units>,
     "portMappings": [
       {
         "containerPort": <port_number>,
         "hostPort": <port_number>,
         "protocol": "tcp"
       }
     ]
   }
 ]
}'
```

3. Launch EC2 instance(s) for your microsservices following the steps below.
   - Generate an SSH key pair: `aws ec2 create-key-pair --key-name <key_name> --query 'KeyMaterial' --output text > <key_name>.pem` and `chmod 400 <key_name>.pem`;
   - Create the EC2 instance: `aws ec2 run-instances --image-id <ami_id> --instance-type <instance_type> --key-name <key_pair_name>`;
      - Depending on the characteristics of your tasks, you may create a single EC2 instance whose instance type has enough resources for the most resource-consuming task, and ECS will manage the scheduling of the services (you'll create them from the task definitions later) in the given pool of EC2 instances.
   - Check the ID of the security group associated with the new instance: `aws ec2 describe-instances --instance-ids <instance_id> --query "Reservations[].Instances[].SecurityGroups[*].GroupId" --output text`
   - Modify the security group to allow SSH inbound traffic: `aws ec2 authorize-security-group-ingress --group-id <security_group_id> --protocol tcp --port 22 --cidr <your_public_ip_address>/32`;
   - Modify the security group to allow inbound traffic to your application: `aws ec2 authorize-security-group-ingress --group-id <security_group_id> --protocol tcp --port <your_service_port> --cidr 0.0.0.0/0`

```
   - Before moving on, make sure your instances are running: `aws ec2
describe-instances --instance-ids <instance_id> --query
"Reservations[*].Instances[*].State.Name" --output text`

4. Register the EC2 instances created on step 3 to the ECS cluster you created
on step 1 to make them available to run tasks.
   - Get the public IP address of the new EC2 instance: `aws ec2
describe-instances --instance-ids <instance_id> --query
"Reservations[*].Instances[*].PublicIpAddress" --output text`;
   - Connect to the instance through SSH: `ssh -i <key_name>.pem
ec2-user@<public_ip>`;
   - Update system packages: `sudo yum update -y`;
   - Install the ECS agent: `sudo yum install -y ecs-init`;
   - Start the Docker daemon: `sudo service docker start`;
   - Configure the ECS cluster: `echo "ECS_CLUSTER=<cluster_name>" | sudo tee
/etc/ecs/ecs.config`;
   - Start and enable the ECS agent: `sudo service ecs start`;
   - Before moving on, exit the SSH connection and make sure the instance has
registered itself to the cluster: `aws ecs list-container-instances --cluster
<cluster_name>`.
      - If not, it is possible that you may need to change the IAM role of the
EC2 instance.

5. Run the task definitions as services.

`aws ecs create-service --cluster <cluster_name> --service-name <service_name>
--desired-count <count> --launch-type <type> --task-definition
<task_definition_name:revision>`

6. Check the deployment status:

`aws ecs describe-services --cluster <cluster_name> --services <service_name>`

## Additional steps

For production environments, you may also set up logging and monitoring, load
balancing, service registry etc.
```

3. **Vídeo de demonstração:** o vídeo está disponível no YouTube por meio deste link.
4. **Código-fonte:** O código-fonte do sistema, bem como os diagramas do sistema e demais artefatos — como coleção do Postman, documento Markdown com as instruções para implantar o sistema, entre outros — estão disponíveis no repositório GitHub do trabalho.