# Bootcamp Spring React 3.0 - Cap. 06

Docker, implantação, CI/CD

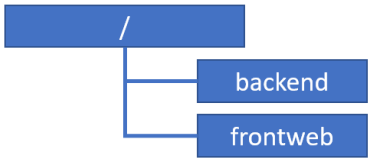
## Competências

- Docker
  - Conceitos
  - Comandos
  - Imagens e Dockerfile
  - Instanciação de containers
  - Volumes
  - DockerHub
- Implantação manual na AWS
  - EC2
  - RDS
  - Conectando remotamente
- CI/CD
  - Heroku
  - AWS
  - Github Actions
  - Stage de homologação
  - Elastic Beanstalk

## Material

https://github.com/devsuperior/dscatalog-resources/blob/master/backend/DEVOPS.md

## Pastas do repositório Git



# CI/CD padrão do Heroku

**1. Criar projeto no Heroku, provisionar Postgres e instanciar base de dados**

**2. Associar o projeto local ao Github e ao Heroku**

```
heroku -v
heroku login
heroku git:remote -a <nome-do-app>
git remote -v
```

**3. Arquivo system.properties**

```
java.runtime.version=11
```

**4. Configurar variáveis de ambiente no Heroku**

```
APP_PROFILE

CLIENT_ID
CLIENT_SECRET
JWT_SECRET
JWT_DURATION

DB_URL
DB_USERNAME
DB_PASSWORD
```

**5. Realizar deploy**

**Repositório comum:**

```
git push heroku main
```

**Monorepositório (subpasta):**

```
git subtree push --prefix backend heroku main
```

# CI/CD com AWS e Github Actions

## Serviços AWS

- IAM - usuários e permissões
- EC2 - máquinas
- Elastic Beanstalk - camada de abstração sobre o EC2
- RDS - banco de dados
- S3 - storage

## Pipeline ou esteira de deploy

Sequência automatizada de passos de implantação. Pode implementar integração contínua, entrega contínua e deploy contínuo.

## O que é CI/CD?

https://www.redhat.com/pt-br/topics/devops/what-is-ci-cd

# Primeiro teste Github Actions

**hello.yml**

```
name: HelloWorld

on:
  push:
    branches:
      - 'main'
jobs:
  build:
    runs-on: ubuntu-latest

    steps:
      - name: Printing Hello World
        run: echo "Hello World!"
```

# Passos / scripts para CI/CD com AWS e Github Actions

## ETAPA 1: Conexão

### Github Repository secrets

- Crie um usuário no IAM
- Configure secrets no repositório Github
  AWS_ACCESS_KEY_ID
  AWS_SECRET_ACCESS_KEY

### Cloud Shell

Evita o “Funciona na minha máquina”
https://console.aws.amazon.com/cloudshell/home

## ETAPA 2: REDE

### Identificador Único: UNIQ

Permite executar esses comandos várias vezes na mesma conta sem conflitar os nomes dos recursos, basta mudar o UNIQ

Também guardaremos as variáveis criadas em um arquivo “envrc”. Se em outro momento você precisar carregar as variáveis, basta rodar o comando: **`source envrc`**

```
export ENV_NAME="pocdscatalog"
export AWS_REGION="us-east-2"
export UNIQ="dsc${ENV_NAME}$(date +'%Y%m%d')"

echo export UNIQ=$UNIQ | tee envrc
echo export AWS_REGION=$AWS_REGION | tee -a envrc

cat envrc # shows environment variables
source envrc # loads environment variables if disconnected
```

## AWS Teste de Autenticação

```
aws sts get-caller-identity
```

## Criar AWS VPC - Virtual Private Cloud (rede)

```
export VPC_ID=$(aws ec2 create-vpc \
    --cidr-block 10.0.0.0/16 \
    --query "Vpc.VpcId" \
    --output text)

echo export VPC_ID=$VPC_ID | tee -a envrc

aws ec2 create-tags --resources $VPC_ID \
    --tags Key=Name,Value="vpc-$UNIQ"

aws ec2 modify-vpc-attribute \
  --enable-dns-hostnames \
  --vpc-id $VPC_ID

aws ec2 modify-vpc-attribute \
  --enable-dns-support \
  --vpc-id $VPC_ID
```

## Criar e conectar o internet gateway

```
export IGW_ID=$(aws ec2 create-internet-gateway \
    --query "InternetGateway.InternetGatewayId" \
    --output text)

echo export IGW_ID=$IGW_ID | tee -a envrc

aws ec2 attach-internet-gateway \
  --vpc-id $VPC_ID \
  --internet-gateway-id $IGW_ID
```

## Setup public Route Table

```
export RTB_ID=$(aws ec2 create-route-table \
    --vpc-id $VPC_ID \
    --query "RouteTable.RouteTableId" \
    --output text)

echo export RTB_ID=$RTB_ID | tee -a envrc

aws ec2 create-route \
    --route-table-id $RTB_ID \
    --destination-cidr-block 0.0.0.0/0 \
    --gateway-id $IGW_ID
```

## Public Subnets - availability zone A

```
export CIDR_A=10.0.200.0/24

export AZ_A=$(aws ec2 describe-availability-zones \
  --query 'AvailabilityZones[0].ZoneName'  \
  --output text)

export NET_A=$(aws ec2 create-subnet \
    --vpc-id $VPC_ID \
    --cidr-block $CIDR_A \
    --availability-zone $AZ_A \
    --query "Subnet.SubnetId" \
    --output text)

echo export NET_A=$NET_A | tee -a envrc

aws ec2 associate-route-table \
    --subnet-id $NET_A \
     --route-table-id $RTB_ID

aws ec2 modify-subnet-attribute  \
    --subnet-id $NET_A  \
    --map-public-ip-on-launch
```

## Public Subnets - availability zone B

```
export CIDR_B=10.0.201.0/24

export AZ_B=$(aws ec2 describe-availability-zones \
  --query 'AvailabilityZones[1].ZoneName'  \
  --output text)

export NET_B=$(aws ec2 create-subnet \
    --vpc-id $VPC_ID \
```

```
    --cidr-block $CIDR_B \
    --availability-zone $AZ_B \
    --query "Subnet.SubnetId" \
    --output text)

echo export NET_B=$NET_B | tee -a envrc

aws ec2 associate-route-table \
    --subnet-id $NET_B \
     --route-table-id $RTB_ID

aws ec2 modify-subnet-attribute  \
    --subnet-id $NET_B  \
    --map-public-ip-on-launch
```

### Verificando os recursos criados até aqui

```
cat envrc
```

## ETAPA 3: Banco de dados

**ATENÇÃO: confira se as variáveis dos recursos ainda estão em memória**

### Variáveis do banco RDS

```
export RDS_NETGRP=$UNIQ-netgrp
export RDS_NAME=$UNIQ-postgresql
export RDS_ROOT_USER=root
export RDS_ROOT_PASSWORD=Masterkey321
export RDS_PORT=5432
export RDS_CIDR=0.0.0.0/0
export RDS_DB=dscatalogdb
export RDS_STORAGE=20
export RDS_INSTANCE=db.t2.micro
export RDS_ENGINE=postgres
export RDS_ENGINE_VERSION=12

echo export RDS_ROOT_USER=$RDS_ROOT_USER | tee -a envrc
echo export RDS_ROOT_PASSWORD=$RDS_ROOT_PASSWORD | tee -a envrc
echo export RDS_NETGRP=$RDS_NETGRP | tee -a envrc
```

### RDS Networking

```
export RDS_SECG=$(aws ec2 create-security-group \
  --group-name dscatalog-rds-secgrp \
  --description "dscatalog-rds-secg" \
  --vpc-id $VPC_ID \
  --query "GroupId" \
```

```
  --output text)

echo export RDS_SECG=$RDS_SECG | tee -a envrc

export RDS_SECG_ID=$(aws ec2 describe-security-groups \
  --filter Name=vpc-id,Values=$VPC_ID
Name=group-name,Values=dscatalog-rds-secgrp \
  --query 'SecurityGroups[*].[GroupId]' \
  --output text)

echo export RDS_SECG_ID=$RDS_SECG_ID | tee -a envrc

aws ec2 authorize-security-group-ingress \
  --group-id $RDS_SECG \
  --protocol tcp \
  --port $RDS_PORT \
  --cidr $RDS_CIDR

aws rds create-db-subnet-group \
    --db-subnet-group-name $RDS_NETGRP \
    --db-subnet-group-description "dscatalog RDS Subnet Group" \
    --subnet-ids $NET_A $NET_B
```

## RDS Instance

```
export RDS_ID=$(aws rds create-db-instance \
  --db-name $RDS_DB  \
  --db-instance-identifier $RDS_NAME \
  --allocated-storage $RDS_STORAGE \
  --db-instance-class $RDS_INSTANCE \
  --engine $RDS_ENGINE \
  --engine-version $RDS_ENGINE_VERSION \
  --master-username $RDS_ROOT_USER \
  --master-user-password $RDS_ROOT_PASSWORD \
  --db-subnet-group-name  $RDS_NETGRP \
  --backup-retention-period 0 \
  --publicly-accessible \
  --vpc-security-group-ids $RDS_SECG \
  --query "DBInstance.DBInstanceIdentifier" \
  --output text)

echo export RDS_ID=$RDS_ID | tee -a envrc
```

## Aguarde até o banco estar disponível

```
aws rds wait db-instance-available --db-instance-identifier $RDS_ID &&
echo "Pronto"
```

### Obtendo a URL do banco criado

```
export RDS_ENDPOINT=$(aws rds describe-db-instances  \
  --db-instance-identifier $RDS_ID  \
  --query "DBInstances[0].Endpoint.Address"  \
  --output text)

export RDS_PORT=$(aws rds describe-db-instances  \
  --db-instance-identifier $RDS_ID  \
  --query "DBInstances[0].Endpoint.Port"\
  --output text)

export RDS_JDBC=jdbc:postgresql://$RDS_ENDPOINT:$RDS_PORT/$RDS_DB

echo export RDS_JDBC=$RDS_JDBC | tee -a envrc
```

### Verificando os recursos criados até aqui

```
cat envrc
```

### Administração do banco de dados

- Conecte ao banco usando sua ferramenta favorita
- Crie a estrutura das tabelas
- Execute o seed do banco

## ETAPA 4: Infraestrutura Elastic Beanstalk e S3

**ATENÇÃO: confira se as variáveis dos recursos ainda estão em memória**

### Variáveis salvas

```
export EB_APP=${UNIQ}ebapp
export EB_ENV=${UNIQ}ebenv
export EB_BUCKET=${UNIQ}versions
export EB_PROFILE=${UNIQ}insprofile
export EB_INSTANCE_TYPES=t2.micro,t3.micro
export EB_SPOT=false
export EB_TEMPLATE=${UNIQ}cfg
export EB_STACK="64bit Amazon Linux 2 v3.2.7 running Corretto 11"

echo export EB_APP=$EB_APP | tee -a envrc
echo export EB_ENV=$EB_ENV | tee -a envrc
echo export EB_BUCKET=$EB_BUCKET | tee -a envrc
echo export EB_PROFILE=$EB_PROFILE | tee -a envrc
echo export EB_INSTANCE_TYPES=$EB_INSTANCE_TYPES | tee -a envrc
```

```
echo export EB_SPOT=$EB_SPOT | tee -a envrc
echo export EB_TEMPLATE=$EB_TEMPLATE | tee -a envrc
```

### Criar aplicação Elastic Beanstalk

```
aws elasticbeanstalk create-application --application-name $EB_APP

aws elasticbeanstalk create-configuration-template \
    --application-name $EB_APP \
    --template-name $EB_TEMPLATE \
    --solution-stack-name "$EB_STACK"
```

### Security Roles / Instance Profiles

```
wget
https://raw.githubusercontent.com/devsuperior/dscatalog-resources/mast
er/aws/eb-ip-policy.json

wget
https://raw.githubusercontent.com/devsuperior/dscatalog-resources/mast
er/aws/eb-ip-trust.json

export EB_ROLE=${UNIQ}ebrole
export EB_POLICYNAME=${UNIQ}policy

aws iam create-role --role-name $EB_ROLE --assume-role-policy-document
file://eb-ip-trust.json

aws iam put-role-policy --role-name $EB_ROLE --policy-name
$EB_POLICYNAME --policy-document file://eb-ip-policy.json

aws iam create-instance-profile --instance-profile-name $EB_PROFILE

aws iam add-role-to-instance-profile --instance-profile-name
$EB_PROFILE --role-name $EB_ROLE
```

### Criar bucket S3

```
aws s3 mb s3://$EB_BUCKET
```

## ETAPA 5: Subir “build zero” manual para nuvem

```
./mvnw clean package
```

** Subir o bundle (jar) para local público na nuvem (Github, S3, etc.)*

**ATENÇÃO: confira se as variáveis dos recursos ainda estão em memória**

```
export BUNDLE_NAME=dscatalog-0.0.1-SNAPSHOT.jar
echo export BUNDLE_NAME=$BUNDLE_NAME | tee -a envrc
```

# ETAPA 6: Implantar “build zero” no Elastic Beanstalk

**ATENÇÃO: confira se as variáveis dos recursos ainda estão em memória**

## Copiar build para S3 e implantar no Elastic Beanstalk

```
wget
https://github.com/devsuperior/dscatalog-resources/raw/master/aws/buil
d0/dscatalog-0.0.1-SNAPSHOT.jar

export EB_VERSION="${UNIQ}v$(date +'%Y%m%d%H%M')"
export EB_VERSION_KEY=$EB_VERSION/dscatalog-0.0.1-SNAPSHOT.jar

aws s3 cp --quiet dscatalog-0.0.1-SNAPSHOT.jar
s3://$EB_BUCKET/$EB_VERSION_KEY

aws s3 ls s3://$EB_BUCKET/$EB_VERSION_KEY

aws elasticbeanstalk create-application-version \
    --application-name $EB_APP \
    --version-label $EB_VERSION \
    --source-bundle S3Bucket=$EB_BUCKET,S3Key=$EB_VERSION_KEY
```

# ETAPA 7: Criação do “environment” da aplicação Elastic Beanstalk

**ATENÇÃO: confira se as variáveis dos recursos ainda estão em memória**

## Preparar arquivo de opções

```
wget
https://raw.githubusercontent.com/devsuperior/dscatalog-resources/mast
er/aws/options.txt.env

sudo yum -y install gettext

rm options.txt
envsubst < options.txt.env > options.txt
cat options.txt
```

### Criar ambiente

```
export EB_CNAME="${UNIQ}ebcname$(date +'%Y%m%d%H%M')"

aws elasticbeanstalk check-dns-availability --cname-prefix $EB_CNAME

aws elasticbeanstalk create-environment \
    --cname-prefix $EB_CNAME \
    --application-name $EB_APP \
    --template-name $EB_TEMPLATE \
    --environment-name $EB_ENV \
    --version-label $EB_VERSION \
    --output json \
    --option-settings file://options.txt
```

***(acompanhe no dashboard do EB)***

## ETAPA 8: Configurar variáveis de ambiente adicionais da aplicação Elastic Beanstalk

**Nota: as variáveis de conexão com o banco já foram definidas em `--option-settings`**

**Configurations -> Software**

CLIENT_ID
CLIENT_SECRET
JWT_SECRET
JWT_DURATION

* Restart Application Servers

## ETAPA 9: Configurar environment secrets no Github

EB_APP
EB_BUCKET
EB_ENV
AWS_REGION
BUNDLE_NAME

## ETAPA 10: Incluir pipeline Github Actions

https://github.com/devsuperior/dscatalog-resources/blob/master/aws/main-to-homolog.yml

# Exterminando os recursos da AWS

source envrc

```
aws elasticbeanstalk terminate-environment --environment-name "$EB_ENV"

aws elasticbeanstalk delete-application --application-name "$EB_APP"

aws s3 rm "s3://$EB_BUCKET" --recursive

aws s3 rb "s3://$EB_BUCKET"

aws rds delete-db-instance --db-instance-identifier "$RDS_ID"
--skip-final-snapshot

aws rds  delete-db-subnet-group --db-subnet-group-name "$RDS_NETGRP"

aws ec2 delete-security-group --group-id "$RDS_SECG_ID"

aws ec2 delete-subnet --subnet-id "$NET_A"

aws ec2 delete-subnet --subnet-id "$NET_B"

aws ec2  delete-route-table --route-table-id "$RTB_ID"

aws ec2 delete-internet-gateway --internet-gateway-id "$IGW_ID"

aws ec2 detach-internet-gateway --internet-gateway-id "$IGW_ID" --vpc-id
"$VPC_ID"

aws ec2 delete-vpc --vpc-id "$VPC_ID"
```