Para criar o projeto em terraform para subir o aplicativo Java para a AWS, interligando-o a um banco de dados Aurora e criando a infraestrutura de rede e segurança, siga os seguintes passos:

1. Configure as credenciais da AWS na sua máquina para que o terraform possa se comunicar com a sua conta AWS. Você pode seguir as instruções da documentação oficial da AWS para configurar as credenciais: **https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-files.html**
2. Crie uma estrutura de diretórios para o seu projeto, como mencionado anteriormente.
3. No diretório "app", crie o arquivo Dockerfile com o seguinte conteúdo:

```
FROM openjdk:11-jre-slim

WORKDIR /app

COPY app.jar /app
COPY application.properties /app

CMD ["java", "-jar", "app.jar"]
```

Este Dockerfile é baseado na imagem openjdk:11-jre-slim, copia o arquivo app.jar e application.properties para o diretório /app e executa o comando "java -jar app.jar" para iniciar a aplicação.

1. No diretório "app", crie o arquivo application.properties com as seguintes configurações:

```
spring.datasource.url=jdbc:mysql://<DB_ENDPOINT>/<DB_NAME>
spring.datasource.username=<DB_USERNAME>
spring.datasource.password=<DB_PASSWORD>
```

Substitua <DB_ENDPOINT>, <DB_NAME> e <DB_USERNAME> pelo endpoint do banco de dados, o nome do banco de dados e o nome de usuário do banco de dados, respectivamente. O valor de <DB_PASSWORD> será fornecido posteriormente.

1. No diretório "db", crie o arquivo create_schema.sql com as seguintes instruções SQL:

```
CREATE TABLE IF NOT EXISTS users (
    id INT PRIMARY KEY,
    name VARCHAR(255)
);
```

Este arquivo cria a tabela "users" no banco de dados.

1. No diretório "terraform", crie o arquivo main.tf com o seguinte conteúdo:

```
provider "aws" {
  region = "us-east-1"
}

resource "aws_vpc" "vpc" {
  cidr_block = "10.0.0.0/16"
}

resource "aws_subnet" "subnet_public_1" {
  cidr_block = "10.0.1.0/24"
  vpc_id     = aws_vpc.vpc.id
  availability_zone = "us-east-1a"
}

resource "aws_subnet" "subnet_public_2" {
  cidr_block = "10.0.2.0/24"
  vpc_id     = aws_vpc.vpc.id
  availability_zone = "us-east-1b"
}

resource "aws_security_group" "sg_http" {
  name_prefix = "http"
  vpc_id      = aws_vpc.vpc.id

  ingress {
    from_port   = 80
    to_port     = 80
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  ingress {
    from_port   = 443
    to_port     = 443
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
```

```
}

resource "aws_security_group" "sg_db" {
  name_prefix = "db"
  vpc_id      = aws_vpc.vpc.id

  ingress {
    from_port       = 3306
    to_portingress {
    from_port       = 3306
    to_port= 3306
protocol = "tcp"
security_groups = [aws_security_group.sg_http.id]
}egress {
from_port = 0
to_port = 0
protocol = "-1"
security_groups = [aws_security_group.sg_http.id]
}tags = {
Name = "db-sg"
}
}resource "aws_db_subnet_group" "db_subnet_group" {
name = "db-subnet-group"
subnet_ids = [aws_subnet.subnet_public_1.id, aws_subnet.subnet_public_2.id]
}resource "aws_db_instance" "db_instance" {
identifier = "db-instance"
engine = "aurora-mysql"
engine_version = "5.7.mysql_aurora.2.08.0"
instance_class = "db.t2.micro"
name = "db"
username = "dbuser"
password = random_password.password.result
db_subnet_group_name = aws_db_subnet_group.db_subnet_group.name
vpc_security_group_ids = [aws_security_group.sg_db.id]
skip_final_snapshot = true
allow_major_version_upgrade = true
}resource "random_password" "password" {
length = 16
special = true
}resource "aws_security_group_rule" "http_inbound" {
security_group_id = aws_security_group.sg_http.id
type = "ingress"
from_port = 80
to_port = 80
protocol = "tcp"
cidr_blocks = ["0.0.0.0/0"]
}resource "aws_security_group_rule" "https_inbound" {
security_group_id = aws_security_group.sg_http.id
type = "ingress"
from_port = 443
to_port = 443
protocol = "tcp"
cidr_blocks = ["0.0.0.0/0"]
}resource "aws_security_group_rule" "db_inbound" {
```

```
security_group_id = aws_security_group.sg_db.id
type = "ingress"
from_port = 3306
to_port = 3306
protocol = "tcp"
cidr_blocks = [aws_subnet.subnet_public_1.cidr_block, aws_subnet.subnet_public_2.cidr_bloc
k]
}resource "aws_security_group_rule" "db_outbound" {
security_group_id = aws_security_group.sg_db.id
type = "egress"
from_port = 0
to_port = 0
protocol = "-1"
cidr_blocks = ["0.0.0.0/0"]
}resource "aws_instance" "app_instance" {
ami = "ami-0c94855ba95c71c99"
instance_type = "t2.micro"
key_name = "mykey"
vpc_security_group_ids = [aws_security_group.sg_http.id]
subnet_id = aws_subnet.subnet_public_1.idprovisioner "file" {
content = "${file("../app/target/app.jar")}"
destination = "/home/ec2-user/app.jar"
}provisioner "file" {
content = "${file("../app/application.properties")}"
destination = "/home/ec2-user/application.properties"
}provisioner "remote-exec" {
inline = [
"sudo amazon-linux-extras install java-openjdk11",
"sudo yum -y provisioner "remote-exec" {
inline = [
"sudo amazon-linux-extras install java-openjdk11",
"sudo yum -y install docker",
"sudo service docker start",
"sudo usermod -a -G docker ec2-user",
"sudo yum install -y git",
"sudo git clone https://github.com/seu_usuario/seu_projeto.git",
"cd seu_projeto",
"sudo docker build -t app .",
"sudo docker run -d -p 80:8080 app"
]
}
}output "public_ip" {
value = aws_instance.app_instance.public_ip
}output "db_endpoint" {
value = aws_db_instance.db_instance.endpoint
}output "db_username" {
value = aws_db_instance.db_instance.username
}output "db_password" {
value = aws_db_instance.db_instance.password
}Agora, é necessário executar os seguintes comandos no diretório onde se encontram os arqu
ivos do Terraform:
```

```
$ terraform init
$ terraform apply
```

O primeiro comando, `terraform init`, inicializa o diretório do Terraform, que baixa as dependências necessárias e prepara o ambiente para a execução dos comandos do Terraform. O segundo comando, `terraform apply`, executa o plano de provisionamento descrito nos arquivos do Terraform e cria os recursos na AWS. Durante a execução do `terraform apply`, o Terraform irá mostrar o plano de execução e solicitará a confirmação do usuário antes de executá-lo.

Ao final do processo, o Terraform irá exibir as informações de saída (`output`) definidas nos arquivos do Terraform, como o endereço IP público da instância EC2 e as informações de acesso ao banco de dados.

Com isso, você terá criado uma instância EC2 com uma aplicação Java em um contêiner Docker conectado a um banco de dados Aurora MySQL, tudo gerenciado pelo Terraform na AWS.