**Tutorial Projeto Integrador Academia Neon**

**Projeto Integrador (Curso DevOps) - Academia Neon**

Projeto Integrador para a Academia Neon DevOps - Grupo DevOpers - Utilizando Ansible, Docker, Jenkins, AWS.

**Sobre o Projeto Integrador**

Introdução: Será fornecido um APP feito em NodeJS que escreve e lê arquivos no S3. As configurações necessárias para esse app funcionar serão definidas por variáveis de ambiente (environment).

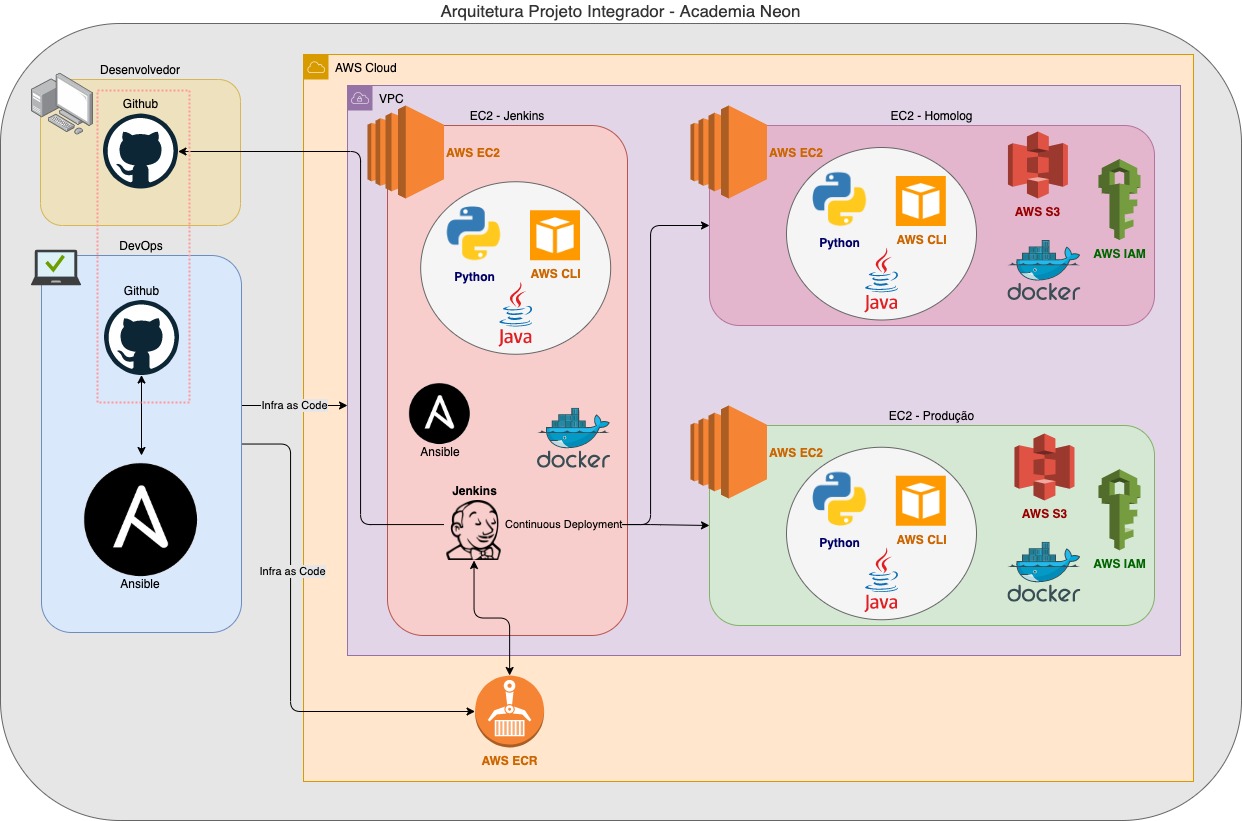
O objetivo: Criar as duas máquinas de app na AWS junto com uma máquina que conterá o Jenkins (para facilitar a evolução

fora da sala de aula). Com o jenkins no ar elas deverão construir as pipelines clonando o projeto, executando os testes e configurando

com as devidas variáveis de ambiente e por fim publicando no ambiente de destino: Prod ou Homolog.

Resultados Esperados: Com os aplicativos NodeJS no ar as alunas devem observar a url /healthcheck de cada um deles para ver se foram ou não configurados com sucesso. Deverão testar a ação de upload de imagens da aplicação que foi fornecida e se estiver com os tokens corretos do S3 fará o upload com sucesso.

**Arquitetura**



**Sobre este documento**

A intenção desse tutorial, além de falar sobre o projeto e as ferramentas utilizadas para a construção do mesmo, é ajudar no entendimento de como o projeto foi construído, explicando passo a passo do que foi feito e como foi feito, para que qualquer pessoa que esteja iniciando seus estudos seja capaz de entender e executar.

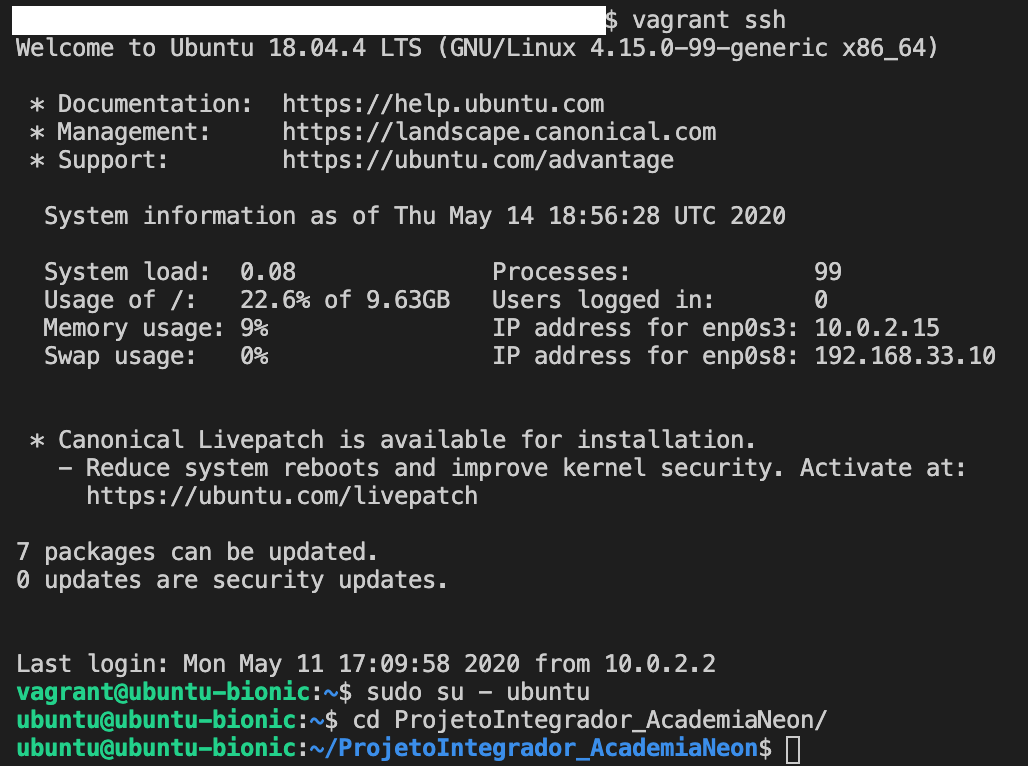
**Preparação Inicial**

Caso não tenha um ambiente de trabalho adequado por favor verificar o repositório [AcademiaNeon\_DevOps](https://github.com/andresavs/AcademiaNeon_DevOps). E somente após isso seguir os passos abaixo.

Para conseguirmos executar os playbooks via Ansible, precisamos configurar as credenciais de segurança em nosso repositório. Supondo que o clone deste projeto tenha sido feito, seguir os passos abaixo conforme descrito.

**1.** É necessário criar uma conta na [AWS](https://portal.aws.amazon.com/billing/signup#/start).

**IMPORTANTE**: Pensando que seu ambiente foi criado conforme orientações do item Preparação Inicial, sempre que for falado nesse readme desktop de trabalho e/ou vm é a máquina linux que é utilizada para criar e executar o nosso projeto, nunca seu computador pessoal, então garanta que seu terminal esteja logado nela, como dito anteriormente, no nosso caso é a vm criada com o vagrant e para garantir que todos os passos sejam executados com sucesso, o terminal precisa estar logado com o usuário ubuntu na vm ubuntu-bionic e dentro da pasta do Projeto:



**2.** Dentro do seu desktop de trabalho, **no nosso caso é a vm criada com o vagrant**, que vamos acessar via vscode, criar um arquivo para as senhas.

Para quem estiver familiarizado pode utilizar o vi para criar/editar o arquivo.

$ *vi ~/.ansible/.vault\_pass*

Ou então utilizar o echo e substituir o texto <"MinhaSenha"> pela "senha".

$ *echo <"MinhaSenha"> ~/.ansible/.vault\_pass*

**3.** Criar o arquivo aws\_credentials.yml dentro da pasta vars do repositório do projeto com as credenciais de segurança do usuário root da AWS.

Acessar o console da AWS com o user root e ir em:

My Security Credentials → Access keys (access key ID and secret access key) → Create New Access Key

**Importante**: Fazer o download das informações ou até mesmo copiar em algum arquivo no seu computador, pois vamos precisar dos dados no futuro e uma vez criado a informação do secret key não fica disponível mais.

Criar o arquivo e colocar as informações conforme abaixo:

AWSAccessKeyId: ABCDE\*\*\*\*\*\*\*

AWSSecretKey: aBCD123-\*\*\*\*\*\*\*

**4.** Encriptar o arquivo aws\_credentials.yml

$ *ansible-vault encrypt playbooks/vars/aws\_credentials.yml*

**5.** Caso queira validar se ficou ok executar os comandos abaixo.

Para visualizar conteúdo do arquivo

$ *cat playbooks/vars/aws\_credentials.yml*

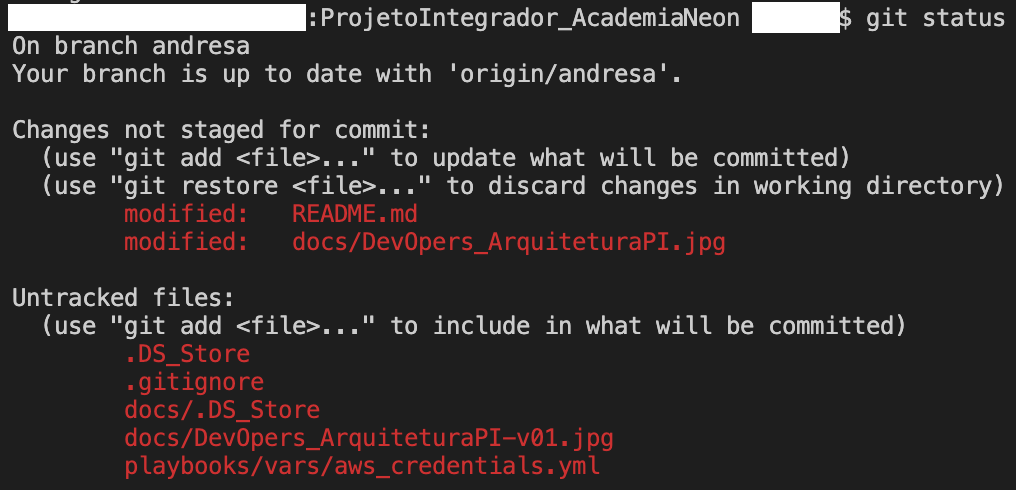
Para descriptografar o arquivo com nosso arquivo de senhas. O comando abaixo não tem quebra de linha.

$ *ansible-vault view playbooks/vars/aws\_credentials.yml --vault-password-file ~/.ansible/.vault\_pass*

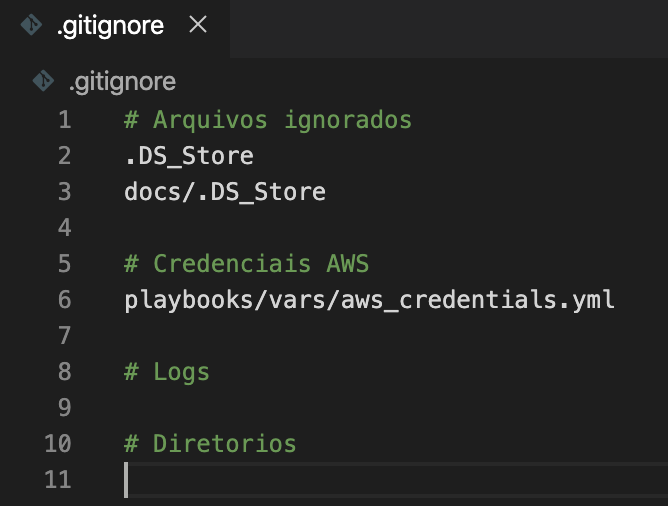
**Importante**: Mesmo criptografado evitar de subir esse arquivo aws\_credentials.yml no git. Para isso podemos utilizar o arquivo .gitignore.

**Utilizando arquivo .gitignore**

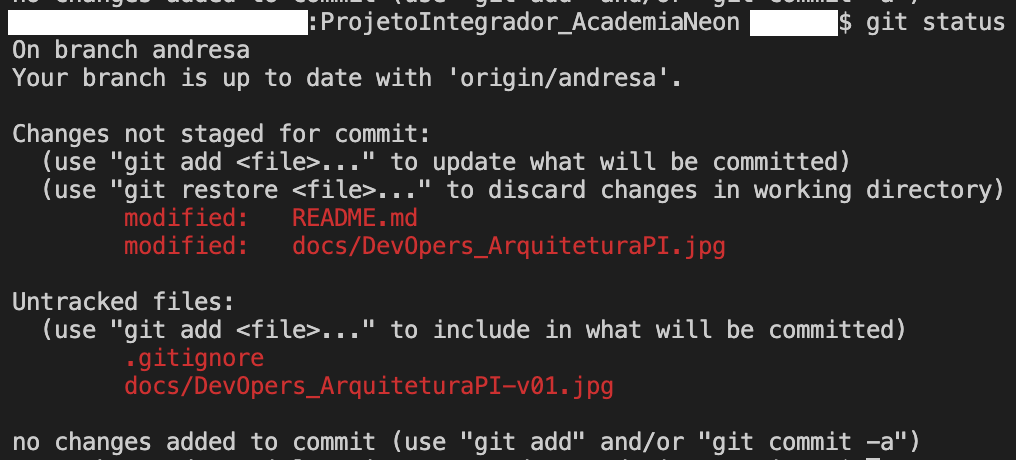
Para não efetuar commits de arquivos e ou pastas podemos criar o arquivo .gitignore na raiz do repositório de nosso projeto, antes de enviar as atualizados para o git, para ficar mais fácil e visual podemos "categorizar" os itens a serem ignorados. Uma dica para saber como deve colocar o nome dos arquivos/pastas dentro do .gitignore, é executar o comando git status e ver o que e como ele traz as informações, assim fica fácil saber como você deve colocar. Abaixo vou deixar um print de um arquivo .gitignore. Após realizar as alterações no repositório o primeiro comando seria o git status, ele vai mostrar todos os arquivos criados e alterados.



Preencha o arquivo gitignore com os arquivos que não deseja adicionar no seu repositório no git.



Faça o comando git status novamente e vai perceber que tudo que colocou dentro do arquivo .gitignore não aparece mais.



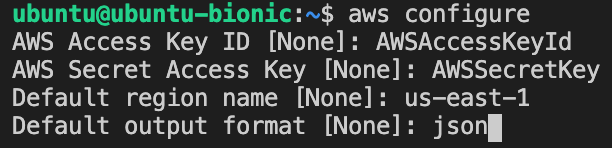
**Configurar AWS CLI no desktop de trabalho**

Para ter acesso aos recursos da aws via linha de comando em nosso desktop de trabalho é necessário configurar as credenciais do seu usuário. Caso queira, seguir os passos descritos abaixo:

Se você já realizou este passo quando criou a vm lá no item Preparação Inicial, não será necessário repetir.

**1.** Configurar o AWS CLI

$ *aws configure*



Preencher com as keys da AWS salvas anteriormente. Colocar a região de trabalho e o formato por padrão é json, mas você pode colocar também.

2. Alguns comandos para testar:

$ *aws iam list-access-keys*

$ *aws ec2 describe-regions*

**Entendendo e Customizando os Playbooks**

Caso queira apenas executar os playbooks para criar a Infra ir para o item Provisionando EC2 + S3 + IAM + ECR na AWS, que consta neste documento ou no [repositório do Git](https://github.com/andresavs/ProjetoIntegrador_AcademiaNeon/tree/andresa#provisionando-ec2--s3--iam--ecr-na-aws).

* **Falando um pouco sobre a estrutura para o ansible**:
  + inventory/hosts: Utilizada para definição dos hosts e algumas variáveis de ambiente importantes para eles. A variável abaixo é uma que precisa ser alterada de acordo com o arquivo aws.yml.

Ex.: *ansible\_ssh\_private\_key\_file=./dh-pi-devops-devopers-key.pem*

[Mais informações sobre inventory e hosts.](https://docs.ansible.com/ansible/2.3/intro_inventory.html)

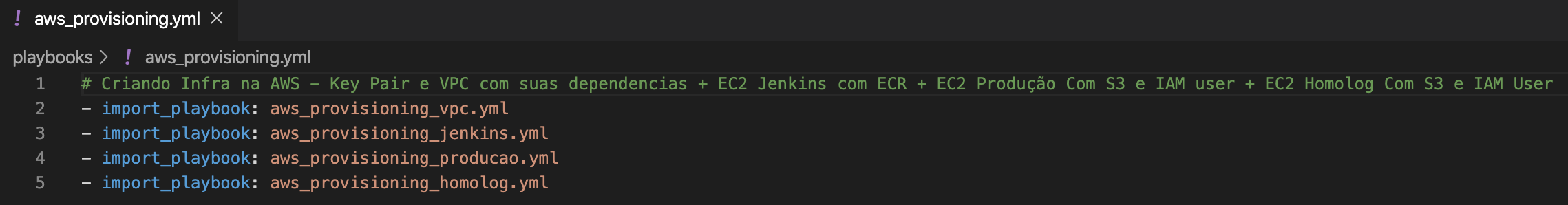
* + inventory/aws\_ec2.yml: Arquivo que contém a configuração da AWS para o comando ansible-inventory utilizado em nosso projeto.
  + ansible.cfg: É o arquivo de configuração do ansible, indica onde fica a pasta inventory por exemplo.

Ex.: *[defaults]*

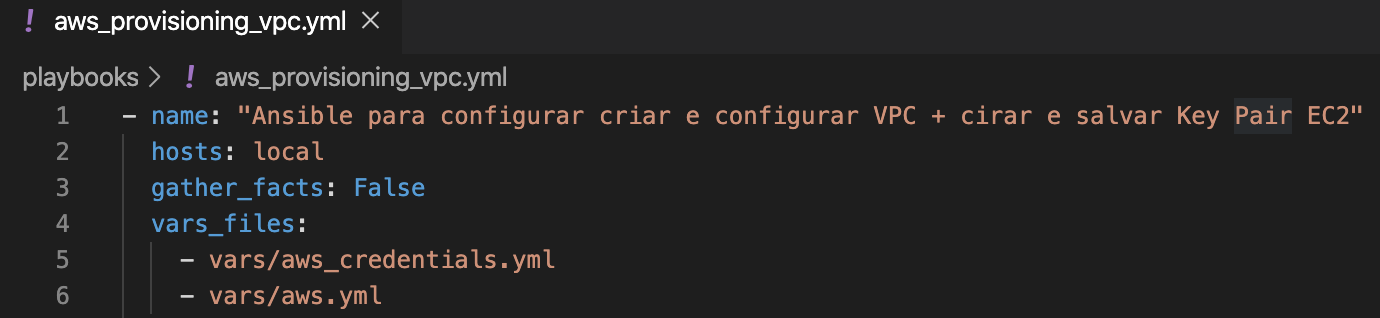
*inventory=inventory/*

[Mais informações sobre o arquivo ansible.cfg.](https://docs.ansible.com/ansible/latest/reference_appendices/config.html)

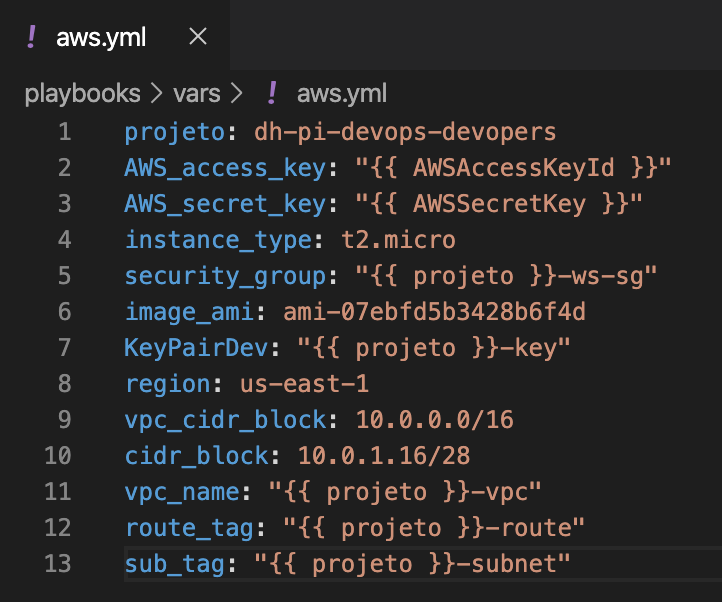
* + .gitignore: Como já falado este arquivo pode ser criado para ignorar itens que você vai atualizar no seu repositório git.
* **Falando um pouco sobre a criação da Infra na AWS**:

Nosso arquivo principal para criar a Infra na AWS é o aws\_provisioning.yml, mas ao abrir ele vou se vai se deparar com outros 4 arquivos dentro precedidos pelo comando import\_playbook. Foi construído dessa forma para que seja executado apenas um playbook que crie toda a Infra, mas que também fique segmentado e mais fácil para entender. 

* + Playbook aws\_provisioning\_vpc.yml



Item var\_files: Neste item temos dois arquivos de variáveis, um está armazenando nossas credenciais da AWS criptografadas (vars/aws\_credentials.yml) que criamos no item Preparação Inicial deste README, e a outro (vars/aws.yml) tem as variáveis que serão usadas nas tasks. Como sempre, segue um link da [Documentação Oficial](https://docs.ansible.com/ansible/2.5/user_guide/playbooks_variables.html) que fala mais sobre variáveis.



Breve comentário sobre cada item:

Linha 1: *projeto: dh-pi-devops-devopers* → Compõe as demais variáveis desse arquivo.

Linha 2: *AWS\_access\_key: "{{ AWSAccessKeyId }}"* → Credenciais AWS - está no arquivo criptografado no item Preparação Inicial vars/aws\_credentials.yml.

Linha 3: *AWS\_secret\_key: "{{ AWSSecretKey }}"* → Credenciais AWS - está no arquivo criptografado no item Preparação Inicial vars/aws\_credentials.yml.

Linha 4: *instance\_type: t2.micro* → Define o tipo da instância que será criado na AWS, para este projeto será uma t2.micro.

Linha 5: *security\_group: "{{ projeto }}-ws-sg"* → Define o nome do Security Group.

Linha 6: *image\_ami: ami-07ebfd5b3428b6f4d* → Define a imagem que vamos utilizar para criar a máquina na AWS.

Linha 7: *KeyPairDev: "{{ projeto }}-key"* → Define o nome da key pair. Este nome é usado no arquivo inventory/hosts.

Linha 8: *region: us-east-1* → Define a [região](https://docs.aws.amazon.com/pt_br/AWSEC2/latest/UserGuide/using-regions-availability-zones.html#concepts-available-regions) que vamos utilizar na AWS.

Linha 9: *vpc\_cidr\_block: 10.0.0.0/16* → Define o CIDR (Tamanho da VPC = quantidade de ips que ela vai ter) block da VPC.

Linha 10: *cidr\_block: 10.0.1.16/28* → Define o CIDR (Tamanho da Subnet = quantidade de ips que ela vai ter) block da Subnet.

Linha 11: *vpc\_name: "{{ projeto }}-vpc"* → Define o nome da VPC.

Linha 12: *route\_tag: "{{ projeto }}-route"* → Define o nome da Route Table (Faz parte da VPC).

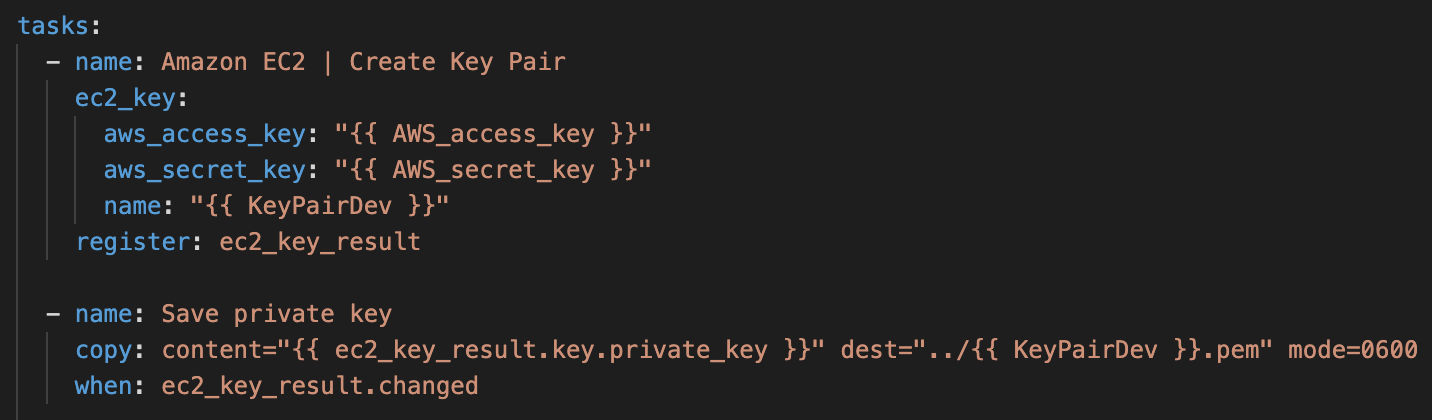
Linha 13: *sub\_tag: "{{ projeto }}-subnet"* → Define o nome da Subnet (Faz parte da VPC).

[Link útil sobre VPC e Subnets e CIDR.](https://docs.aws.amazon.com/vpc/latest/userguide/what-is-amazon-vpc.html)

[Link útil sobre VPC e Route Tables.](https://docs.aws.amazon.com/vpc/latest/userguide/VPC_Route_Tables.html)

* + Criando a Key Pair das instâncias EC2

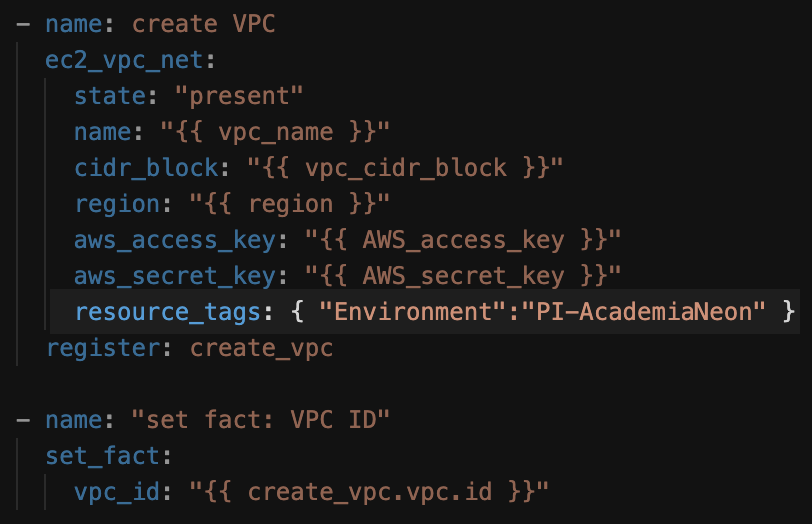
Apesar de ser um item das EC2s, ele ficou aqui pois é usado em todas as EC2 e é criado é baixado apenas uma única vez. Para saber mais sobre a EC2 Key Pairs acessar a [Documentação Oficial](https://docs.aws.amazon.com/pt_br/AWSEC2/latest/UserGuide/ec2-key-pairs.html).



[Link útil - módulo ansible key pair.](https://docs.ansible.com/ansible/latest/modules/ec2_key_module.html)

* + Criando a Amazon VPC - Amazon Virtual Private Cloud

Podemos dizer que a Amazon VPC é uma seção isolada logicamente dentro da nuvem da AWS onde você cria e executa os demais serviços. Para mais informações acessar a [Documentação Oficial](https://aws.amazon.com/pt/vpc/). Para criar a VPC direto no console da AWS é bem simples e intuitivo e muitos itens obrigatórios e dependentes são criados automaticamente, todos os itens "obrigatórios" foram criados em no aws\_provisioning\_vpc.yml playbook e podem ser identificados dentro do item **tasks → name**. Todos os itens que estão "{{ xxxxx }}" são variáveis que definimos em nosso arquivo **vars/aws\_credentials.yml** ou são "itens de configuração padrão". O único item que pode ser customizado direto no **aws\_provisioning\_vpc.yml** é o **resource\_tags: { "Environment":"PI-AcademiaNeon" }**, que é usado para identificar o ambiente daquela vpc, no nosso projeto, usamos PI-AcademiaNeon, pois ele se refere exatamente ao projeto integrador do Curso DevOps da Academia Neon. Esse item por ser mais segmentado ainda, por exemplo, na vpc criamos um ambiente de Desenvolvimento, e criamos mais de um subnet - segmentamos os ranges de ips para dois sub-ambientes (não é nosso caso hoje), onde teremos o Dev DB para banco de dados e o Dev App para a aplicação.



[Link útil - módulo ansible ec2-vpc.](https://docs.ansible.com/ansible/latest/modules/ec2_vpc_net_module.html)

[Link útil - módulo ansible ec2-subnet.](https://docs.ansible.com/ansible/latest/modules/ec2_vpc_subnet_module.html)

[Link útil - módulo ansible ec2-securitygroup.](https://docs.ansible.com/ansible/latest/modules/ec2_group_module.html)

[Link útil - módulo ansible ec2-internetgateway.](https://docs.ansible.com/ansible/latest/modules/ec2_vpc_igw_module.html)

[Link útil - módulo ansible ec2-routetable.](https://docs.ansible.com/ansible/latest/modules/ec2_vpc_route_table_module.html)

* + Playbooks aws\_provisioning\_jenkins, aws\_provisioning\_homolog e aws\_provisioning\_producao.

Nesses 3 playbooks são criados 3 (uma em cada um) [Amazon EC2](https://aws.amazon.com/pt/ec2/) - Amazon Elastic Compute Cloud, que pode ser considerada uma máquina virtual na nuvem, mas o conceito real dela é muito mais que isso.

1. aws\_provisioning\_jenkins.yml: A EC2 criada nesse playbook será o servidor Jenkins, por isso o nome. E também será criado um repositório [ECR](https://aws.amazon.com/pt/ecr/) - Amazon Elastic Container Registry, que é um serviço totalmente gerenciado que armazena imagens de conteiners do Docker, nele será armazenado e versionado as imagens que nosso pipeline irá gerar. Além do arquivo de variáveis também podemos definir variáveis específicas por playbooks, neste temos o nome que nossa EC2 terá, vide linha 8 e 9.

Ex.: *vars:*

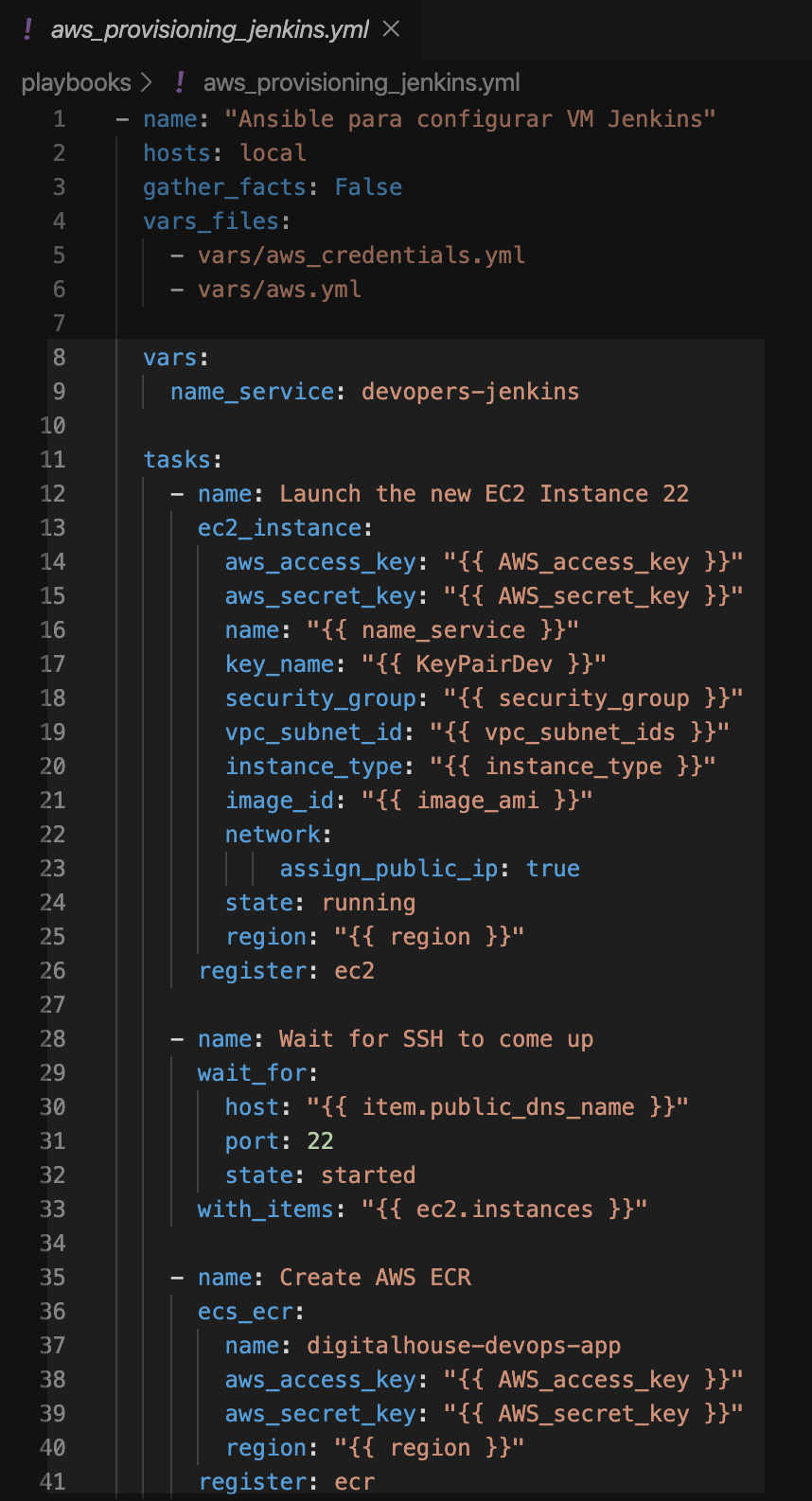
*name\_service: devopers-jenkins*

Temos as seguintes tasks nesse playbook:

*- name: Launch the new EC2 Instance 22* → Cria a instância EC2 de acordo com as variáveis do itens vars\_files e o vars.

*- name: Wait for SSH to come up* → Aguarda o serviço de SSH ficar disponível, ou seja, a terminar a configuração e máquina ficar running.

*- name: Create AWS ECR* → Cria o repositório de imagens de containers Docker, o nome é definido por \*name: digitalhouse-devops-app\*, e usamos o nome da imagem como nome do repositório. [Link útil - módulo ansible ecr.](https://docs.ansible.com/ansible/latest/modules/ecs_ecr_module.html)



1. aws\_provisioning\_homolog.yml: A EC2 criada nesse playbook será o servidor de Homologação, por isso o nome. E também será criado um usuário [IAM](https://aws.amazon.com/pt/iam/) - AWS Identity and Access Management, onde gerenciamos os acessos aos serviços e recursos da AWS, criando usuários, grupos, roles, entre outros. E um bucket [S3](https://aws.amazon.com/pt/s3/) - Amazon Simple Storage Service (Amazon S3), é um serviço de armazenamento de objetos resiliente, para o ambiente de Homologação. Além do arquivo de variáveis também podemos definir variáveis específicas por playbooks, neste temos o nome que nossa EC2 terá (vide linha 8 e 9), o usuário IAM (vide linha 10) e o nome do nosso bucket S3 (vide linha 11).

Ex.: *vars:*

*name\_service: devopers-homolog*

*name\_aim\_user\_s3: dh\_devopers\_homolog*

*name\_bucket\_s3: dh-devopers-homolog*

Temos as seguintes tasks nesse playbook:

*- name: Launch the new EC2 Instance 22* → Cria a instância EC2 de acordo com as variáveis do itens vars\_files e o vars.

*- name: Wait for SSH to come up* → Aguarda o serviço de SSH ficar disponível, ou seja, a terminar a configuração e máquina ficar running.

*- name: Create iam user "{{ name\_aim\_user\_s3 }}"* → Criar o usuário IAM e dá as permissões.

*- name: name: Create a buckets* → Cria o bucket S3 (apesar de criar esse módulo é utilizado mais para gerenciar os arquivos).

*- name: Create S3* → Cria o bucket S3 e realiza as configurações.

[Link útil - módulo ansible iam user.](https://docs.ansible.com/ansible/latest/modules/iam_user_module.html)

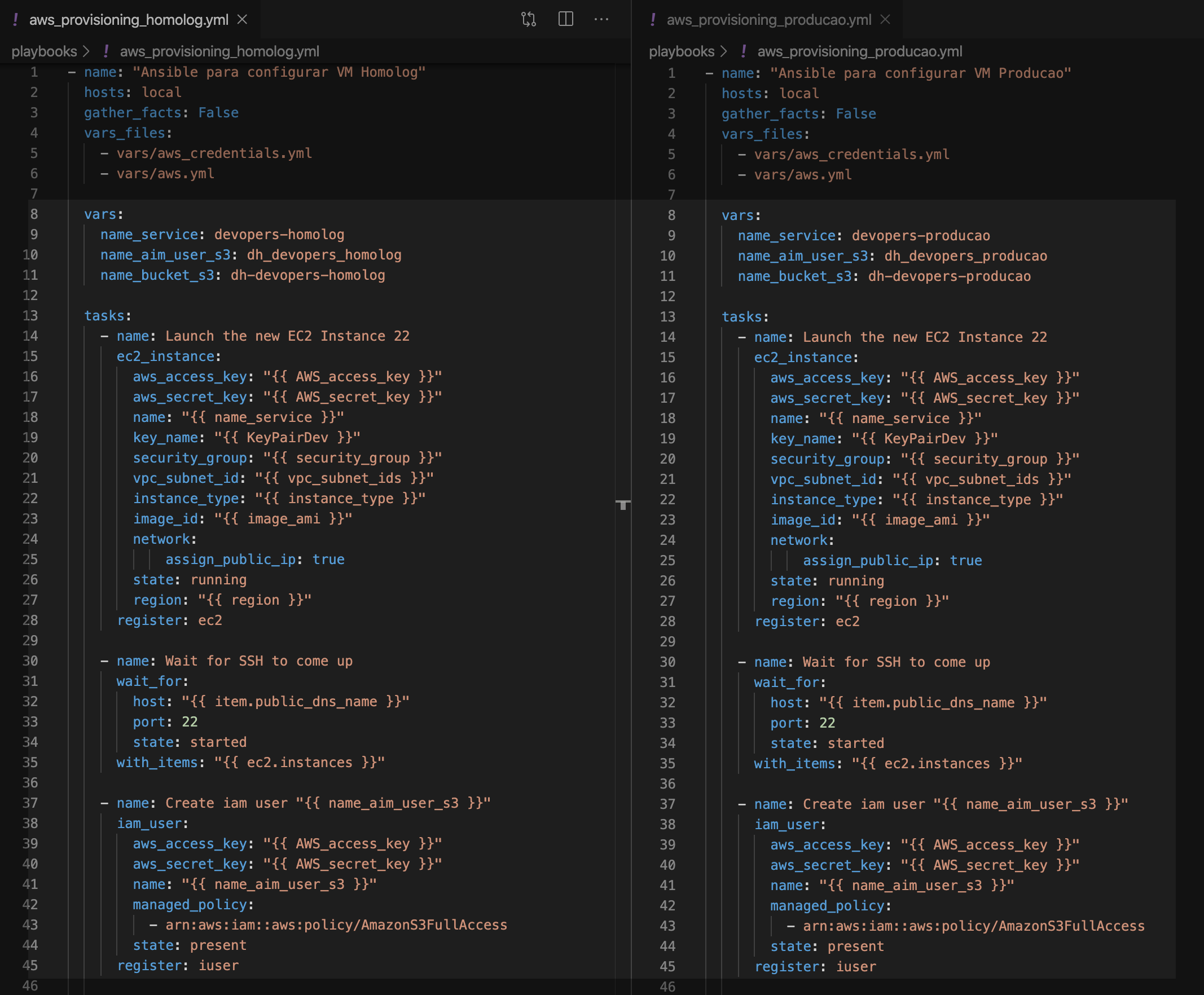
[Link útil - módulo ansible aws s3.](https://docs.ansible.com/ansible/latest/modules/aws_s3_module.html)

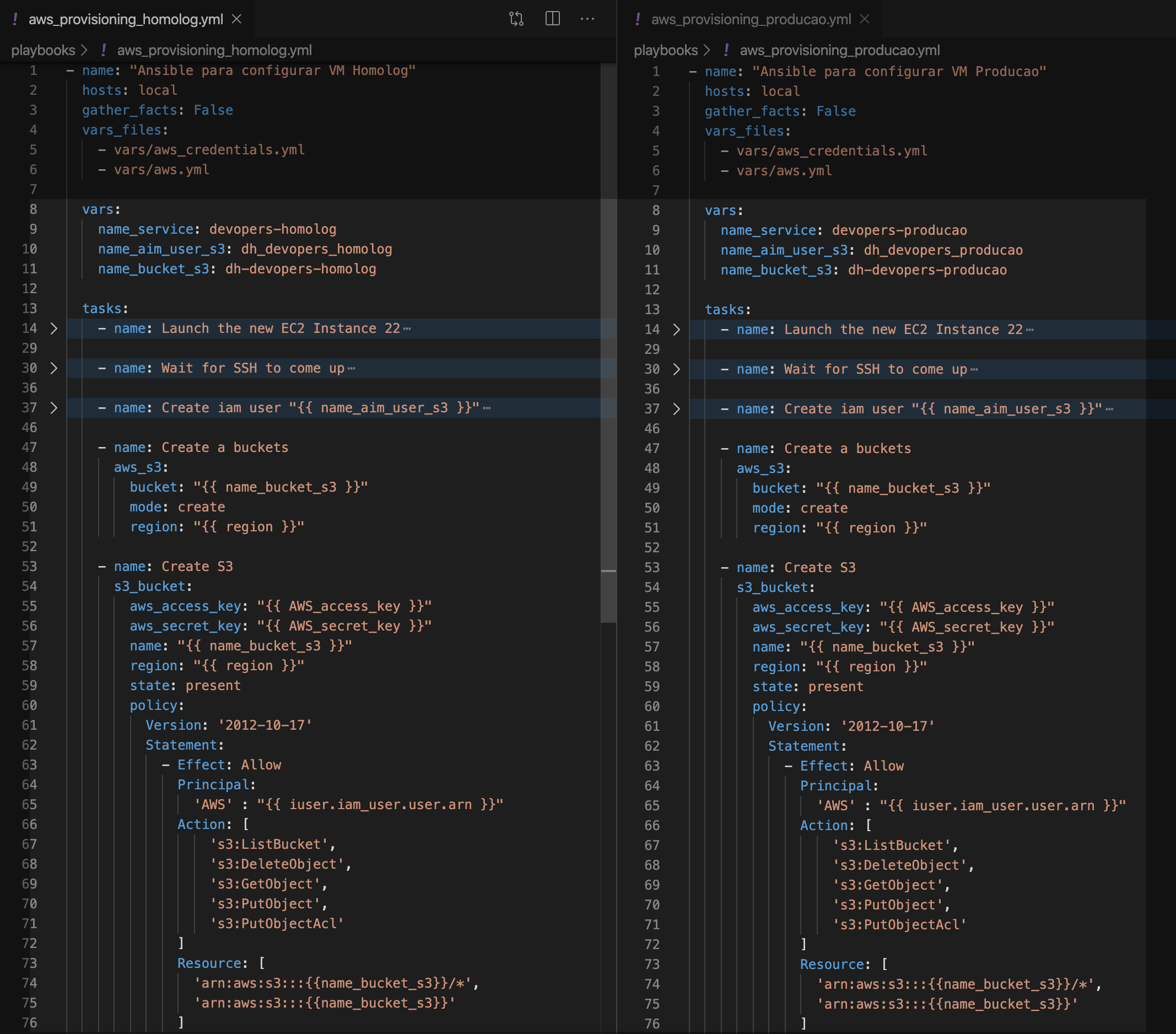
[Link útil - módulo ansible s3 bucket.](https://docs.ansible.com/ansible/latest/modules/s3_bucket_module.html)

[Link útil - módulo ansible ec2.](https://docs.ansible.com/ansible/latest/modules/ec2_module.html)

1. aws\_provisioning\_produção.yml: Ele é identico ao item anterior, aws\_provisioning\_homolog.yml, exceto pelos nome das variáveis usadas dentro do playbook.

**Importante**: O nome do bucket S3 possui algumas [regras](https://docs.aws.amazon.com/AmazonS3/latest/dev/BucketRestrictions.html#bucketnamingrules) para ser criado com sucesso, seja pelo playbook como pelo console da Amazon. O básico seria criar com letras minúsculas, sem números ou acentos, entre 3 e 63 caracteres e um nome único - que ninguém tenha usado ainda.





* Falando um pouco sobre a configuração das máquinas criadas\*\*:

Dividimos a configuração em 4 arquivos e utilizamos eles de forma dinâmica com o conceito de hosts + o [módulo](https://docs.ansible.com/ansible/latest/modules/ec2_instance_info_module.html) que coleta informações das instâncias, ou seja, podemos incluir quantas maquinas forem necessárias e executar o playbook apenas uma vez, e eles fará a instalação e/ou configuração em todas as máquinas "tagueadas":

Caso altere o nome das EC2, será necessário alterar a linha 14 nos dois arquivos abaixo (config\_all-ec2.yml e install\_docker\_all-ec2.yml).

*"tag:Name": ["devopers-jenkins","devopers-homolog","devopers-producao"]*

* + config\_all-ec2.yml: Atualiza pacotes em todas as EC2. Além de instalar uma série de softwares e suas dependências, dentre eles awscli, java, python.
  + install\_docker\_all-ec2.yml: Instala o docker em todas as EC2. O item abaixo atribuiu o usuário informado no grupo do docker.

*docker\_\_users: ["ubuntu","jenkins"]*

* + install\_ansible\_ec2-jenkins.yml: Instala ansible e pacotes para facilitar configurações.
  + install\_jenkins\_ec2-jenkins.yml: Instala e configura o Jenkins na EC2 do Jenkins. Os itens abaixo podem/devem ser alterados:

*jenkins\_admin\_username: devopers*

*jenkins\_admin\_password: devopers*

É necessário alterar a linha 14 nos 4 arquivos acima, caso altere o nome das EC2.

*"tag:Name": ["devopers-jenkins"]*

[Link útil - Módulos Ansible de todas as Clouds.](https://docs.ansible.com/ansible/latest/modules/list_of_cloud_modules.html#)

**Provisionando EC2 + S3 + IAM + ECR na AWS**

Partindo do pressuposto que seu ambiente de trabalho já está preparado e com as devidas ferramentas instaladas e configuradas, podemos iniciar a criação do ambiente na AWS, caso contrário volte para a Preparação Inicial.

O objetivo é criar um ambiente de Homologação e Produção + um servidor que rodará o Jenkins.

Ao ser executado o item abaixo irá criar uma VPC e todos os itens obrigatórios relacionados a ela (por exemplo, subnet, security group, entre outros), logo após irá gerar uma key-pair e salvar. Vai criar também uma EC2 para o Jenkins e um ECR (Elastic Container Registry), em seguida uma EC2 para Produção, um user e um bucket S3. E por último criará uma EC2 para Homologação, um user e um bucket S3.

$ *ansible-playbook playbooks/aws\_provisioning.yml*

O script acima utiliza os seguintes scripts:

aws\_provisioning\_vpc.yml,

aws\_provisioning\_jenkins.yml,

aws\_provisioning\_producao.yml,

aws\_provisioning\_homolog.yml.

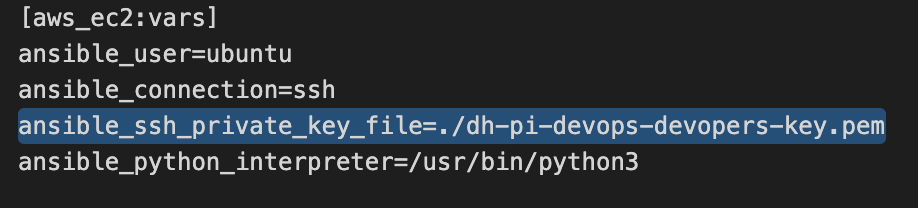
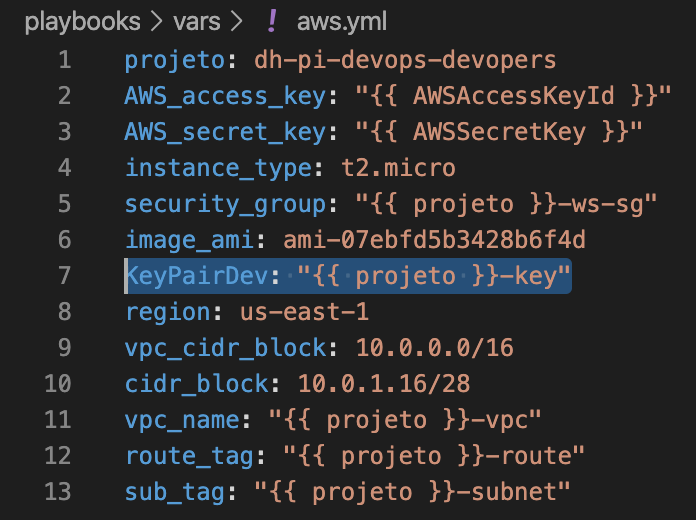
Para validar se foi criado acessar o console da AWS ou executar o seguinte comando:

$ *ansible-inventory --graph aws\_ec2*

**Configurando Máquinas EC2**

Após todo o ambiente criado na AWS, precisamos configurar as máquinas, todas as 3 EC2 criadas terão os pacotes atualizados e os seguintes itens instalados via ansible: docker, awscli, java, python, entre outros. Já a EC2 do Jenkins vamos instalar também o ansible e o jenkins.

**Antes de iniciar a configuração das máquinas o arquivo inventory/hosts precisa ser alterado, o item *ansible\_ssh\_private\_key\_file* precisa conter a chave que você criou no arquivo playbooks/vars/aws.yml.**



Atualizar pacotes e instalar awscli, java, python - Esse script foi preparado para atualizar as 3 EC2 ao ser executado…

$ *ansible-playbook playbooks/config\_all-ec2.yml*

Instalar docker - Esse script foi preparado para atualizar as 3 EC2 ao ser executado…

$ *ansible-playbook playbooks/install\_docker\_all-ec2.yml*

Instalar ansible - Esse script foi preparado para atualizar apenas a EC2 do Jenkins ao ser executado…

$ *ansible-playbook playbooks/install\_ansible\_ec2-jenkins.yml*

Instalar jenkins - Esse script foi preparado para atualizar apenas a EC2 do Jenkins ao ser executado…

$ *ansible-playbook playbooks/install\_jenkins\_ec2-jenkins.yml*

Observação: todos os playbooks podem ser executados com o parâmetro "-vvv" para ter um nível maior de detalhes.

Ex: $ *ansible-playbook playbooks/aws\_provisioning.yml -vvv*

**Configurando Jenkins**

Instalar plugins Jenkins que vamos utilizar em nosso pipeline. Acessar a url de acordo com o nome ou ip público gerado na AWS, para isso será necessário entrar no console e copiar um dos itens conforme imagem abaixo.



Após logar no Jenkins com usuário e senha definidos no playbook install\_jenkins\_ec2-jenkins.yml, ir em Gerenciar Jenkins → Gerenciar de Plugins → Disponíveis → procurar e selecionar os seguintes itens:

1.pipeline

2.docker pipeline

3.ssh

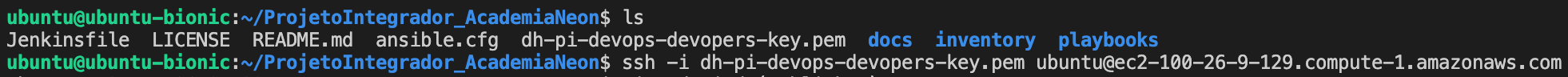
4.github

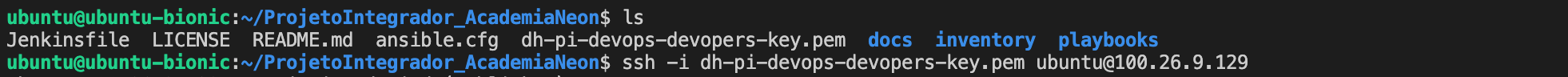
5.github api

6.amazon ecr

Acessar a EC2 via ssh conforme exemplo abaixo (é possível pegar esse comando no console da AWS também) para validar se o usuário jenkins está incluso no grupo docker do Linux (id jenkins):

*ssh -i key.pem usuario@host*





Caso o usuário não esteja executar o comando abaixo para incluir e depois reiniciar o serviço.

* Verificar usuário

$ *id jenkins*

* Incluir o usuário no grupo

$ *sudo addgroup jenkins docker*

* Parar o serviço do jenkins

$ *sudo service jenkins stop*

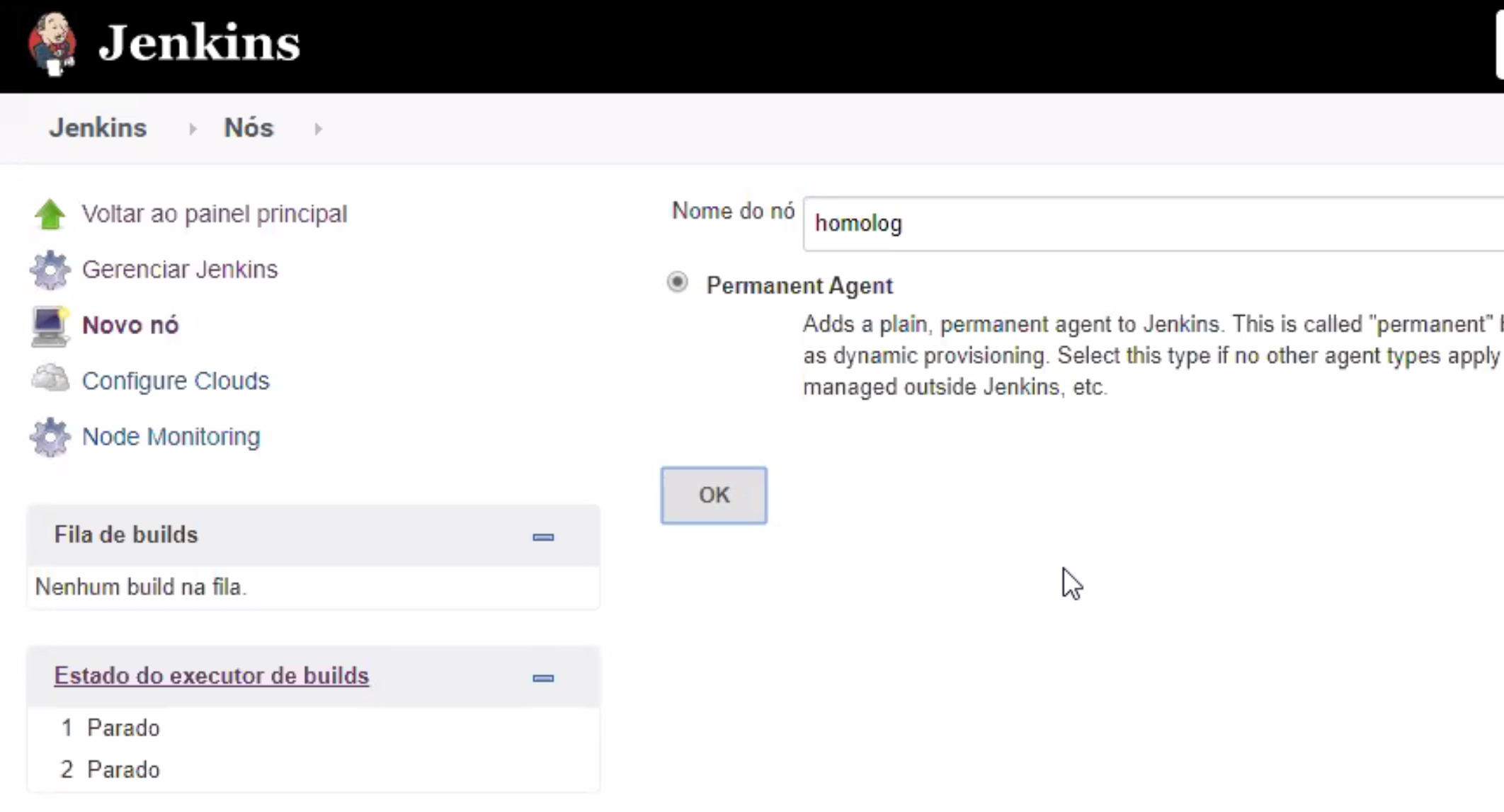
* Subir o serviço do jenkins

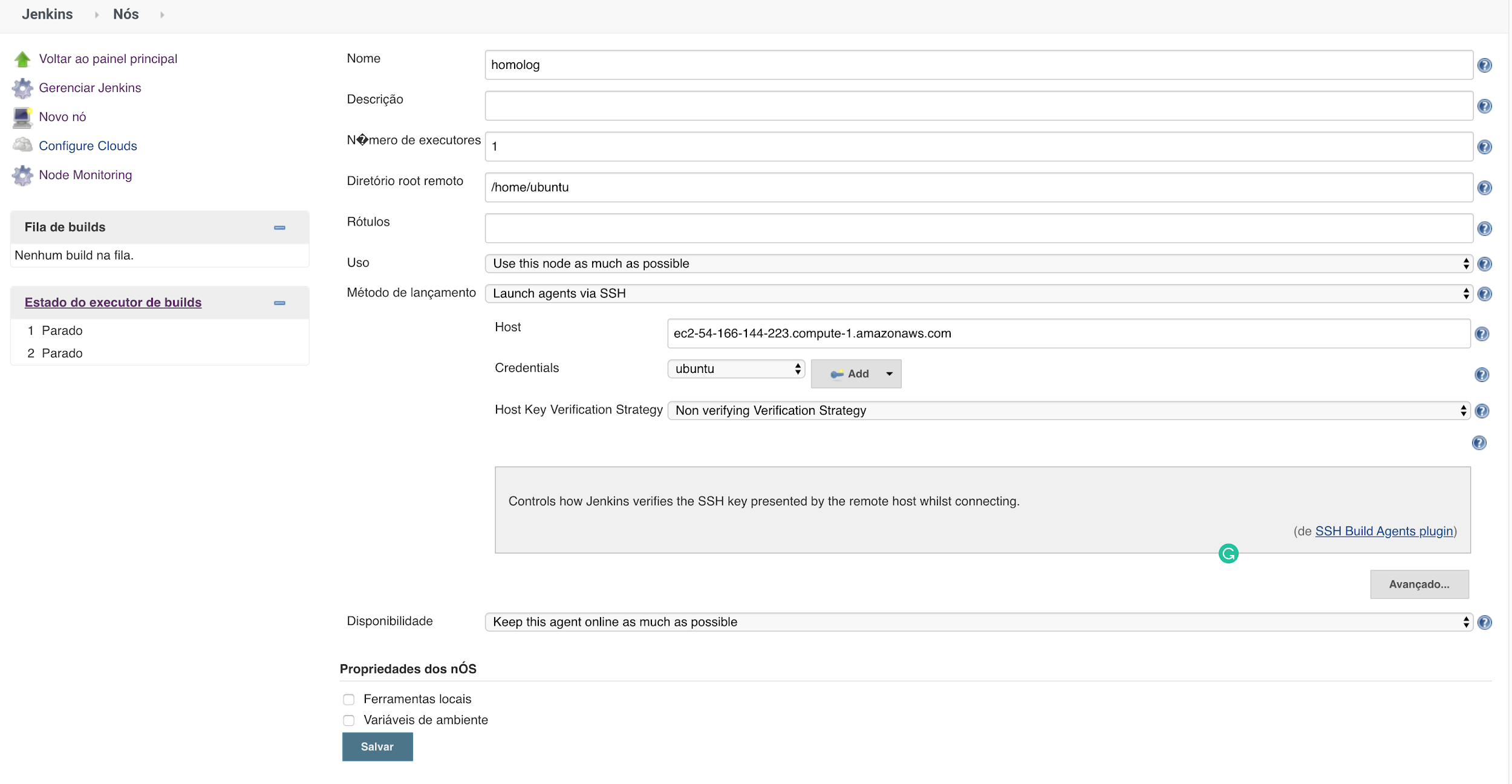
$ *sudo service jenkins start*

* Validar o serviço do jenkins

$ *sudo service jenkins status*

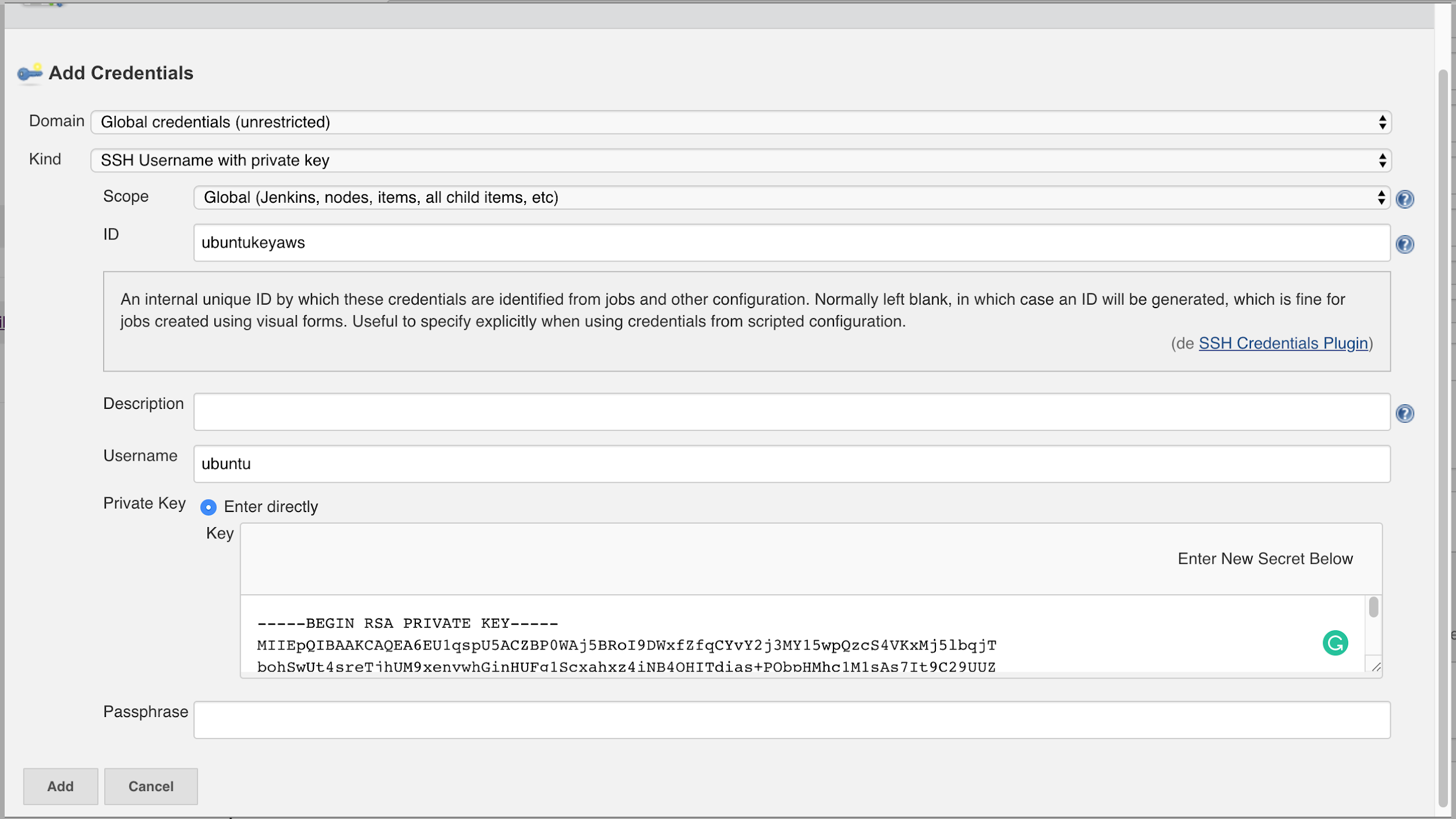
É necessário configurar os nodes de Homologação e Produção para a execução do pipeline, para tal ir em ***Gerenciar Jenkins → Gerenciar nós → novo nó***

******

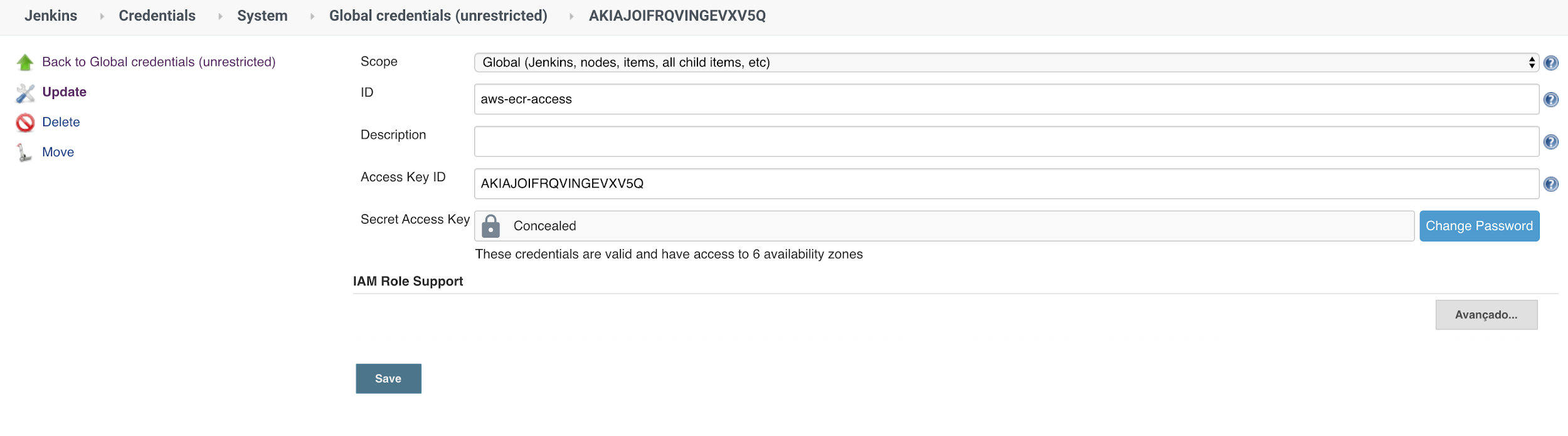


Também será preciso cadastrar as credenciais, tanto as Credenciais de acesso via SSH como credenciais de acesso da AWS.

***1.*** Adicionar Credenciais para acessar as instâncias: ***Credentials → Add Credentials***. Escolher Global e SSH username + private key



***2.*** Adicionar Credenciais para acessar AWS ECR: ***Credentials → Add Credentials***. Escolher AWS Credentials + Global + ID e Private Key AWS



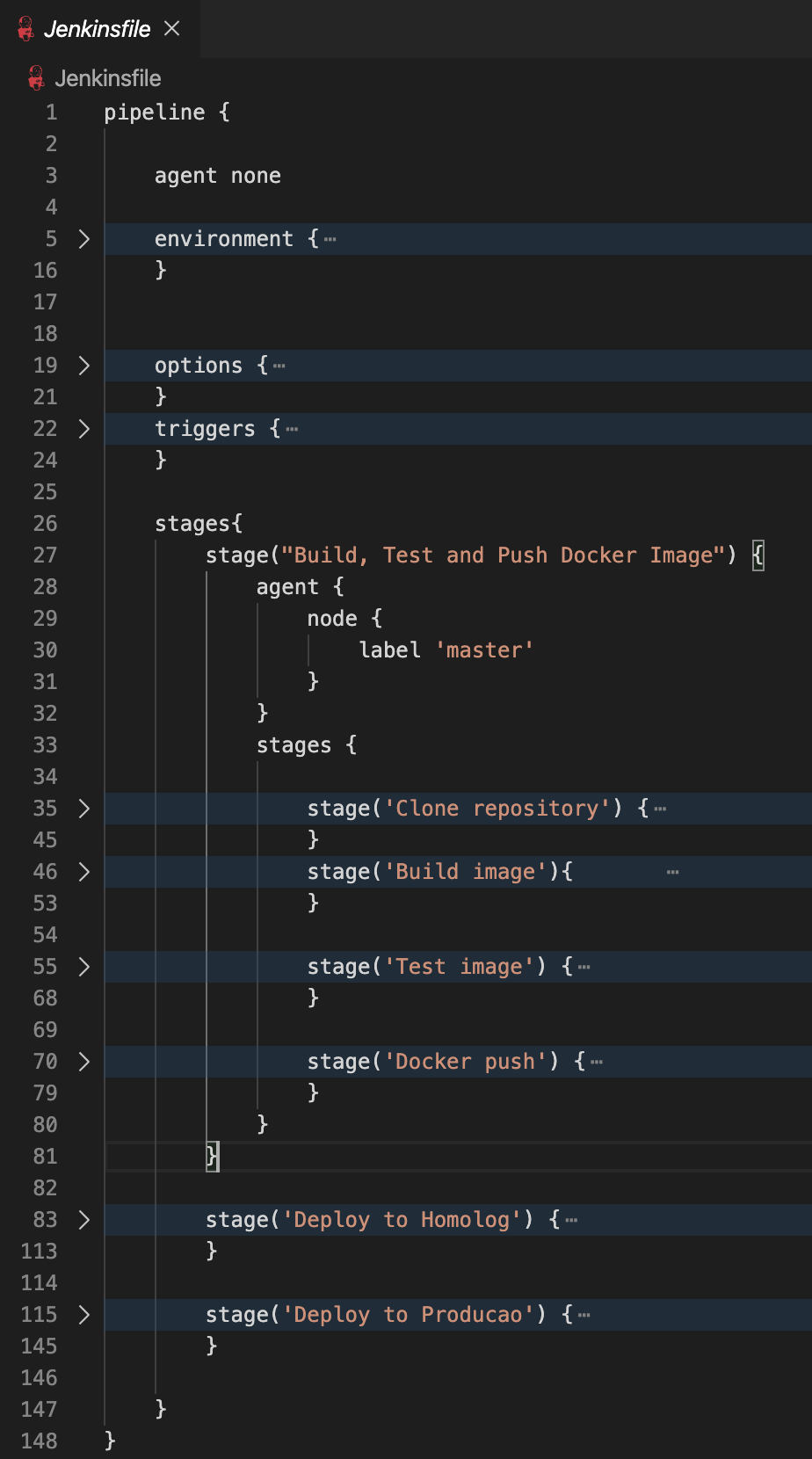
Links úteis sobre o uso de Credenciais AWS com Jenkins:

[Pushing to ECR Using Jenkins Pipeline Plugin](https://blog.mikesir87.io/2016/04/pushing-to-ecr-using-jenkins-pipeline-plugin/)

[AWS: create an Elastic Container Registry and Jenkins deploy job](https://rtfm.co.ua/en/aws-create-an-elastic-container-registry-and-jenkins-deploy-job/)

**Customizando Jenkinsfile**

Antes de começarmos a customizar uma breve explicação de cada estágio do Jenkinsfile.



Dentro do item ***stage("Build, Test and Push Docker Image")*** que será executado no servidor do Jenkins, temos 4 passos:

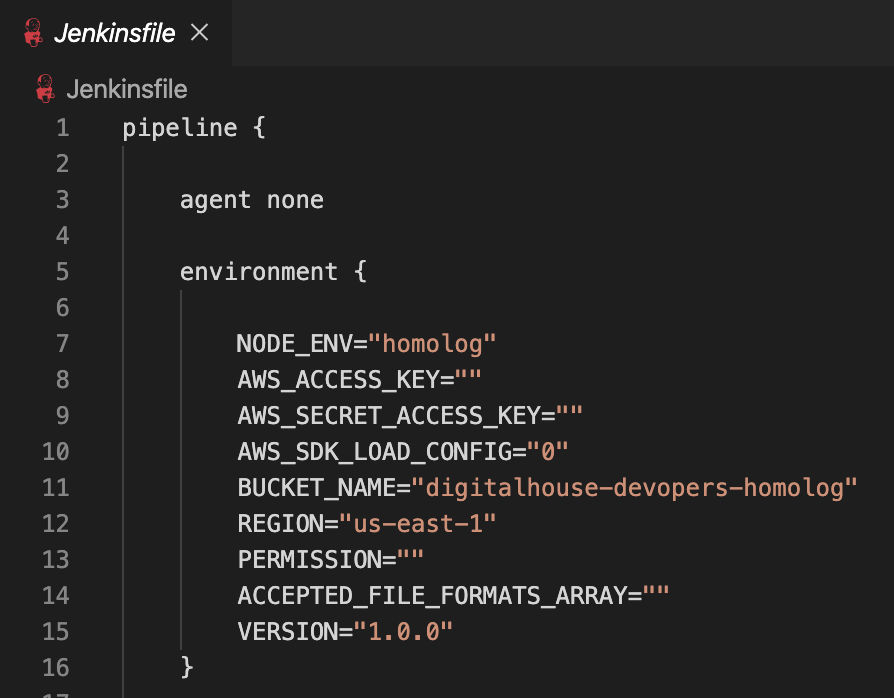
* ***stage('Clone repository')***: Clonar o repositório que vamos definir no pipeline.
* ***stage('Build image')***: Fazer o build da imagem.
* ***stage('Test image')***: Subir um container da imagem na porta informada e fazer o healthcheck.
* ***stage('Docker push')***: Subir a imagem no AWS ECR e versionar.

Depois temos os itens ***stage('Deploy to Homolog')*** e ***stage('Deploy to Producao')***, ambos vão fazer o deploy da aplicação e da mesma forma, a diferença será o servidor é claro, que foi definido nas configurações de nodes/nos do Jenkins. O primeiro passo é buscar a imagem no AWS ECR, se o container estiver no ar ele pára, depois exclui, e somente depois roda e faz o healthcheck. Importante lembrar que para seguir de um estágio para o outro todos tem que ser executados com sucesso.

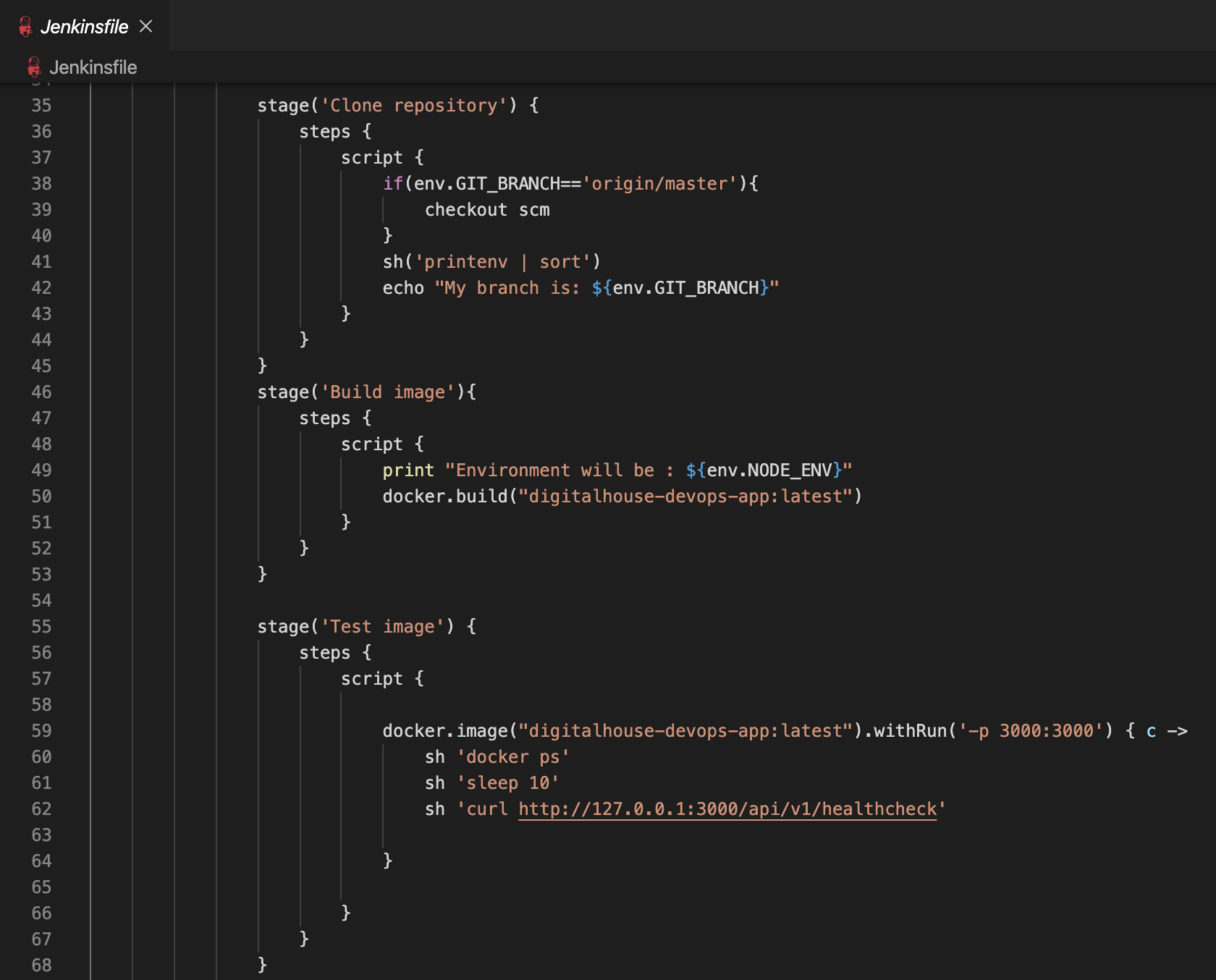
Utilizamos o catchError {...} pois na primeira execução o container não existe e dava erro na hora de parar e apagar o container e o pipeline não era concluído, é uma melhoria que ainda precisa ser feita, utilizar comandos docker e linux para validar se o container existe e se está no ar antes de executar os comandos de stop e rm. Na primeira execução, apesar do pipeline ficar verde, o job ficará vermelho, pois o erro foi ignorado, já na segundo teremos 100% de sucesso

Será necessário customizar o arquivo Jenkinsfile, lembrando que ele deverá ser salvo no repositório do desenvolvedor, que no nosso caso é separado do repositório do DevOps.

* As variáveis de ambiente que estão na linha 7 até a 15, devem ser alteradas de acordo com o projeto, elas são utilizadas no teste local.

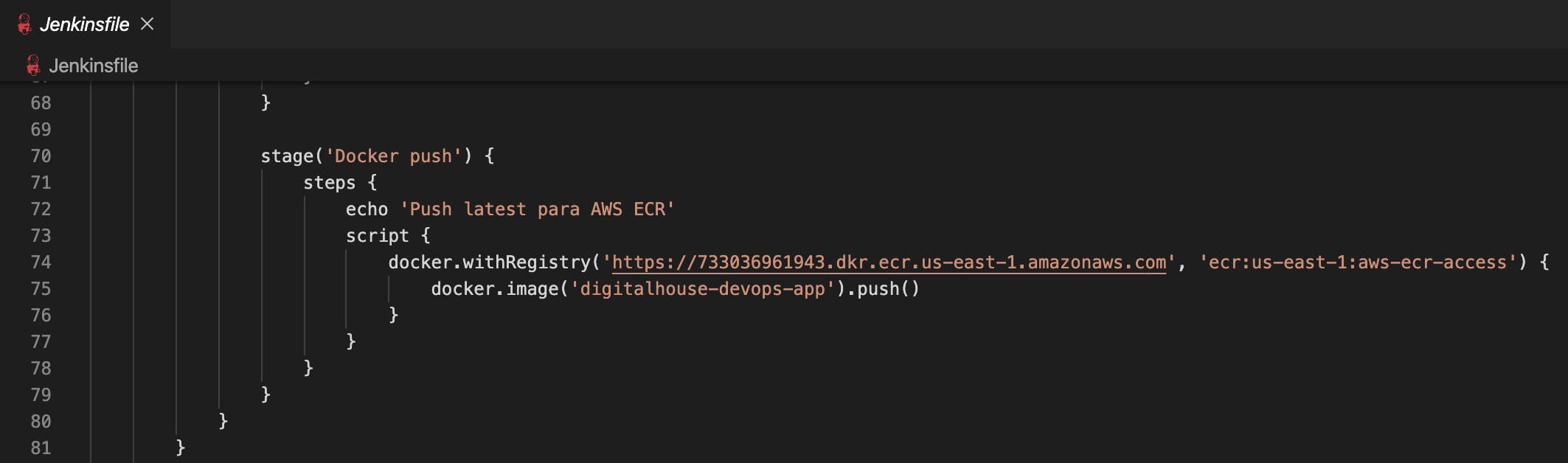


* Aqui vamos configurar o teste local. As linhas que devem ser alteradas são: 38 (colocar o nome da branch do git do desenvolvedor, conforme imagem abaixo foi utilizado a master), 50 (nome:versao que a imagem vai receber, conforme imagem abaixo foi utilizado digitalhouse-devops-app:latest) e 59 (nome:versao - manter a utilizada na linha 50 e a porta que a aplicação vai rodar, foi usado 3000:3000).



* Neste item vamos configurar o push da imagem para o AWS ECR. As linhas que devem ser alteradas são: 74 (é a url do repositório que criamos na AWS, deve ser pego no console da AWS - e ao colocar no arquivo o que foi copiado do console manter o https:// e retirar a barra e o nome do repositório, depois manter o item ecr:regiao-que-criou-o-repositorio:nome que foi dado a credencial do jenkins) e 75 (repetir apenas o nome da imagem utilizada na linha 50, sem a versão).





* Neste item vamos configurar o Deploy para Homologação. As linhas que devem ser alteradas são: 86 (nome do node/nó de homologação criado no Jenkins), 92 (nome da branch do desenvolvedor que irá usar), 94 e 95 (alteração idêntica ao item de cima, porém aqui vamos fazer o pull da imagem para utilizarmos em nosso deploy), 101 e 102 (nome que deu para o container) e 104 (para efetuar o run algumas variáveis precisar ser passadas, além da imagem que está na AWS, veja abaixo).

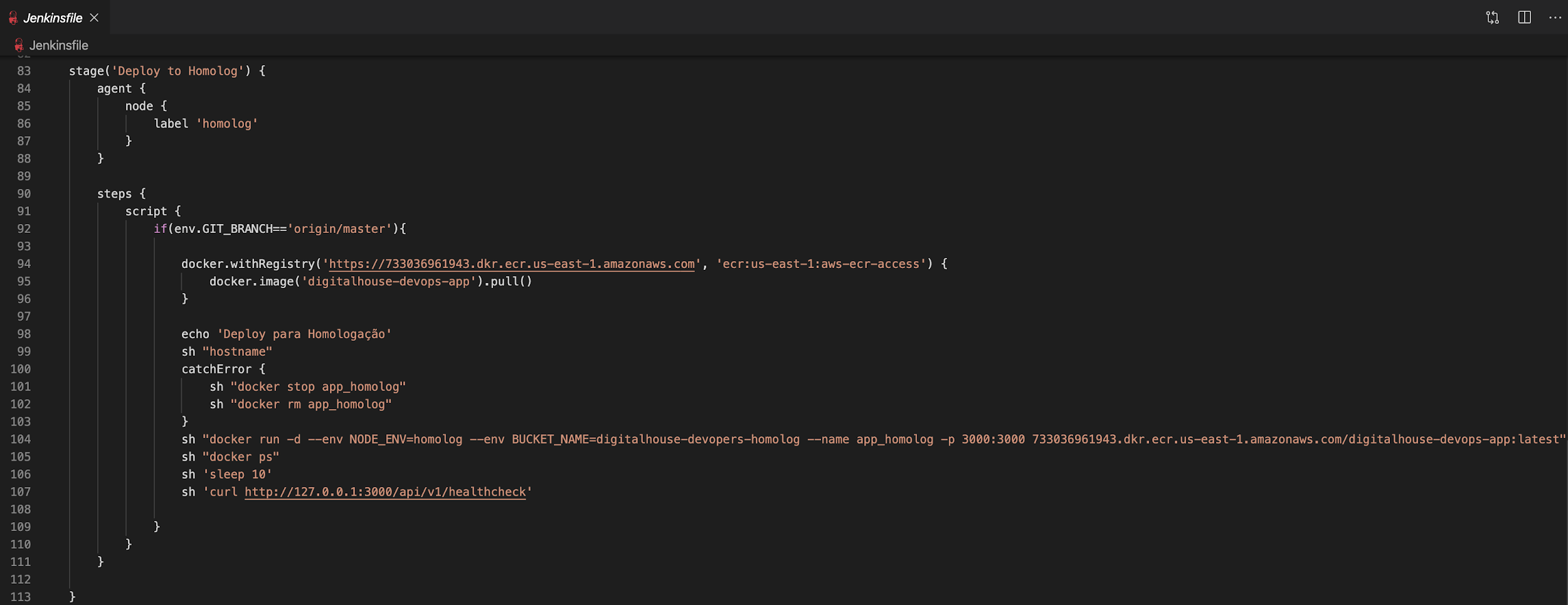
*--env NODE\_ENV=homolog* → Ambiente que está sendo feito o deploy e a aplicação vai mostrar no output do pipeline healthcheck.

*--env BUCKET\_NAME=digitalhouse-devopers-homolog* → Nome do bucket do ambiente.

*--name app\_homolog* → Nome do container docker.

*-p 3000:3000* → Porta que vai rodar.

*733036961943.dkr.ecr.us-east-1.amazonaws.com/digitalhouse-devops-app:latest* → repositório aws ecr (só copiar a url) + versão que vamos utilizar.



* Neste item vamos configurar o Deploy para Produção. As linhas que devem ser alteradas são: 118 (nome do node/nó de produção criado no Jenkins), 124 (nome da branch do desenvolvedor que irá usar), 126 e 127 (alteração identica ao item do ecr, porém aqui vamos fazer o pull da imagem para utilizarmos em nosso deploy), 133 e 134 ((nome que deu para o container) e 136 (para efetuar o run algumas variáveis precisar ser passadas, além da imagem que está na AWS, veja abaixo).

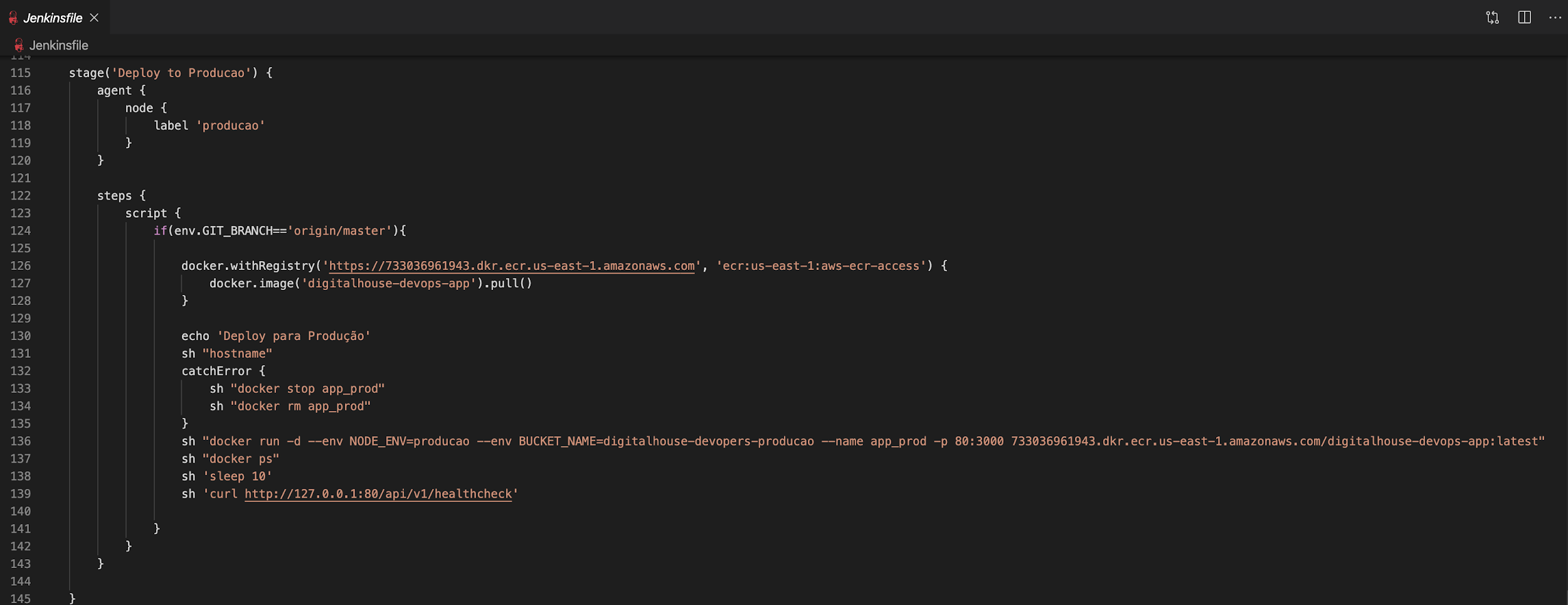
*--env NODE\_ENV=producao* → Ambiente que está sendo feito o deploy e a aplicação vai mostrar no output do pipeline healthcheck.

*--env BUCKET\_NAME=digitalhouse-devopers-producao* → Nome do bucket do ambiente.

*--name app\_prod* → Nome do container docker.

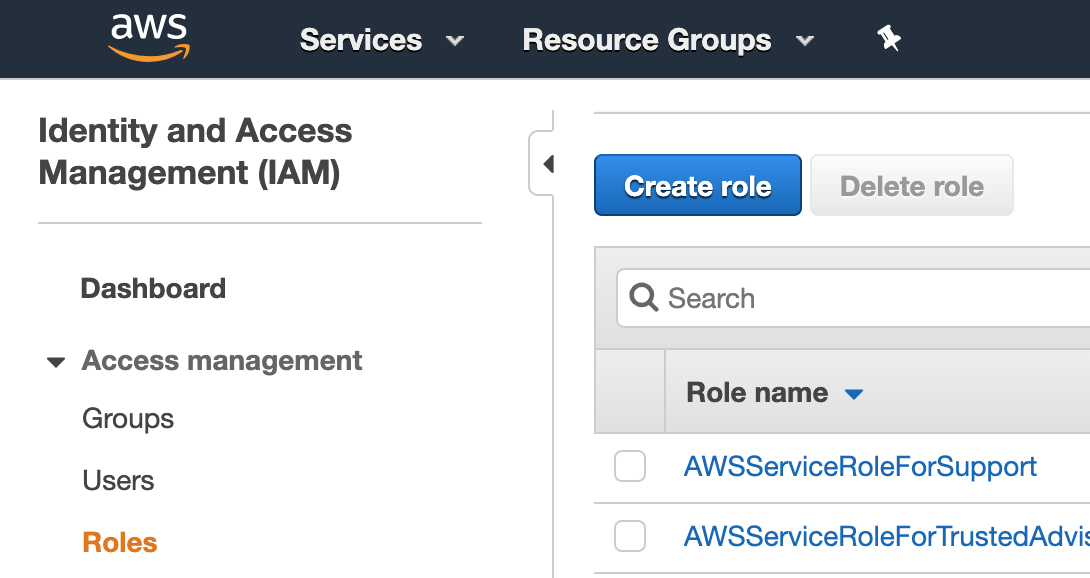
*-p 80:3000* → Porta que vai rodar.

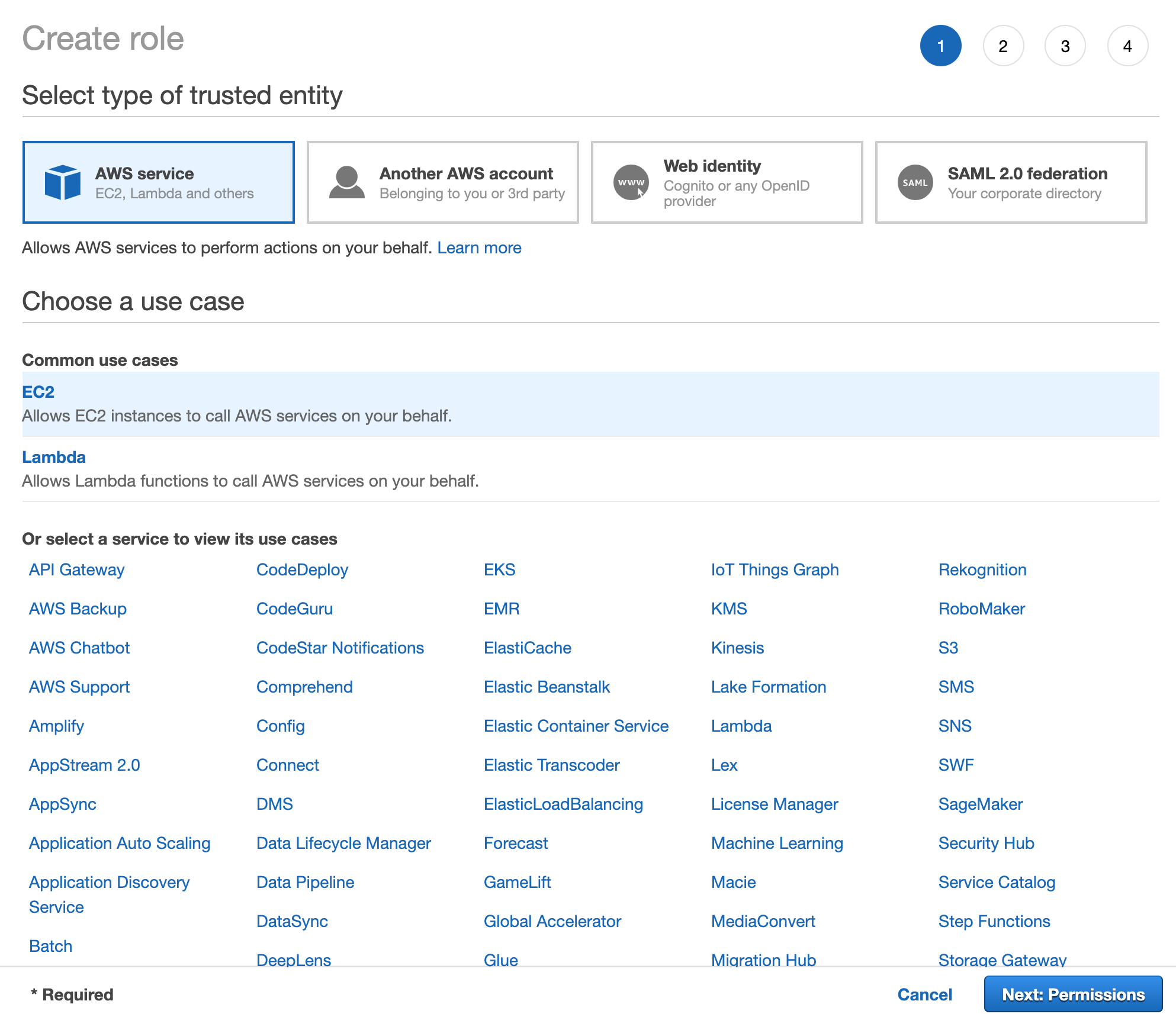
*733036961943.dkr.ecr.us-east-1.amazonaws.com/digitalhouse-devops-app:latest* → repositorio aws ecr (só copiar a url) + versão que vamos utilizar.



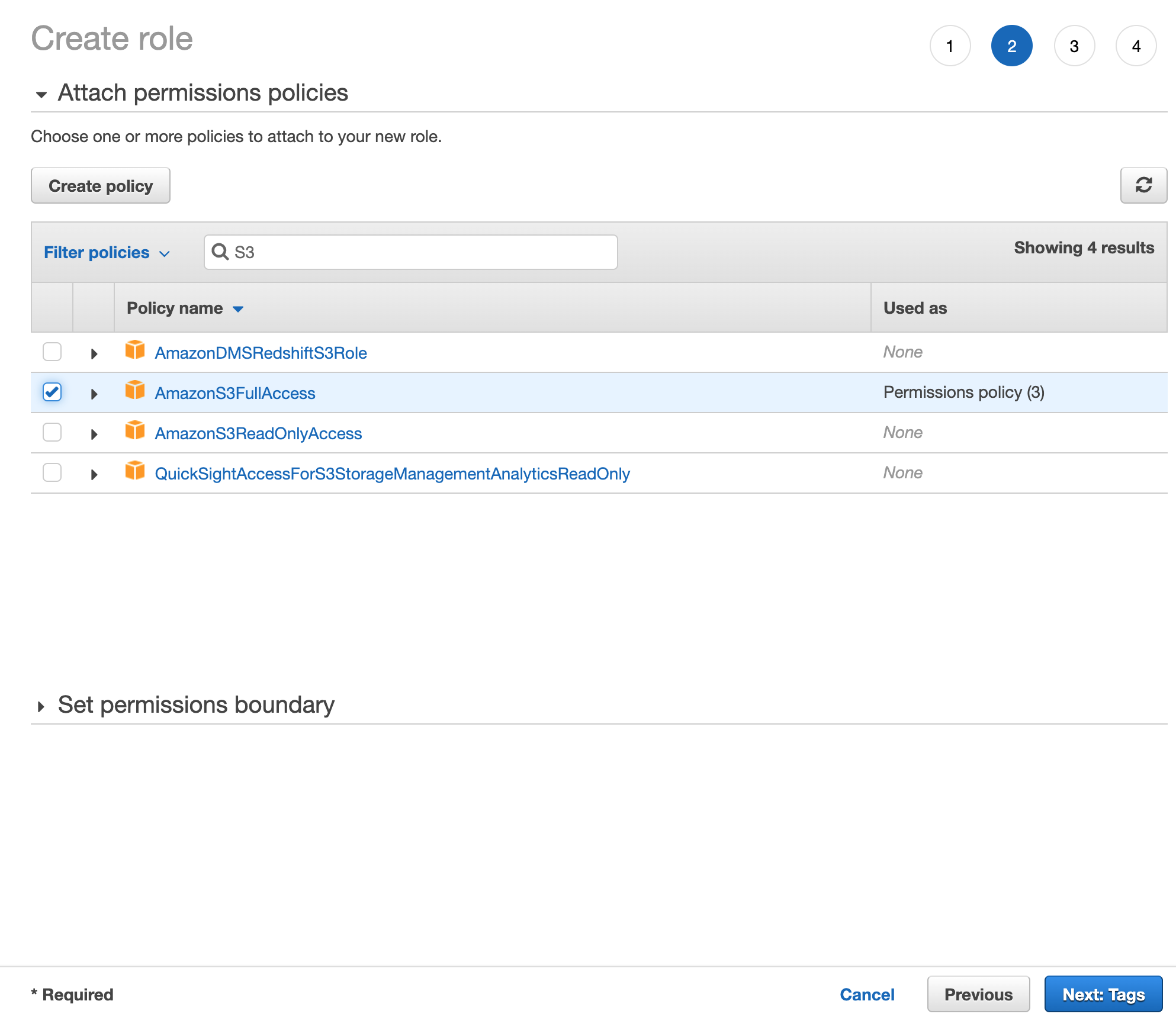
**Importante**: Outros itens podem ser alterados de acordo com seu entendimento, por exemplo stage e echo, que são textos que vão ser mostrados no output e fazem referência ao que está sendo feito. No deploy de homologação e produção também poderia ter usado credenciais do ECR, porém optamos por criar uma role na AWS para fazer esse permissionamento.

Para a criação da role seguir os seguintes passos:

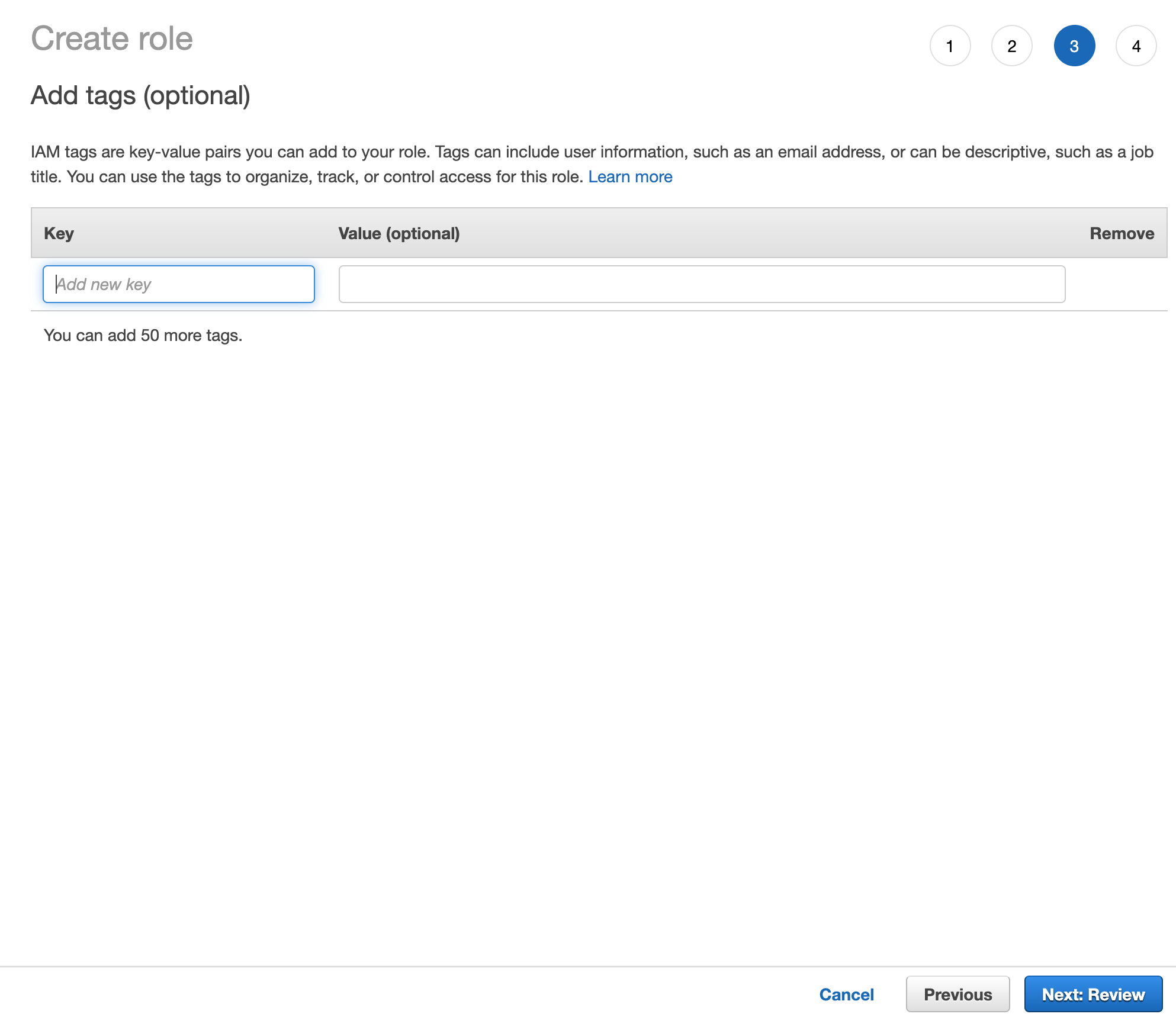
* No console AWS ir em Services → IAM → Roles → Create Roles 
* Na primeira tela, escolher quem vai assumir a role, a permissão, em nosso caso a EC2

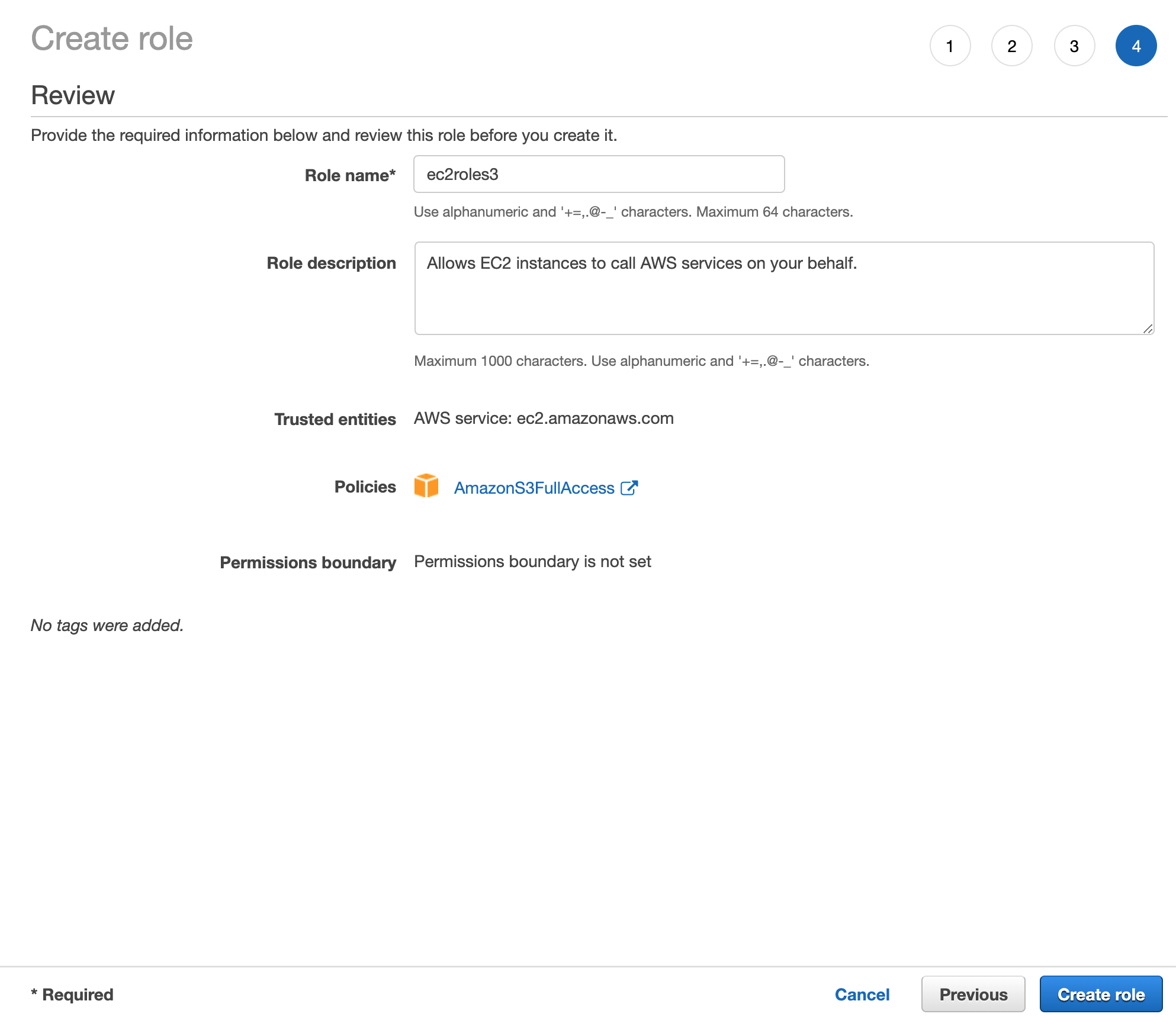


* Na segunda tela, escolher o que vai pode ser feito, em nosso caso a EC2 vai ter acesso total ao bucket S3



* Na terceira tela, pode deixar em branco, usado para identificação



* E por último dar um nome para a role e criar

Já se optar pela criação das Credenciais AWS no Jenkins além de trocar as linhas 104 e 136 (sh "docker run.....) no Jenkinsfile pelo item abaixo, ainda é preciso configurar credenciais para cada ambiente, igual foi feito para o AWS ECR.

*withCredentials([[$class:'AmazonWebServicesCredentialsBinding'*

*, credentialsId: 'nomeCredencial']]) {*

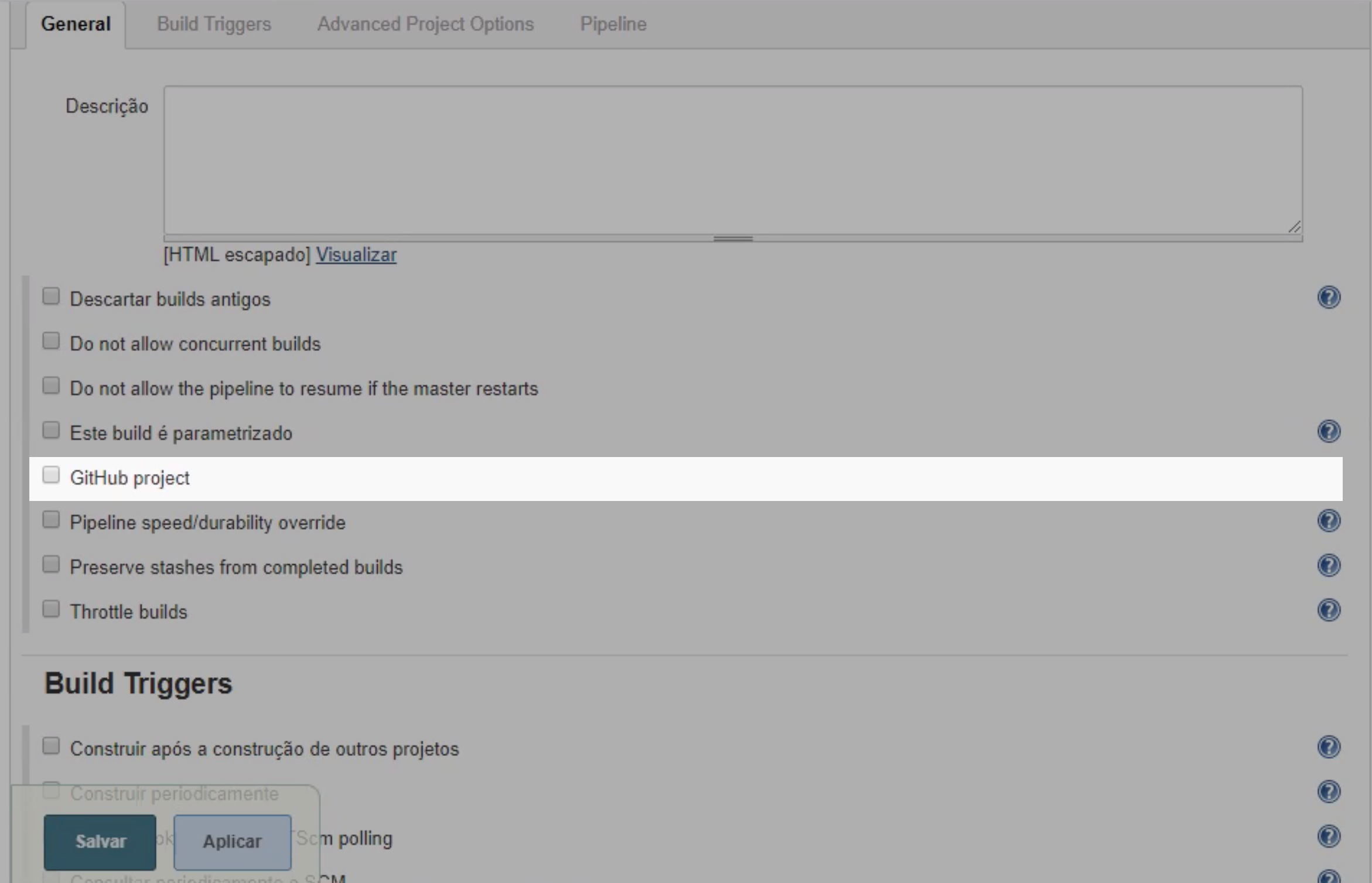
*sh "docker run -d --name nome\_app -p 30:3000 -e NODE\_ENV=ambiente -e AWS\_ACCESS\_KEY=$AWS\_ACCESS\_KEY\_ID -e AWS\_SECRET\_ACCESS\_KEY=$AWS\_SECRET\_ACCESS\_KEY -e BUCKET\_NAME=nome\_bucket 733036961943.dkr.ecr.us-east-1.amazonaws.com/digitalhouse-devops-app:latest"*

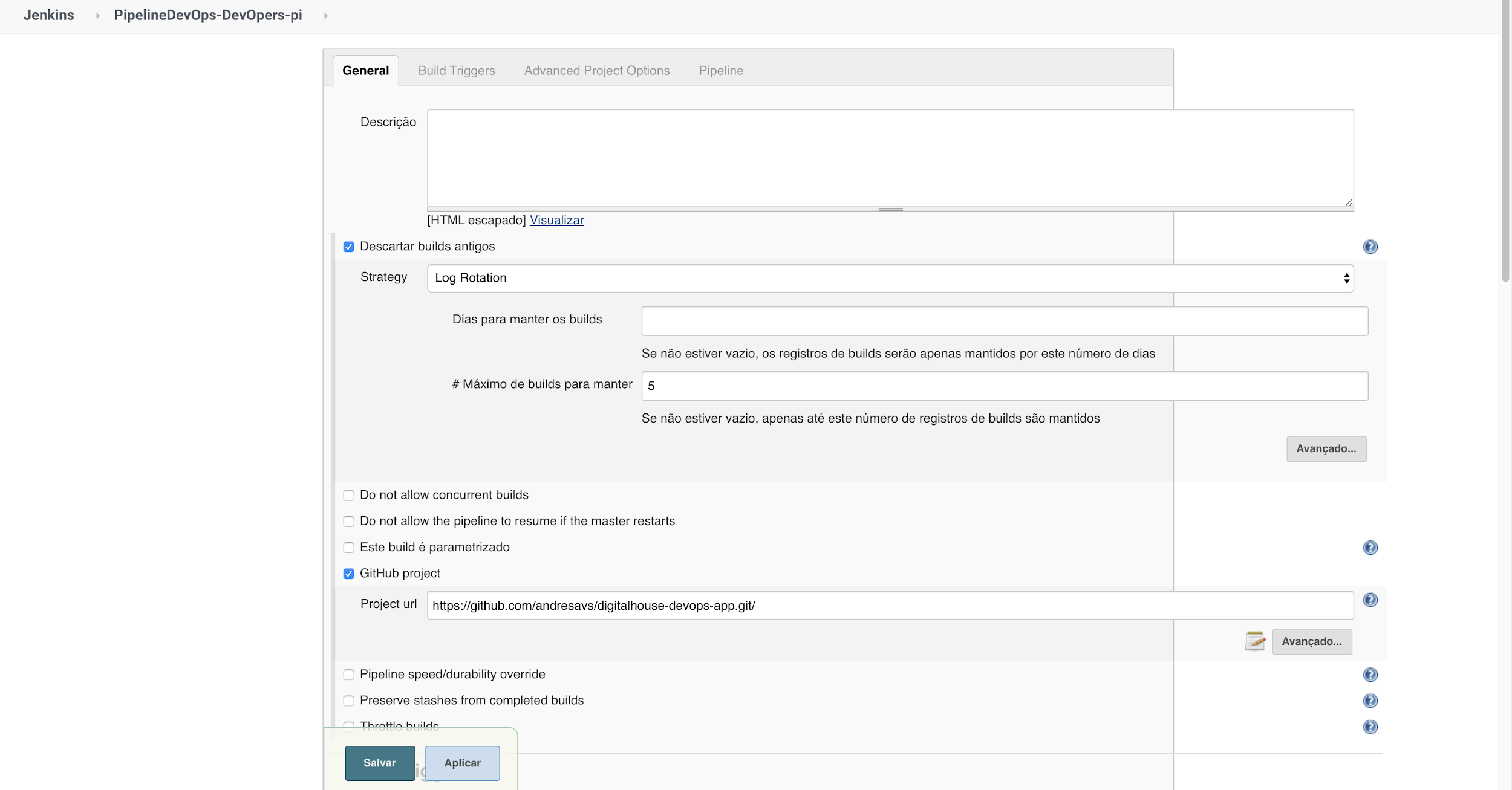
*}*

**Pipeline Jenkins**

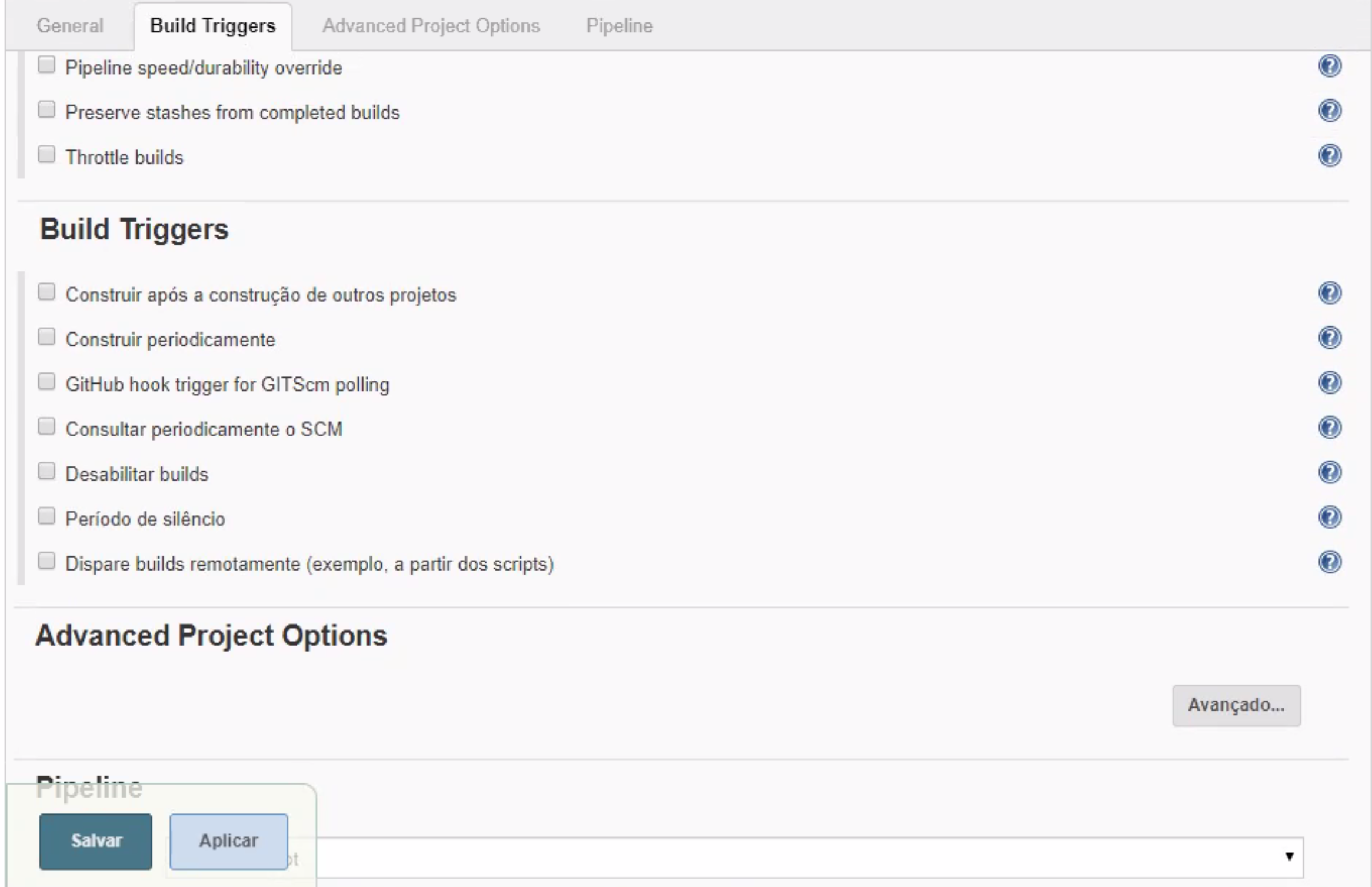
Após realizar todas as configurações é possível criar um Job Pipeline que irá no repositório git e fará todo o deploy através do Jenkinsfile.

Na aba General marcar o GitHub project e informar a url do git do Desenvolvedor, onde está o Jenkinsfile.

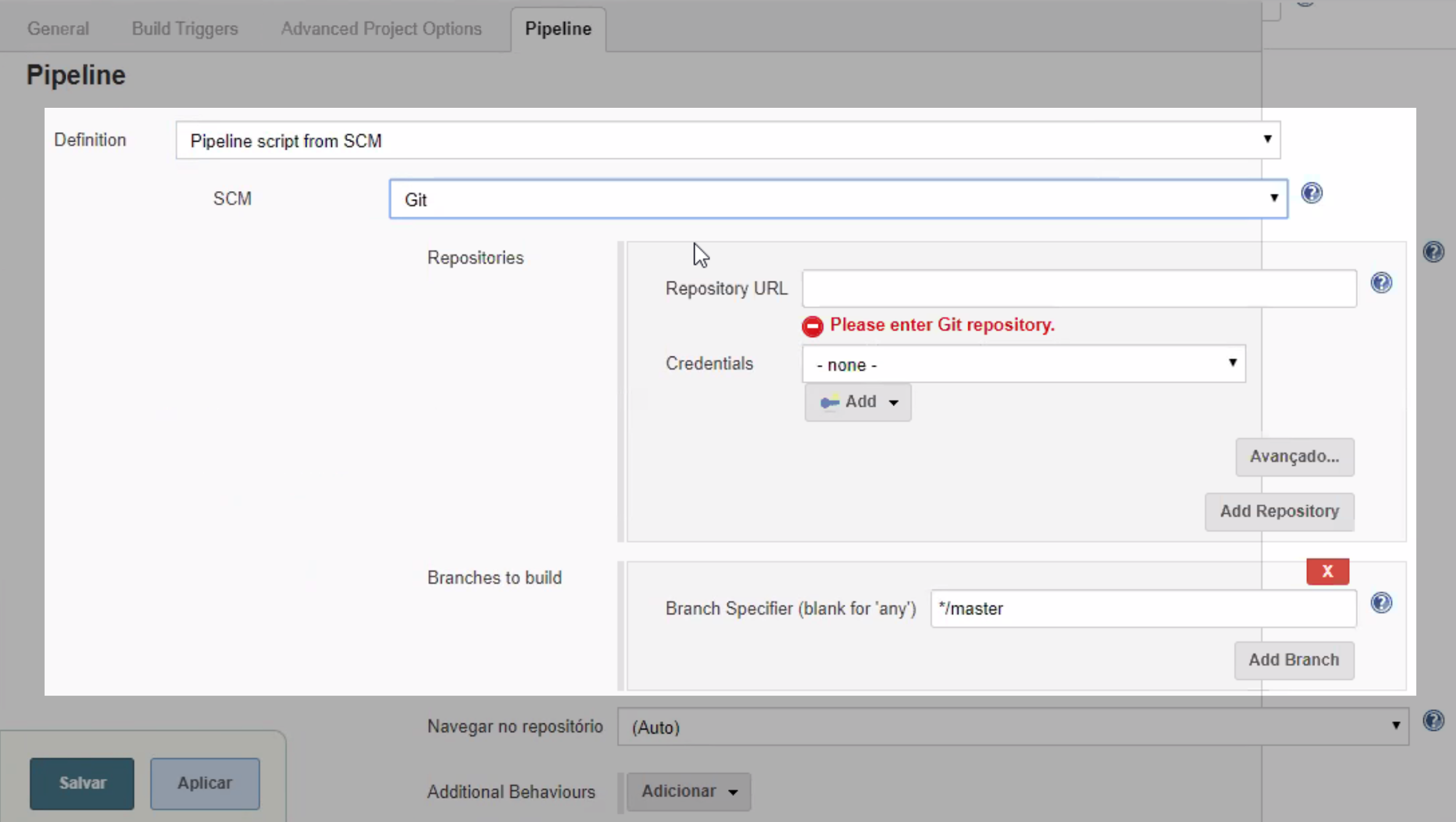


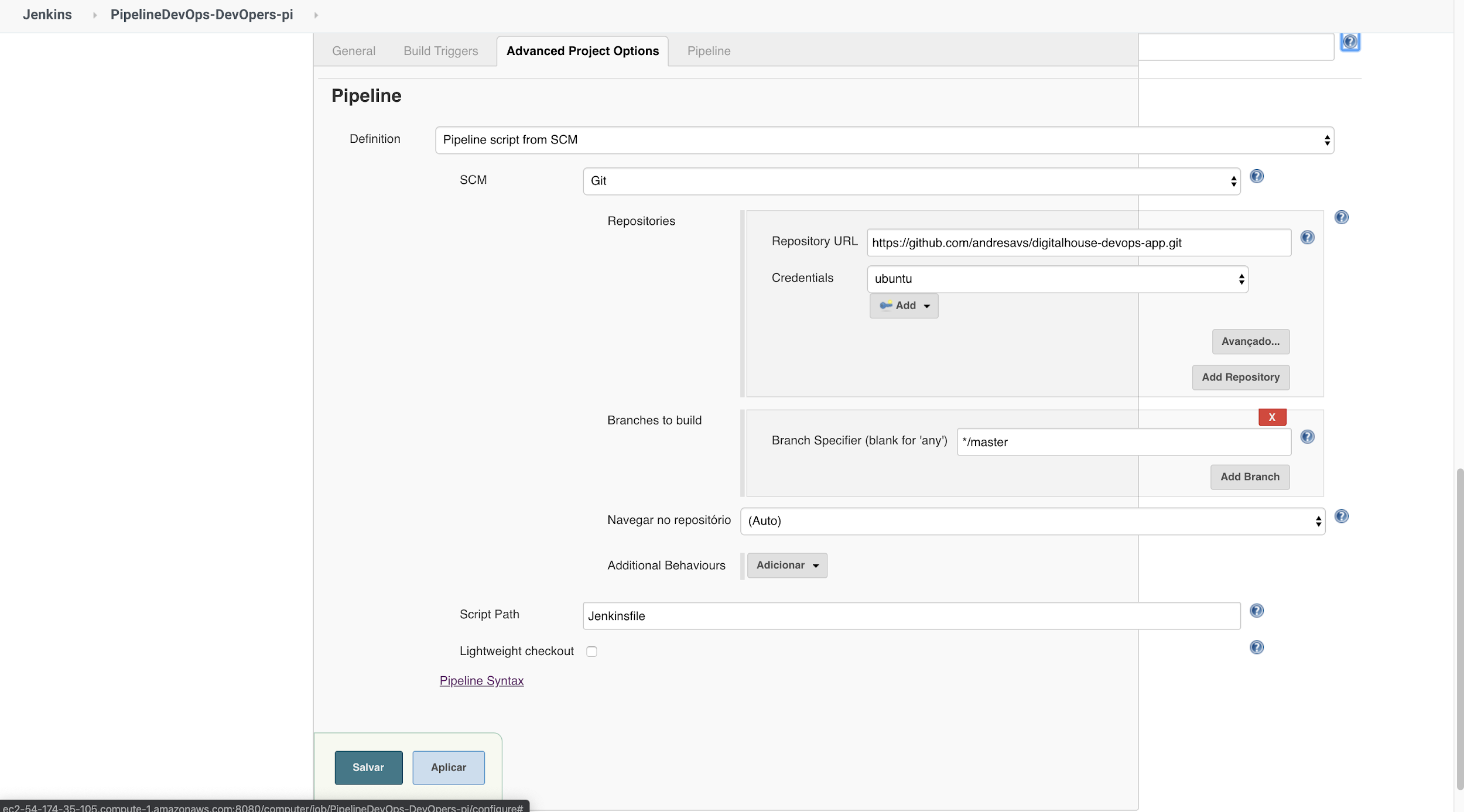


Nas abas Build Triggers e Advanced Project Options nada precisa ser informado.



Na aba Pipeline em Definition escolher **Pipeline script from SCM** (Source Code Management). Em SCM escolher Git. Repository URL informar a url do git do desenvolvedor. Credentials informar uma credencial, utilizamos a mesma da configuração dos nodes/nos. Branch Specifier é a branch do repósitorio.





[Link útil - Pipeline como código no Jenkins.](https://imasters.com.br/back-end/pipeline-como-codigo-no-jenkins)

**APP NodeJS utilizado**

Foi realizado um fork do [repositório do Desenvolvedor](https://github.com/bgsouza/digitalhouse-devops-app.git) para [repositório Dev do Projeto](https://github.com/andresavs/digitalhouse-devops-app.git) pois teríamos que customizar o arquivo Jenkinsfile.

**Referências**

\* Professores

\* Bruno G. Souza - <https://github.com/bgsouza/digitalhouse-devops-app>

\* Krishna Pennacchioni - <https://github.com/agentelinux/devops-pi/tree/grupo1>

\* Material do curso - Playground Digital House

\* Documentação Oficial

\* <https://www.ansible.com/>

\* <https://galaxy.ansible.com/>

\* <https://github.com/>

\* <https://www.docker.com/>

\* <https://www.jenkins.io/>

\* <https://aws.amazon.com/pt/>

\* <https://www.markdownguide.org/>