

Tutorial Projeto Integrador Academia Neon

Projeto Integrador (Curso DevOps) - Academia Neon

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

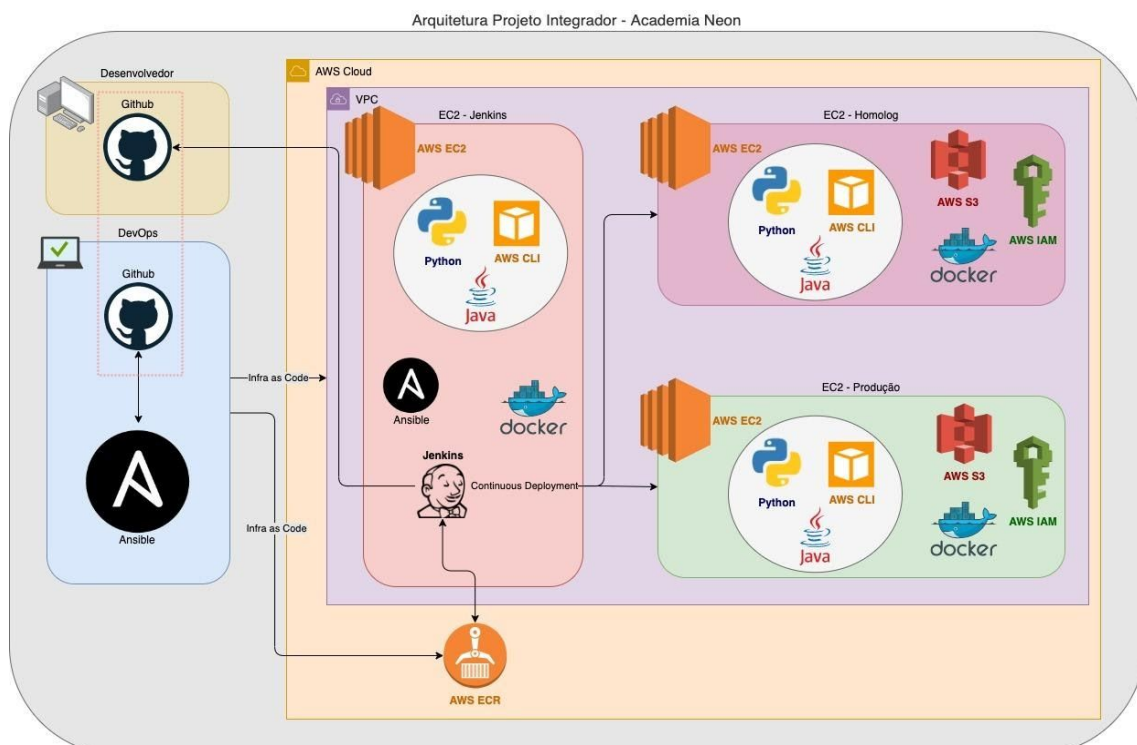
Sobre o Projeto Integrador

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](#). 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](#).

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:

```
$ vagrant ssh
Welcome to Ubuntu 18.04.4 LTS (GNU/Linux 4.15.0-99-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:   https://landscape.canonical.com
 * Support:      https://ubuntu.com/advantage

System information as of Thu May 14 18:56:28 UTC 2020

System load:  0.08               Processes:            99
Usage of /:   22.6% of 9.63GB    Users logged in:     0
Memory usage: 9%                IP address for enp0s3: 10.0.2.15
Swap usage:   0%                IP address for enp0s8: 192.168.33.10

 * Canonical Livepatch is available for installation.
   - Reduce system reboots and improve kernel security. Activate at:
     https://ubuntu.com/livepatch

7 packages can be updated.
0 updates are security updates.

Last login: Mon May 11 17:09:58 2020 from 10.0.2.2
vagrant@ubuntu-bionic:~$ sudo su - ubuntu
ubuntu@ubuntu-bionic:~$ cd ProjetoIntegrador_AcademiaNeon/
ubuntu@ubuntu-bionic:~/ProjetoIntegrador_AcademiaNeon$
```

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.

```
ProjetoIntegrador_AcademiaNeon $ git status
On branch andresa
Your branch is up to date with 'origin/andresa'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   README.md
        modified:   docs/DevOps_ArquiteturaPI.jpg

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .DS_Store
        .gitignore
        docs/.DS_Store
        docs/DevOps_ArquiteturaPI-v01.jpg
        playbooks/vars/aws_credentials.yml
```

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

```
.gitignore
1  # Arquivos ignorados
2  .DS_Store
3  docs/.DS_Store
4
5  # Credenciais AWS
6  playbooks/vars/aws_credentials.yml
7
8  # Logs
9
10 # Diretorios
11
```

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

```
ProjetoIntegrador_AcademiaNeon $ git status
On branch andresa
Your branch is up to date with 'origin/andresa'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   README.md
        modified:   docs/DevOps_ArquiteturaPI.jpg

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .gitignore
        docs/DevOps_ArquiteturaPI-v01.jpg

no changes added to commit (use "git add" and/or "git commit -a")
```

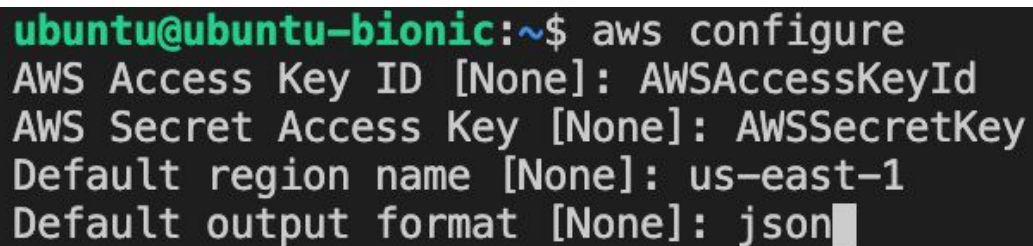
Configurar AWS CLI no desktop de trabalho

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
```

A terminal window with a black background and green text. The prompt is 'ubuntu@ubuntu-bionic:~\$'. The command 'aws configure' has been entered. The output shows four prompts: 'AWS Access Key ID [None]:', 'AWS Secret Access Key [None]:', 'Default region name [None]:', and 'Default output format [None]:'. The user has entered 'AWSAccessKeyId', 'AWSSecretKey', 'us-east-1', and 'json' respectively. A cursor is visible at the end of the last line.

```
ubuntu@ubuntu-bionic:~$ aws configure
AWS Access Key ID [None]: AWSAccessKeyId
AWS Secret Access Key [None]: AWSSecretKey
Default region name [None]: us-east-1
Default output format [None]: json
```

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](#).

- **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.](#)

- 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.](#)

- .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:**

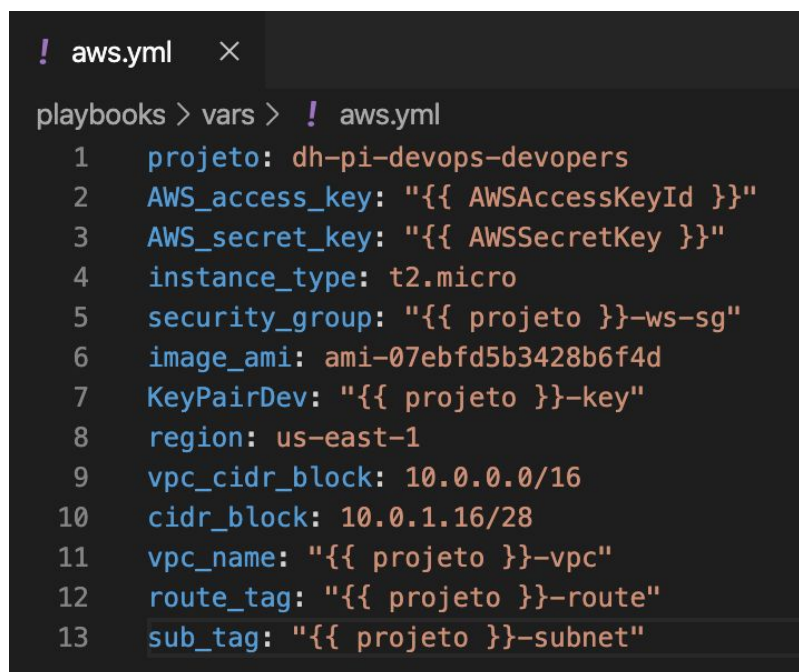
Nosso arquivo principal para criar a Infra na AWS é o `aws_provisioning.yml`, mas ao abri-lo vai se deparar com outros 4 arquivos precedidos pelo comando `import_playbook`. Ele foi construído dessa forma para que seja executado apenas um playbook que crie toda a Infra, mas que também fique segmentado e mais fácil para entender.

```
! aws_provisioning.yml ×
playbooks > ! aws_provisioning.yml
1 # Criando Infra na AWS - Key Pair e VPC com suas dependencias + EC2 Jenkins com ECR + EC2 Produção Com S3 e IAM user + EC2 Homolog Com S3 e IAM User
2 - import_playbook: aws_provisioning_vpc.yml
3 - import_playbook: aws_provisioning_jenkins.yml
4 - import_playbook: aws_provisioning_producao.yml
5 - import_playbook: aws_provisioning_homolog.yml
```

- Playbook `aws_provisioning_vpc.yml`

```
! aws_provisioning_vpc.yml ×
playbooks > ! aws_provisioning_vpc.yml
1 - name: "Ansible para configurar criar e configurar VPC + criar e salvar Key Pair EC2"
2   hosts: local
3   gather_facts: False
4   vars_files:
5     - vars/aws_credentials.yml
6     - vars/aws.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 o outro (`vars/aws.yml`) tem as variáveis que serão usadas nas tasks. Como sempre, segue um link da [Documentação Oficial](#) que fala mais sobre variáveis.



```
! aws.yml ×
playbooks > vars > ! aws.yml
1 projeto: dh-pi-devops-devopers
2 AWS_access_key: "{{ AWSAccessKeyId }}"
3 AWS_secret_key: "{{ AWSSecretKey }}"
4 instance_type: t2.micro
5 security_group: "{{ projeto }}-ws-sg"
6 image_ami: ami-07ebfd5b3428b6f4d
7 KeyPairDev: "{{ projeto }}-key"
8 region: us-east-1
9 vpc_cidr_block: 10.0.0.0/16
10 cidr_block: 10.0.1.16/28
11 vpc_name: "{{ projeto }}-vpc"
12 route_tag: "{{ projeto }}-route"
13 sub_tag: "{{ projeto }}-subnet"
```

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](#) 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.](#)

[Link útil sobre VPC e Route Tables.](#)

- 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](#).

```
tasks:
  - name: Amazon EC2 | Create Key Pair
    ec2_key:
      aws_access_key: "{{ AWS_access_key }}"
      aws_secret_key: "{{ AWS_secret_key }}"
      name: "{{ KeyPairDev }}"
      register: ec2_key_result

  - name: Save private key
    copy: content="{{ ec2_key_result.key.private_key }}" dest="../{{ KeyPairDev }}.pem" mode=0600
    when: ec2_key_result.changed
```

[Link útil - módulo ansible key pair.](#)

- 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](#). 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 ambiente poderia ser mais segmentado, por exemplo, na vpc criaríamos 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.

```
- name: create VPC
  ec2_vpc_net:
    state: "present"
    name: "{{ vpc_name }}"
    cidr_block: "{{ vpc_cidr_block }}"
    region: "{{ region }}"
    aws_access_key: "{{ AWS_access_key }}"
    aws_secret_key: "{{ AWS_secret_key }}"
    resource_tags: { "Environment": "PI-AcademiaNeon" }
  register: create_vpc

- name: "set fact: VPC ID"
  set_fact:
    vpc_id: "{{ create_vpc.vpc.id }}"
```

[Link útil - módulo ansible ec2-vpc.](#)

[Link útil - módulo ansible ec2-subnet.](#)

[Link útil - módulo ansible ec2-securitygroup.](#)

[Link útil - módulo ansible ec2-internetgateway.](#)

[Link útil - módulo ansible ec2-routetable.](#)

- Playbooks `aws_provisioning_jenkins`, `aws_provisioning_homolog` e `aws_provisioning_producao`.

Nesses 3 playbooks são criados 3 (uma em cada um) [Amazon 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](#) - Amazon Elastic Container Registry, que é um serviço totalmente gerenciado que armazena imagens de containers 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.](#)

```
! aws_provisioning_jenkins.yml ×
playbooks > ! aws_provisioning_jenkins.yml
1  - name: "Ansible para configurar VM Jenkins"
2    hosts: local
3    gather_facts: False
4    vars_files:
5      - vars/aws_credentials.yml
6      - vars/aws.yml
7
8    vars:
9      name_service: devopers-jenkins
10
11   tasks:
12     - name: Launch the new EC2 Instance 22
13       ec2_instance:
14         aws_access_key: "{{ AWS_access_key }}"
15         aws_secret_key: "{{ AWS_secret_key }}"
16         name: "{{ name_service }}"
17         key_name: "{{ KeyPairDev }}"
18         security_group: "{{ security_group }}"
19         vpc_subnet_id: "{{ vpc_subnet_ids }}"
20         instance_type: "{{ instance_type }}"
21         image_id: "{{ image_ami }}"
22         network:
23           assign_public_ip: true
24         state: running
25         region: "{{ region }}"
26       register: ec2
27
28     - name: Wait for SSH to come up
29       wait_for:
30         host: "{{ item.public_dns_name }}"
31         port: 22
32         state: started
33       with_items: "{{ ec2.instances }}"
34
35     - name: Create AWS ECR
36       ecs_ecr:
37         name: digitalhouse-devops-app
38         aws_access_key: "{{ AWS_access_key }}"
39         aws_secret_key: "{{ AWS_secret_key }}"
40         region: "{{ region }}"
41       register: ecr
```

2. `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](#) - 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](#) - 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: 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.](#)

[Link útil - módulo ansible aws s3.](#)

[Link útil - módulo ansible s3 bucket.](#)

[Link útil - módulo ansible ec2.](#)

3. `aws_provisioning_produção.yml`: Ele é idêntico 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](#) 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.

```
! aws_provisioning_homolog.yml ×
playbooks > ! aws_provisioning_homolog.yml
1 - name: "Ansible para configurar VM Homolog"
2   hosts: local
3   gather_facts: False
4   vars_files:
5     - vars/aws_credentials.yml
6     - vars/aws.yml
7
8   vars:
9     name_service: devopers-homolog
10    name_aim_user_s3: dh_devopers_homolog
11    name_bucket_s3: dh-devopers-homolog
12
13  tasks:
14    - name: Launch the new EC2 Instance 22
15      ec2_instance:
16        aws_access_key: "{{ AWS_access_key }}"
17        aws_secret_key: "{{ AWS_secret_key }}"
18        name: "{{ name_service }}"
19        key_name: "{{ KeyPairDev }}"
20        security_group: "{{ security_group }}"
21        vpc_subnet_id: "{{ vpc_subnet_ids }}"
22        instance_type: "{{ instance_type }}"
23        image_id: "{{ image_ami }}"
24        network:
25          assign_public_ip: true
26        state: running
27        region: "{{ region }}"
28        register: ec2
29
30    - name: Wait for SSH to come up
31      wait_for:
32        host: "{{ item.public_dns_name }}"
33        port: 22
34        state: started
35        with_items: "{{ ec2.instances }}"
36
37    - name: Create iam user "{{ name_aim_user_s3 }}"
38      iam_user:
39        aws_access_key: "{{ AWS_access_key }}"
40        aws_secret_key: "{{ AWS_secret_key }}"
41        name: "{{ name_aim_user_s3 }}"
42        managed_policy:
43          - arn:aws:iam:aws:policy/AmazonS3FullAccess
44        state: present
45        register: iuser
46
```

```
! aws_provisioning_producao.yml ×
playbooks > ! aws_provisioning_producao.yml
1 - name: "Ansible para configurar VM Producao"
2   hosts: local
3   gather_facts: False
4   vars_files:
5     - vars/aws_credentials.yml
6     - vars/aws.yml
7
8   vars:
9     name_service: devopers-producao
10    name_aim_user_s3: dh_devopers_producao
11    name_bucket_s3: dh-devopers-producao
12
13  tasks:
14    - name: Launch the new EC2 Instance 22
15      ec2_instance:
16        aws_access_key: "{{ AWS_access_key }}"
17        aws_secret_key: "{{ AWS_secret_key }}"
18        name: "{{ name_service }}"
19        key_name: "{{ KeyPairDev }}"
20        security_group: "{{ security_group }}"
21        vpc_subnet_id: "{{ vpc_subnet_ids }}"
22        instance_type: "{{ instance_type }}"
23        image_id: "{{ image_ami }}"
24        network:
25          assign_public_ip: true
26        state: running
27        region: "{{ region }}"
28        register: ec2
29
30    - name: Wait for SSH to come up
31      wait_for:
32        host: "{{ item.public_dns_name }}"
33        port: 22
34        state: started
35        with_items: "{{ ec2.instances }}"
36
37    - name: Create iam user "{{ name_aim_user_s3 }}"
38      iam_user:
39        aws_access_key: "{{ AWS_access_key }}"
40        aws_secret_key: "{{ AWS_secret_key }}"
41        name: "{{ name_aim_user_s3 }}"
42        managed_policy:
43          - arn:aws:iam:aws:policy/AmazonS3FullAccess
44        state: present
45        register: iuser
46
```

```
! aws_provisioning_homolog.yml ×
playbooks > ! aws_provisioning_homolog.yml
1 - name: "Ansible para configurar VM Homolog"
2   hosts: local
3   gather_facts: False
4   vars_files:
5     - vars/aws_credentials.yml
6     - vars/aws.yml
7
8   vars:
9     name_service: devopers-homolog
10    name_aim_user_s3: dh_devopers_homolog
11    name_bucket_s3: dh-devopers-homolog
12
13  tasks:
14    - name: Launch the new EC2 Instance 22--
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30    - name: Wait for SSH to come up--
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46    - name: Create iam user "{{ name_aim_user_s3 }}"--
47
48
49
50
51
52
53    - name: Create a buckets
54      aws_s3:
55        bucket: "{{ name_bucket_s3 }}"
56        mode: create
57        region: "{{ region }}"
58
59    - name: Create S3
60      s3_bucket:
61        aws_access_key: "{{ AWS_access_key }}"
62        aws_secret_key: "{{ AWS_secret_key }}"
63        name: "{{ name_bucket_s3 }}"
64        region: "{{ region }}"
65        state: present
66        policy:
67          Version: '2012-10-17'
68          Statement:
69            - Effect: Allow
70              Principal:
71                'AWS' : "{{ iuser.iam_user.user.arn }}"
72              Action: [
73                's3:ListBucket',
74                's3:DeleteObject',
75                's3:GetObject',
76                's3:PutObject',
77                's3:PutObjectAcl'
78              ]
79              Resource: [
80                'arn:aws:s3:::{{name_bucket_s3}}/*',
81                'arn:aws:s3:::{{name_bucket_s3}}'
82              ]
83
```

```
! aws_provisioning_producao.yml ×
playbooks > ! aws_provisioning_producao.yml
1 - name: "Ansible para configurar VM Producao"
2   hosts: local
3   gather_facts: False
4   vars_files:
5     - vars/aws_credentials.yml
6     - vars/aws.yml
7
8   vars:
9     name_service: devopers-producao
10    name_aim_user_s3: dh_devopers_producao
11    name_bucket_s3: dh-devopers-producao
12
13  tasks:
14    - name: Launch the new EC2 Instance 22--
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30    - name: Wait for SSH to come up--
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46    - name: Create iam user "{{ name_aim_user_s3 }}"--
47
48
49
50
51
52
53    - name: Create a buckets
54      aws_s3:
55        bucket: "{{ name_bucket_s3 }}"
56        mode: create
57        region: "{{ region }}"
58
59    - name: Create S3
60      s3_bucket:
61        aws_access_key: "{{ AWS_access_key }}"
62        aws_secret_key: "{{ AWS_secret_key }}"
63        name: "{{ name_bucket_s3 }}"
64        region: "{{ region }}"
65        state: present
66        policy:
67          Version: '2012-10-17'
68          Statement:
69            - Effect: Allow
70              Principal:
71                'AWS' : "{{ iuser.iam_user.user.arn }}"
72              Action: [
73                's3:ListBucket',
74                's3:DeleteObject',
75                's3:GetObject',
76                's3:PutObject',
77                's3:PutObjectAcl'
78              ]
79              Resource: [
80                'arn:aws:s3:::{{name_bucket_s3}}/*',
81                'arn:aws:s3:::{{name_bucket_s3}}'
82              ]
83
```

- 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](#) que coleta informações das instâncias, ou seja, podemos incluir quantas máquinas 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.](#)

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`.

```
playbooks > vars > ! aws.yml
1 projeto: dh-pi-devops-devopers
2 AWS_access_key: "{{ AWSAccessKeyId }}"
3 AWS_secret_key: "{{ AWSSecretKey }}"
4 instance_type: t2.micro
5 security_group: "{{ projeto }}-ws-sg"
6 image_ami: ami-07ebfd5b3428b6f4d
7 KeyPairDev: "{{ projeto }}-key"
8 region: us-east-1
9 vpc_cidr_block: 10.0.0.0/16
10 cidr_block: 10.0.1.16/28
11 vpc_name: "{{ projeto }}-vpc"
12 route_tag: "{{ projeto }}-route"
13 sub_tag: "{{ projeto }}-subnet"

[aws_ec2:vars]
ansible_user=ubuntu
ansible_connection=ssh
ansible_ssh_private_key_file=./dh-pi-devops-devopers-key.pem
ansible_python_interpreter=/usr/bin/python3
```

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...

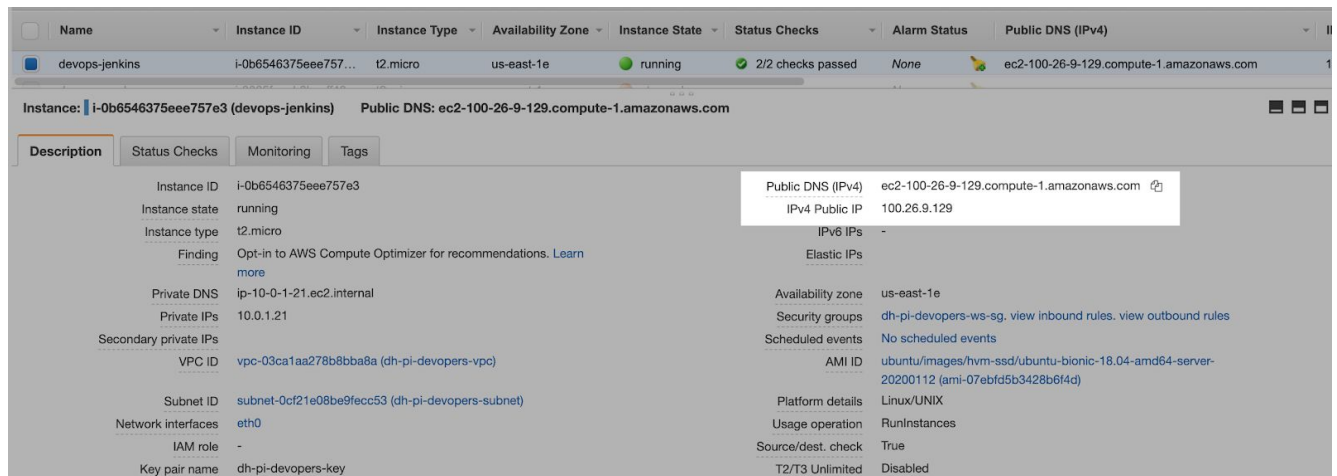
```
$ ansible-playbook playbooks/install_jenkins_ec2-jenkins.yml
```

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
```

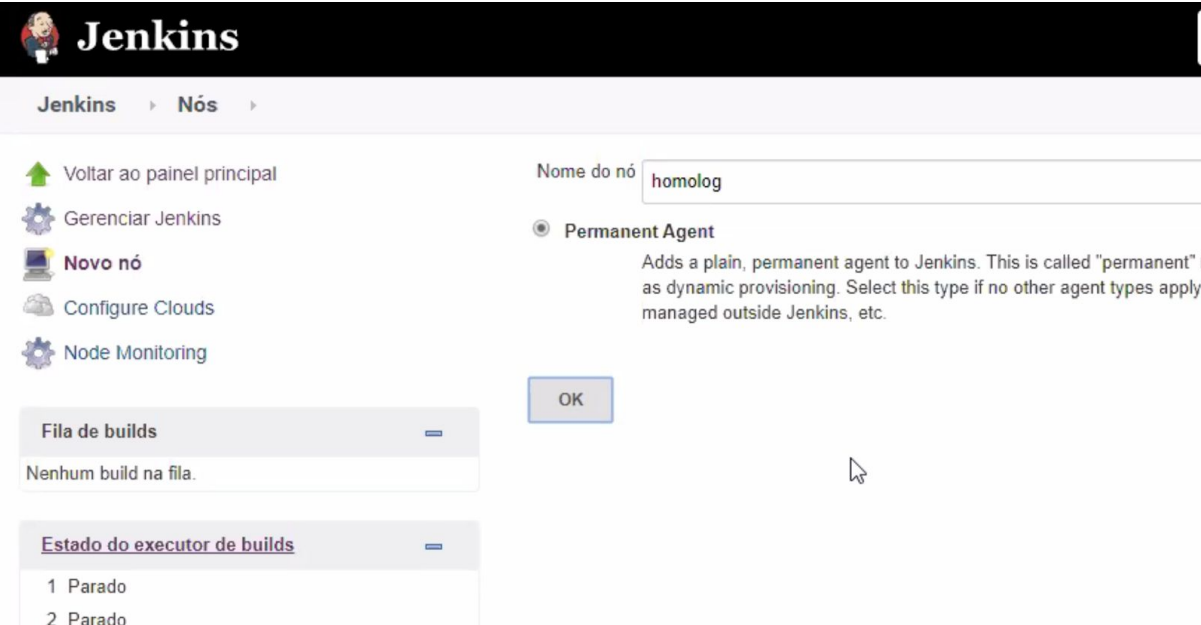
```
ubuntu@ubuntu-bionic:~/ProjetoIntegrador_AcademiaNeon$ ls
Jenkinsfile LICENSE README.md ansible.cfg dh-pi-devops-devopers-key.pem docs inventory playbooks
ubuntu@ubuntu-bionic:~/ProjetoIntegrador_AcademiaNeon$ ssh -i dh-pi-devops-devopers-key.pem ubuntu@ec2-100-26-9-129.compute-1.amazonaws.com
```

```
ubuntu@ubuntu-bionic:~/ProjetoIntegrador_AcademiaNeon$ ls
Jenkinsfile LICENSE README.md ansible.cfg dh-pi-devops-devopers-key.pem docs inventory playbooks
ubuntu@ubuntu-bionic:~/ProjetoIntegrador_AcademiaNeon$ ssh -i dh-pi-devops-devopers-key.pem ubuntu@100.26.9.129
```

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**



The screenshot shows the Jenkins web interface. At the top is the Jenkins logo and name. Below it is a breadcrumb trail: Jenkins > Nós >. On the left sidebar, there are links: Voltar ao painel principal (with a green arrow icon), Gerenciar Jenkins (with a gear icon), Novo nó (with a laptop icon and highlighted in blue), Configure Clouds (with a cloud icon), and Node Monitoring (with a gear icon). The main content area is titled 'Novo nó' and contains a form. The 'Nome do nó' field has the value 'homolog'. Below this, the 'Permanent Agent' option is selected with a radio button. A description for 'Permanent Agent' reads: 'Adds a plain, permanent agent to Jenkins. This is called "permanent" as dynamic provisioning. Select this type if no other agent types apply managed outside Jenkins, etc.' There is an 'OK' button below the description. On the left side of the main content area, there are two expandable sections: 'Fila de builds' (collapsed) showing 'Nenhum build na fila.' and 'Estado do executor de builds' (collapsed) showing a list with two items, both labeled 'Parado'.

Jenkins > Nós

- [Voltar ao painel principal](#)
- [Gerenciar Jenkins](#)
- [Novo nó](#)
- [Configure Clouds](#)
- [Node Monitoring](#)

Fila de builds

Nenhum build na fila.

Estado do executor de builds

1 Parado
2 Parado

Nome: homolog

Descrição:

Número de executores: 1

Diretório root remoto: /home/ubuntu

Rótulos:

Uso: Use this node as much as possible

Método de lançamento: Launch agents via SSH

Host: ec2-54-166-144-223.compute-1.amazonaws.com

Credentials: ubuntu

Host Key Verification Strategy: Non verifying Verification Strategy

Controls how Jenkins verifies the SSH key presented by the remote host whilst connecting. (de SSH Build Agents plugin)

Disponibilidade: Keep this agent online as much as possible

Propriedades dos nós

☐ Ferramentas locais

☐ Variáveis de ambiente

Salvar

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

Add Credentials

Domain: Global credentials (unrestricted)

Kind: SSH Username with private key

Scope: Global (Jenkins, nodes, items, all child items, etc)

ID: ubuntukeyaws

An internal unique ID by which these credentials are identified from jobs and other configuration. Normally left blank, in which case an ID will be generated, which is fine for jobs created using visual forms. Useful to specify explicitly when using credentials from scripted configuration. (de SSH Credentials Plugin)

Description:

Username: ubuntu

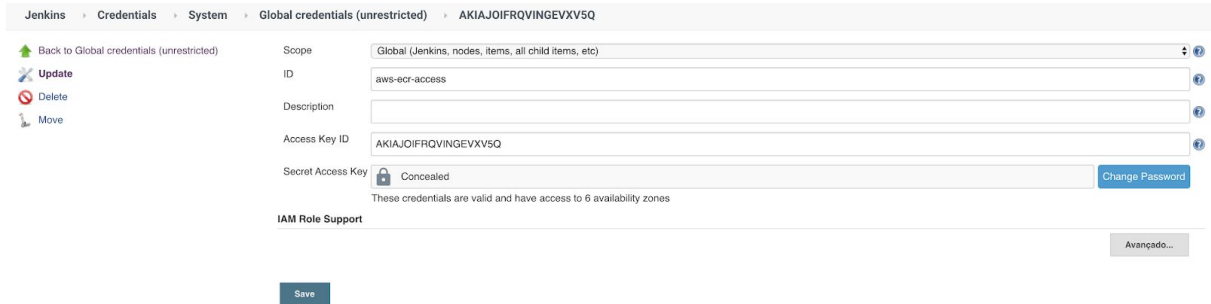
Private Key: Enter directly

Key: -----BEGIN RSA PRIVATE KEY-----
MIIEpQIBAAKCAQEAg6EULgspU5ACZBP0WAj5BRoI9DwxfZfgCYvY2j3MY15wpQzcS4VKxMj51bqjT
bohSwUt4sreTihUM9xenvwhGinHUFc1Scxahxz4iNB4OHITdias+PObbHMc1MlsAs7It9C29UUZ

Passphrase:

Add Cancel

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



The screenshot shows the Jenkins 'Add Credentials' form for a new AWS credential. The breadcrumb trail at the top is 'Jenkins > Credentials > System > Global credentials (unrestricted) > AKIAJOIFRQVINGEVXV5Q'. On the left, there are navigation links: 'Back to Global credentials (unrestricted)', 'Update', 'Delete', and 'Move'. The main form fields are: 'Scope' (dropdown menu set to 'Global (Jenkins, nodes, items, all child items, etc)'), 'ID' (text field with 'aws-ecr-access'), 'Description' (empty text field), 'Access Key ID' (text field with 'AKIAJOIFRQVINGEVXV5Q'), and 'Secret Access Key' (password field with a lock icon and 'Concealed' text, with a 'Change Password' button). Below the 'Secret Access Key' field, a message states: 'These credentials are valid and have access to 6 availability zones'. At the bottom left is a 'Save' button, and at the bottom right is an 'Avançado...' button. The 'IAM Role Support' section is partially visible at the bottom.

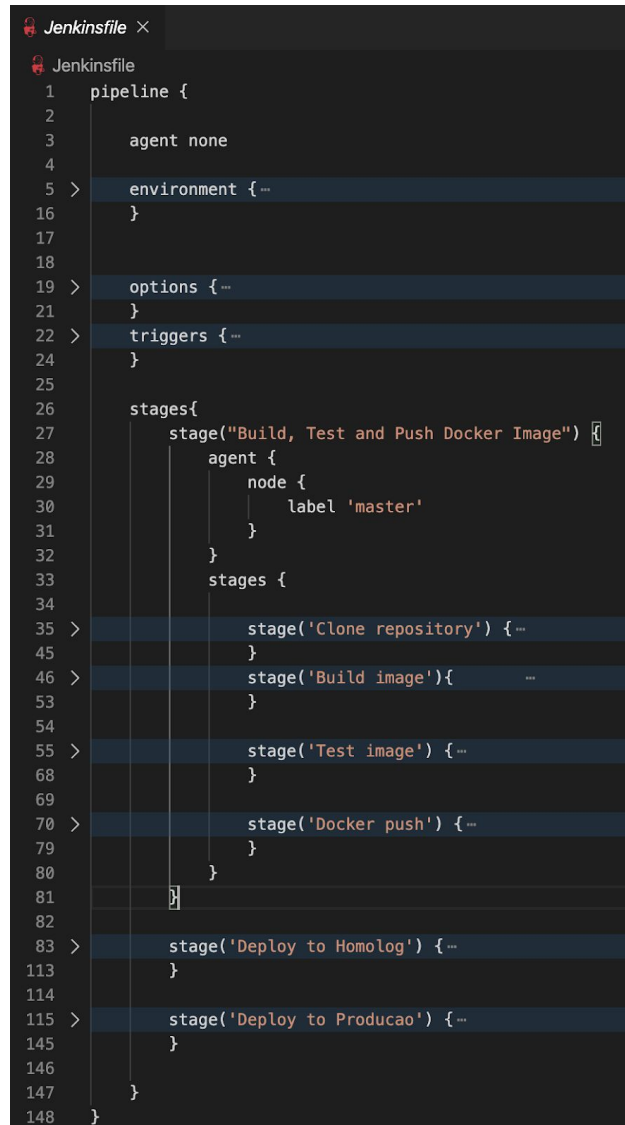
Links úteis sobre o uso de Credenciais AWS com Jenkins:

[Pushing to ECR Using Jenkins Pipeline Plugin](#)

[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.



```
1 pipeline {
2
3   agent none
4
5   > environment { ...
16   }
17
18
19   > options { ...
21   }
22   > triggers { ...
24   }
25
26   stages{
27     stage("Build, Test and Push Docker Image") {
28       agent {
29         node {
30           label 'master'
31         }
32       }
33       stages {
34
35         > stage('Clone repository') { ...
45         }
36         > stage('Build image'){ ...
53         }
37
38         > stage('Test image') { ...
68         }
39
40         > stage('Docker push') { ...
79         }
41       }
42     }
43
44     > stage('Deploy to Homolog') { ...
113     }
45
46     > stage('Deploy to Producao') { ...
145     }
47   }
48 }
```

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

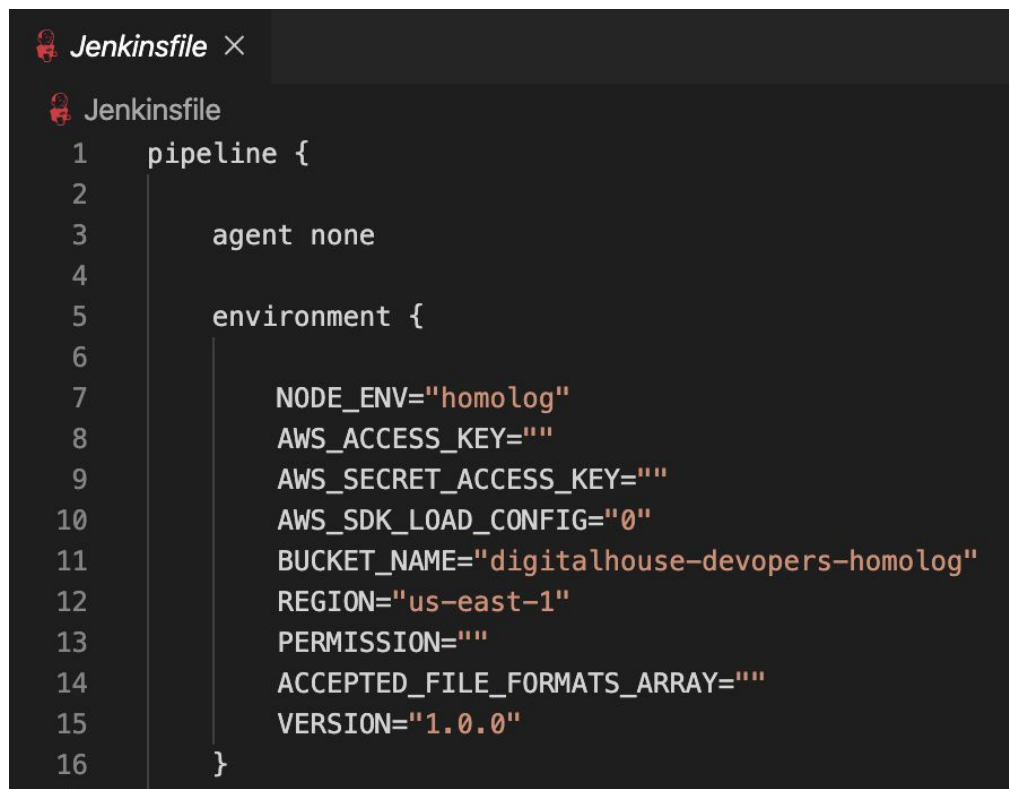
- ***stage('Clone repository')***: Clonar o repositório que vamos definir no pipeline.
- ***stage('Build image')***: Fazer o build da imagem.
- ***stage('Test image')***: Subir um container da imagem na porta informada e fazer o healthcheck.
- ***stage('Docker push')***: Subir a imagem no AWS ECR e versionar.

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.



```
Jenkinsfile
1 pipeline {
2
3     agent none
4
5     environment {
6
7         NODE_ENV="homolog"
8         AWS_ACCESS_KEY=""
9         AWS_SECRET_ACCESS_KEY=""
10        AWS_SDK_LOAD_CONFIG="0"
11        BUCKET_NAME="digitalhouse-devopers-homolog"
12        REGION="us-east-1"
13        PERMISSION=""
14        ACCEPTED_FILE_FORMATS_ARRAY=""
15        VERSION="1.0.0"
16    }
```

- 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).

```
Jenkinsfile x
Jenkinsfile
35 stage('Clone repository') {
36     steps {
37         script {
38             if(env.GIT_BRANCH=='origin/master'){
39                 checkout scm
40             }
41             sh('printenv | sort')
42             echo "My branch is: ${env.GIT_BRANCH}"
43         }
44     }
45 }
46 stage('Build image'){
47     steps {
48         script {
49             print "Environment will be : ${env.NODE_ENV}"
50             docker.build("digitalhouse-devops-app:latest")
51         }
52     }
53 }
54
55 stage('Test image') {
56     steps {
57         script {
58
59             docker.image("digitalhouse-devops-app:latest").withRun('-p 3000:3000') { c ->
60                 sh 'docker ps'
61                 sh 'sleep 10'
62                 sh 'curl http://127.0.0.1:3000/api/v1/healthcheck'
63             }
64         }
65     }
66 }
67
68 }
```

- 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).




```

Jenkinsfile
68
69
70     stage('Docker push') {
71         steps {
72             echo 'Push latest para AWS ECR'
73             script {
74                 docker.withRegistry('https://733036961943.dkr.ecr.us-east-1.amazonaws.com', 'ecr:us-east-1:aws-ecr-access') {
75                     docker.image('digitalhouse-devops-app').push()
76                 }
77             }
78         }
79     }
80
81 }

```

- 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 precisam 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-devops-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.

```

Jenkinsfile
83     stage('Deploy to Homolog') {
84         agent {
85             node {
86                 label 'homolog'
87             }
88         }
89
90         steps {
91             script {
92                 if(env.GIT_BRANCH=='origin/master'){
93
94                     docker.withRegistry('https://733036961943.dkr.ecr.us-east-1.amazonaws.com', 'ecr:us-east-1:aws-ecr-access') {
95                         docker.image('digitalhouse-devops-app').pull()
96                     }
97
98                     echo 'Deploy para Homologação'
99                     sh "hostname"
100                     catchError {
101                         sh "docker stop app_homolog"
102                         sh "docker rm app_homolog"
103                     }
104                     sh "docker run -d --env NODE_ENV=homolog --env BUCKET_NAME=digitalhouse-devops-homolog --name app_homolog -p 3000:3000 733036961943.dkr.ecr.us-east-1.amazonaws.com/digitalhouse-devops-app:latest"
105                     sh "docker ps"
106                     sh 'sleep 10'
107                     sh 'curl http://127.0.0.1:3000/api/v1/healthcheck'
108                 }
109             }
110         }
111     }
112
113 }

```

- 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 idêntica 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 precisam 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-devops-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 → repositório aws ecr (só copiar a url) + versão que vamos utilizar.

```

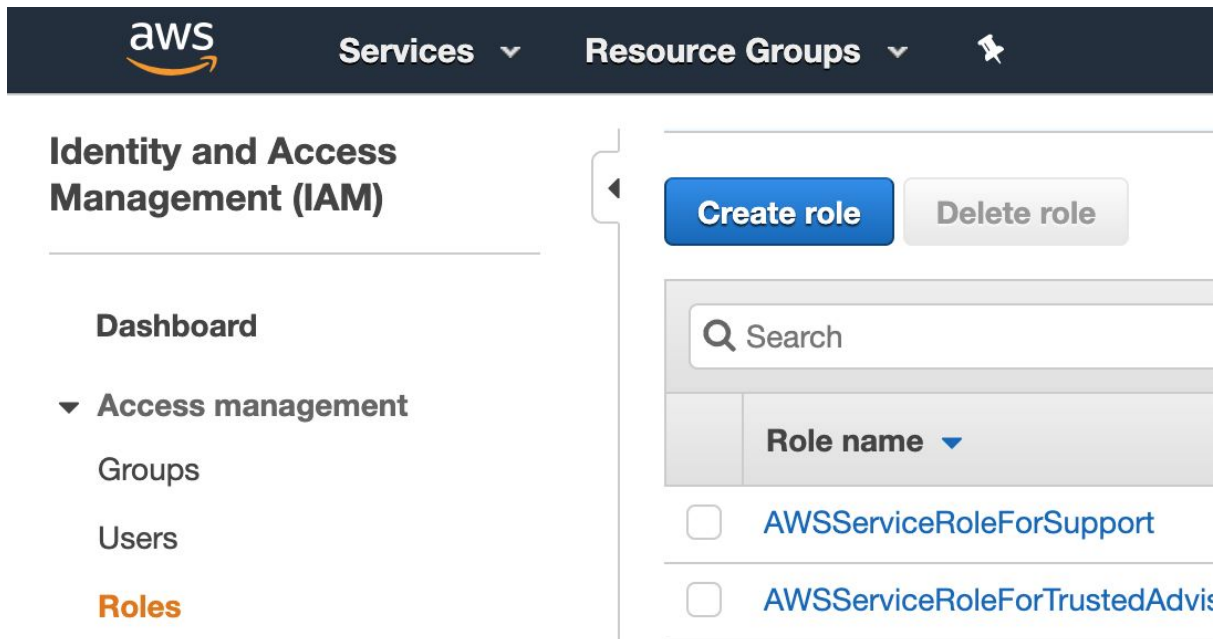
115 stage('Deploy to Producao') {
116     agent {
117         node {
118             label 'producao'
119         }
120     }
121 }
122
123 steps {
124     script {
125         if(env.GIT_BRANCH=='origin/master'){
126             docker.withRegistry('https://733036961943.dkr.ecr.us-east-1.amazonaws.com', 'ecr:us-east-1:aws-ecr-access') {
127                 docker.image('digitalhouse-devops-app').pull()
128             }
129
130             echo 'Deploy para Produção'
131             sh 'hostname'
132             catchError {
133                 sh "docker stop app_prod"
134                 sh "docker rm app_prod"
135             }
136             sh "docker run -d --env NODE_ENV=producao --env BUCKET_NAME=digitalhouse-devops-producao --name app_prod -p 80:3000 733036961943.dkr.ecr.us-east-1.amazonaws.com/digitalhouse-devops-app:latest"
137             sh "docker ps"
138             sh 'sleep 10'
139             sh 'curl http://127.0.0.1:80/api/v1/healthcheck'
140         }
141     }
142 }
143
144 }
145

```

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

Create role

Select type of trusted entity

1 2 3 4

AWS service
EC2, Lambda and others

Another AWS account
Belonging to you or 3rd party

Web identity
Cognito or any OpenID provider

SAML 2.0 federation
Your corporate directory

Allows AWS services to perform actions on your behalf. [Learn more](#)

Choose a use case

Common use cases

EC2
Allows EC2 instances to call AWS services on your behalf.

Lambda
Allows Lambda functions to call AWS services on your behalf.

Or select a service to view its use cases

| | | | | |
|---|--|---|----------------------------------|---------------------------------|
| API Gateway | CodeDeploy | EKS | IoT Things Graph | Rekognition |
| AWS Backup | CodeGuru | EMR | KMS | RoboMaker |
| AWS Chatbot | CodeStar Notifications | ElastiCache | Kinesis | S3 |
| AWS Support | Comprehend | Elastic Beanstalk | Lake Formation | SMS |
| Amplify | Config | Elastic Container Service | Lambda | SNS |
| AppStream 2.0 | Connect | Elastic Transcoder | Lex | SWF |
| AppSync | DMS | ElasticLoadBalancing | License Manager | SageMaker |
| Application Auto Scaling | Data Lifecycle Manager | Forecast | Machine Learning | Security Hub |
| Application Discovery Service | Data Pipeline | GameLift | Macie | Service Catalog |
| Batch | DataSync | Global Accelerator | MediaConvert | Step Functions |
| | DeepLens | Glue | Migration Hub | Storage Gateway |

* Required

Cancel **Next: Permissions**

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

Create role

1234

▼ Attach permissions policies

Choose one or more policies to attach to your new role.





Create policy

↺

Filter policies ▼

Q S3

Showing 4 results

| | Policy name ▼ | Used as |
|-------------------------------------|---|------------------------|
| <input type="checkbox"/> | ▶  AmazonDMSRedshiftS3Role | None |
| <input checked="" type="checkbox"/> | ▶  AmazonS3FullAccess | Permissions policy (3) |
| <input type="checkbox"/> | ▶  AmazonS3ReadOnlyAccess | None |
| <input type="checkbox"/> | ▶  QuickSightAccessForS3StorageManagementAnalyticsReadOnly | None |

▶ Set permissions boundary

* Required

Cancel

Previous

Next: Tags

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

Create role

1234

Add tags (optional)

IAM tags are key-value pairs you can add to your role. Tags can include user information, such as an email address, or can be descriptive, such as a job title. You can use the tags to organize, track, or control access for this role. [Learn more](#)

| Key | Value (optional) | Remove |
|--|----------------------|--------|
| <input type="text" value="Add new key"/> | <input type="text"/> | |

You can add 50 more tags.

Cancel

Previous

Next: Review

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

Create role

1234

Review

Provide the required information below and review this role before you create it.

Role name*

ec2roles3

Use alphanumeric and '+=,@-_' characters. Maximum 64 characters.

Role description


Allows EC2 instances to call AWS services on your behalf.

Maximum 1000 characters. Use alphanumeric and '+=,@-_' characters.

Trusted entities

AWS service: ec2.amazonaws.com

Policies

 AmazonS3FullAccess [↗](#)

Permissions boundary

Permissions boundary is not set

No tags were added.

* Required

Cancel

Previous

Create role

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.

The screenshot shows the Jenkins Job Configuration page with the 'General' tab selected. The 'Descrição' field is empty. Below it, there are several checkboxes: 'Descartar builds antigos', 'Do not allow concurrent builds', 'Do not allow the pipeline to resume if the master restarts', 'Este build é parametrizado', 'GitHub project' (which is highlighted), 'Pipeline speed/durability override', 'Preserve stashes from completed builds', and 'Throttle builds'. At the bottom, there is a 'Build Triggers' section with checkboxes for 'Construir após a construção de outros projetos' and 'Construir periodicamente'. The 'Salvar' and 'Aplicar' buttons are visible at the bottom left.

This close-up shows the 'GitHub project' checkbox checked. Below it, the 'Project url' field contains the text 'https://github.com/andresavs/digitalhouse-devops-app.git/'. To the right of the field is an 'Avançado...' button. At the bottom, the 'Salvar' and 'Aplicar' buttons are visible.

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

The screenshot shows the Jenkins configuration page with the following tabs: General, Build Triggers, Advanced Project Options, and Pipeline. The Build Triggers tab is active, showing a list of checkboxes for various build triggers. The Advanced Project Options tab is also visible, showing a button labeled 'Avançado...'. The Pipeline tab is partially visible at the bottom, showing buttons for 'Salvar' and 'Aplicar'.

Build Triggers

- ☐ Pipeline speed/durability override
- ☐ Preserve stashes from completed builds
- ☐ Throttle builds
- ☐ Construir após a construção de outros projetos
- ☐ Construir periodicamente
- ☐ GitHub hook trigger for GITScm polling
- ☐ Consultar periodicamente o SCM
- ☐ Desabilitar builds
- ☐ Período de silêncio
- ☐ Dispare builds remotamente (exemplo, a partir dos scripts)

Advanced Project Options

Avançado...

Pipeline

Salvar Aplicar

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.

The screenshot shows the Jenkins Pipeline configuration page with the following tabs: General, Build Triggers, Advanced Project Options, and Pipeline. The Pipeline tab is active, showing the 'Definition' section with a dropdown menu set to 'Pipeline script from SCM'. Below this, the 'SCM' section is set to 'Git'. The 'Repositories' section shows a 'Repository URL' field with a red error message 'Please enter Git repository.' and a 'Credentials' dropdown set to '- none -'. The 'Branches to build' section shows a 'Branch Specifier (blank for 'any')' field set to '*/master'. At the bottom, there is a 'Navegar no repositório' dropdown set to '(Auto)' and an 'Additional Behaviours' section with an 'Adicionar' button.

Pipeline

Definition: Pipeline script from SCM

SCM: Git

Repositories:

- Repository URL: (Error: Please enter Git repository.)
- Credentials: (Add button)

Branches to build:

- Branch Specifier (blank for 'any'): (Add Branch button)

Navegar no repositório: (Auto)

Additional Behaviours: Adicionar

Jenkins > PipelineDevOps-DevOps-pi >

General Build Triggers **Advanced Project Options** Pipeline

Pipeline

Definition Pipeline script from SCM

SCM Git

Repositories

Repository URL <https://github.com/andresava/digitalhouse-devops-app.git>

Credentials ubuntu

Avançado...
Add Repository

Branches to build

Branch Specifier (blank for 'any') */master

Add Branch

Navegar no repositório (Auto)

Additional Behaviours Adicionar

Script Path Jenkinsfile

Lightweight checkout ☐

[Pipeline Syntax](#)

Salvar Aplicar

[Link útil - Pipeline como código no Jenkins.](#)

APP NodeJS utilizado

Foi realizado um fork do [repositório do Desenvolvedor](#) para [repositório Dev do Projeto](#) 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/>