

Documento Técnico – Fundamentos de DevOps

Título: Trabalho Final – Fundamentos de DevOps

Aluno: Pedro Henrique Moreira Montes

Turma: BSI 5

1. Introdução

Este projeto tem como objetivo implementar uma arquitetura completa de DevOps com deploy contínuo (GitOps) utilizando ArgoCD. A solução abrange o provisionamento automatizado de uma máquina virtual, criação de um cluster Kubernetes com **kind**, deploy de uma aplicação full stack composta por frontend (HTML + JS), backend (FastAPI) e banco de dados (PostgreSQL), além da integração total via CI/CD e ArgoCD. Todas as aplicações são empacotadas em containers Docker e gerenciadas via manifests Kustomize.

2. Escolha do Ambiente

Justificativa da Escolha do Ambiente

Para o desenvolvimento deste projeto, optei por utilizar uma infraestrutura baseada em **máquina virtual provisionada com Vagrant e VirtualBox**, com automação via **Ansible** e execução do cluster Kubernetes usando o **Kind (Kubernetes in Docker)**. Essa escolha foi feita com base em três pilares principais: **controle total sobre o ambiente, flexibilidade na configuração e realismo para fins acadêmicos e profissionais**.

1. Isolamento e Reprodutibilidade

Ao optar por uma **VM dedicada**, criei um ambiente totalmente isolado do sistema operacional host, evitando conflitos de dependências e garantindo reprodutibilidade. O uso do Vagrant permite levantar esse ambiente com um único comando, garantindo que qualquer pessoa consiga recriar o mesmo cenário com as mesmas configurações de rede, memória, CPU e sistema operacional.

2. Automação com Ansible

A automação foi um critério essencial. Com o Ansible, foi possível provisionar automaticamente tudo o que o projeto exigia: Docker, Kind, Kubectl, ArgoCD, Nginx, entre outras ferramentas. Essa abordagem permitiu reduzir erros humanos, economizar tempo e tornar o processo mais profissional, seguindo boas práticas DevOps.

3. Simulação realista de um ambiente de produção

Mesmo sendo um ambiente local, a arquitetura proposta simula de forma fiel um cenário real de deploy contínuo: com provisionamento de infraestrutura, criação de cluster Kubernetes, deploy automatizado com GitOps (ArgoCD) e aplicação full stack funcionando com integração completa. A escolha de ferramentas open-source, amplamente usadas no mercado, reforça a aplicabilidade do que foi feito.

4. Acessibilidade e controle

Ao utilizar o modo **bridge** na VM, foi possível expor os serviços (frontend, backend, banco, ArgoCD) para serem acessados diretamente via navegador na rede local, facilitando os testes e validações da aplicação sem depender de serviços externos ou em nuvem.

3. Provisionamento

- **Ferramentas utilizadas:** Vagrant + Ansible
- **Justificativa:**

Vagrant foi usado para criar uma VM padronizada, isolada e fácil de iniciar com um único comando (`vagrant up`).

Ansible automatizou a instalação das ferramentas (Docker, Kind, kubectl, ArgoCD etc.) de forma organizada, sem precisar configurar manualmente.

A combinação permite reproduzir o ambiente de forma rápida, confiável e sem depender do sistema operacional da máquina host.

- **Scripts criados:**
 - **Vagrantfile** para provisionamento da VM
 - Playbooks Ansible para instalar dependências (Docker, Kind, kubectl, ArgoCD)

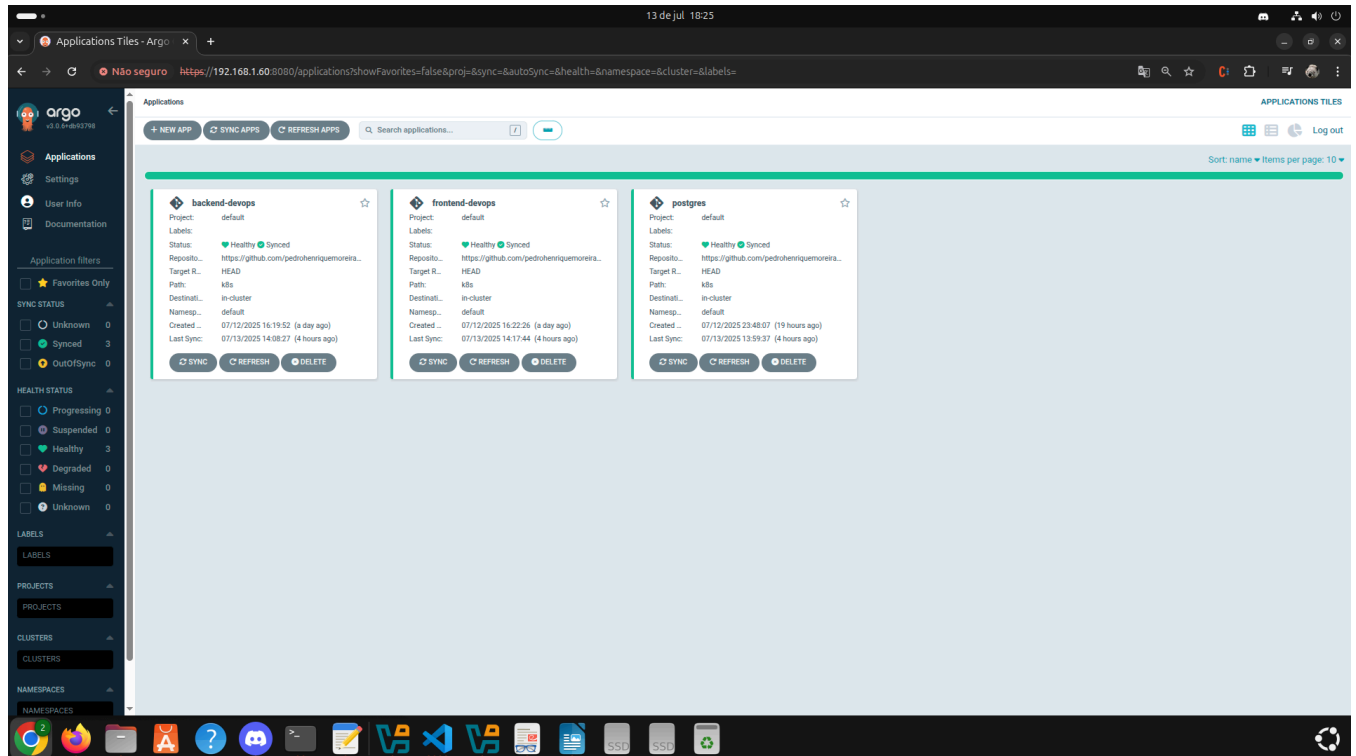
4. Cluster Kubernetes

- **Ferramenta usada:** `kind` (Kubernetes in Docker)
- **Configuração dos nós:**
 - 1 nó de controle (único nó no cluster)
 - Exposição de portas para comunicação entre os serviços
- **Testes realizados:**
 - `kubectl get pods`, `kubectl port-forward`, `kubectl logs`
 - Verificação da saúde dos serviços com `/ping` e `/db`

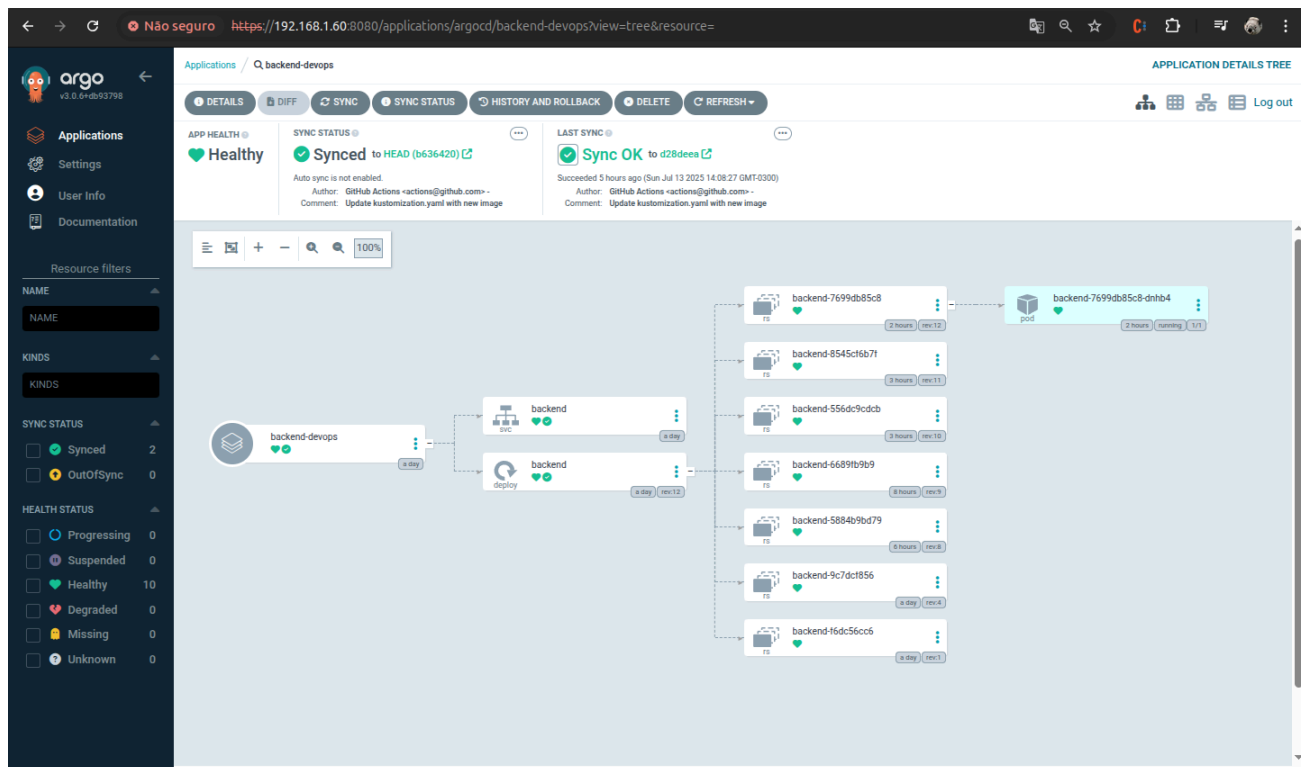
5. GitOps com ArgoCD

- **Instalação:** Aplicada via Ansible com manifest oficial (`install_argocd.yaml`)
- **Repositórios Git criados:**
 - [Infraestrutura do projeto](#)
 - [frontend-devops](#)
 - [backend-devops](#)
 - [devops-banco-dados](#)
- **Deploy automatizado:** Utilização do `application.yaml` para cada componente com `sync` e `auto-prune`
- **Screenshots:**

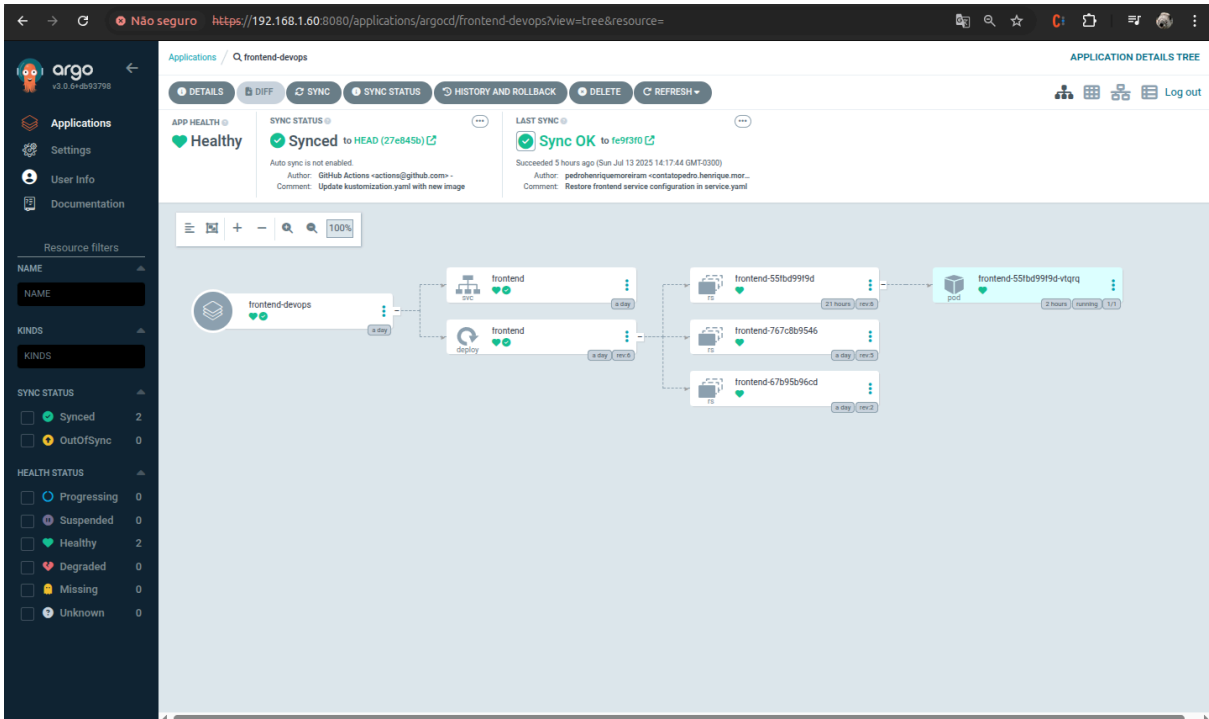
Print da tela inicial do ArgoCD:



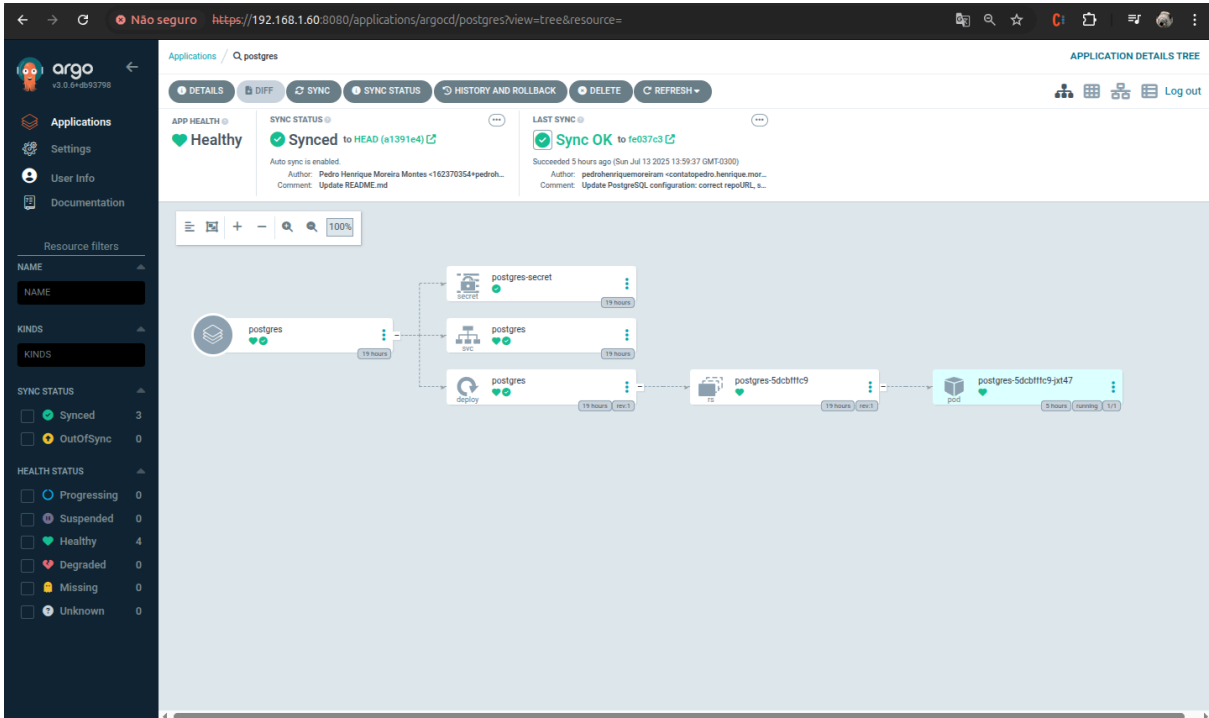
Print do backend-devops:



Print do Frontend-devops:



Print do PostgreSQL:



6. Aplicação

Backend (FastAPI)

Justificativa pelo uso de FastAPI:

- É um framework **leve, rápido e moderno** baseado em Python 3.
- Permite criar APIs REST de forma rápida e com **ótima performance**.
- Ideal para projetos acadêmicos e profissionais com foco em **produtividade e clareza**.
- Conecta ao PostgreSQL e expõe dois endpoints: `/` e `/db`
- Comunicação validada com SQLAlchemy e posteriormente com `psycopg2` para conexão direta
- Habilitado suporte a CORS para frontend

Banco de Dados (PostgreSQL)

Justificativa pela escolha de PostgreSQL:

- É um banco de dados **relacional, robusto e confiável**, amplamente usado no mercado.
- Suporta **tipos complexos, integridade de dados e alta performance**.
- É gratuito, open-source e se integra facilmente com Python (via `psycopg2` ou `SQLAlchemy`).
- Excelente para **ambientes Docker/Kubernetes**, com imagem oficial estável.
- Deploy via imagem oficial `postgres:16`
- Variáveis sensíveis injetadas via `Secret`
- Exposto via `NodePort` para testes externos

Frontend (HTML + JS via Nginx)

- Página simples com botão que faz `fetch` ao endpoint `/db` do backend
- Comunicação validada com mensagens de sucesso e erro
- Exposição feita com `NodePort` e porta 8282

O serviço do Frontend, Backend e PostgreSQL foram integrados da seguinte forma:

Frontend (port 8282) → Backend (port 8181) → PostgreSQL (port 5432)

7. Como Reproduzir o Projeto?

Abaixo está o passo a passo completo para reproduzir este projeto em qualquer ambiente compatível com Linux.

Pré-requisitos





- Git
- VirtualBox
- Vagrant
- Docker
- Python 3.11+
- `kubect1`
- `kind`
- `Ansible`
- Acesso à internet

Etapas:

1. Clone os repositórios:

Repositórios do Projeto

Este projeto é dividido em quatro repositórios, cada um com uma responsabilidade específica:

Repositório	Descrição	Observações
 PACKER-VAGRANT-VIRTUALBOX-MAIN	Contém toda a infraestrutura do projeto: scripts de provisionamento com Packer, Vagrant, Ansible, além dos manifests do Kubernetes.	É o único repositório que exige modificação. Ele orquestra toda a infraestrutura.
 devops-backend	Backend da aplicação, feito com FastAPI. Possui rota <code>/db</code> para teste de conexão com o banco.	Apenas clone, a imagem já está publicada no Docker Hub.
 devops-frontend	Frontend simples com HTML e JS, que se comunica com o backend.	Apenas clone, a imagem já está publicada no Docker Hub.
 devops-banco-dados	Repositório com manifest do PostgreSQL e secret com usuário/senha.	Clone para visualização, mas não precisa ser alterado.

Repositório Infraestrutura:

git clone <https://github.com/pedrohenriquemoreiram/packer-vagrant-virtualbox-main.git>

Repositório Frontend:

git clone <https://github.com/pedrohenriquemoreiram/frontend-devops.git>

Repositório Backend:

git clone <https://github.com/pedrohenriquemoreiram/backend-devops.git>

Repositório Banco de dados:

git clone <https://github.com/pedrohenriquemoreiram/devops-banco-dados.git>

Nota: Após clonar os repositórios, se você tentar fazer vagrant up para provisionar a máquina dentro da pasta da infraestrutura, pode ocorrer um erro pois falta dependências, como vagrant, packer. Caso não tenha, por favor instale-os dentro da pasta da infraestrutura, certifique-se de ter o Virtualbox instalado também.

Packer: Certifique-se de ter o Packer instalado em sua máquina. Você pode baixar a versão mais recente do Packer em [pakcerio.io](https://packer.io) em formato binário.

Vagrant: Instale o Vagrant em sua máquina. Você pode encontrar as instruções de instalação em vagrantup.com em formato binário.

VirtualBox: Instale o VirtualBox, que é o provedor de virtualização utilizado pelo Vagrant. Você pode baixar o VirtualBox em [virtualbox.org](https://www.virtualbox.org)

Os binários devem estar na mesma pasta que o arquivo packer.pkr.hcl.

Além disso, será necessário que seu computador tenha tanto Python, quanto Ansible instalados em seu computador.

2. Executar os comandos na seguinte ordem:

```
packer init .
```

```
packer plugin install github.com/hashicorp/virtualbox
```

```
packer plugin install github.com/hashicorp/vagrant
```

```
packer build debian.json - Irá fazer o provisionamento da máquina.
```

```
vagrant box add debian12 debian12.box
```

Dentro da pasta do projeto, aplique o comando para provisionar a máquina:

```
vagrant up
```

Isso irá provisionar sua máquina virtual.

NÃO MEXA E NÃO APERTE NENHUMA TECLA DURANTE O PROVISIONAMENTO, POIS TUDO É FEITO DE FORMA AUTOMÁTICA POR CAUSA DOS ARQUIVOS “preseed.cfg” e “debian.json” que são os responsáveis pela configuração de sua máquina.

No terminal da máquina hospedeira:

ssh-keygen # Não precisa preencher nenhuma opção

ssh-copy-id -i <caminho da chave gerada> vagrant@<ip da máquina virtual gerada>

Na pasta do ansible faça as instalações dos arquivos .yml que serão refletidos para a máquina provisionada.

Antes de fazer os comandos, entre no arquivo hosts e em **webservers 1** coloque o ip da sua máquina provisionada.

No campo “ansible_ssh_private_key_file=” coloque o caminho gerado com o ssh-keygen.

```
1  [webservers]
2  webserver1 ansible_host=192.168.1.60
3
4  [all:vars]
5  ansible_ssh_private_key_file=/home/pedropc/.ssh/id_ed25519
6  ansible_user=vagrant
```

ansible-playbook -i hosts install_nginx.yml

ansible-playbook -i hosts install_docker.yml

ansible-playbook -i hosts install_kind.yml

ansible-playbook -i install_kubectl.yml

ansible-playbook -i raise_nodes.yml

ansible-playbook -i hosts install_argocd.yml

Após fazer todas as instalações, caso queira hostear o seu ArgoCD, aplique o comando:

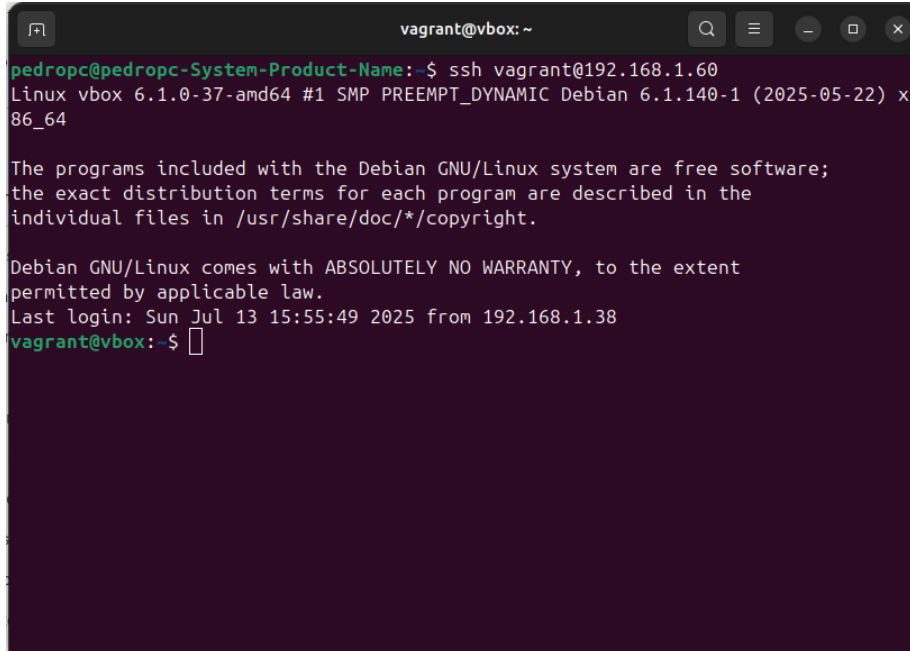
ansible-playbook -i hosts start_argocd.yml

3. Acesse a VM

Abra um terminal na sua máquina local e digite o seguinte comando:

```
ssh vagrant@ip_da_sua_máquina_provisionada
```

Ele irá permitir a conexão com a máquina provisionada via SSH.

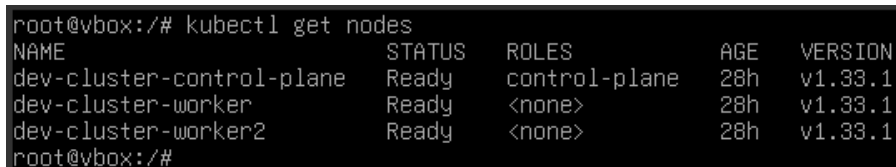
A terminal window titled 'vagrant@vbox: ~' showing an SSH session. The prompt is 'pedropc@pedropc-System-Product-Name:~\$' and the command 'ssh vagrant@192.168.1.60' has been executed. The output shows the SSH banner for Linux vbox 6.1.0-37-amd64 #1 SMP PREEMPT_DYNAMIC Debian 6.1.140-1 (2025-05-22) x86_64, followed by the Debian GNU/Linux system's free software notice and warranty disclaimer. The last login was on Sun Jul 13 15:55:49 2025 from 192.168.1.38. The prompt is now 'vagrant@vbox:~\$' with a cursor.

```
vagrant@vbox: ~  
pedropc@pedropc-System-Product-Name:~$ ssh vagrant@192.168.1.60  
Linux vbox 6.1.0-37-amd64 #1 SMP PREEMPT_DYNAMIC Debian 6.1.140-1 (2025-05-22) x86_64  
  
The programs included with the Debian GNU/Linux system are free software;  
the exact distribution terms for each program are described in the  
individual files in /usr/share/doc/*/copyright.  
  
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent  
permitted by applicable law.  
Last login: Sun Jul 13 15:55:49 2025 from 192.168.1.38  
vagrant@vbox:~$
```

4. Suba o cluster Kubernetes com **kind**

O script provisionado já configura um cluster **kind** automaticamente. Verifique com:

```
kubectl get nodes
```

A terminal window showing the output of the 'kubectl get nodes' command. The output is a table with columns: NAME, STATUS, ROLES, AGE, and VERSION. There are three nodes listed: dev-cluster-control-plane, dev-cluster-worker, and dev-cluster-worker2, all with a status of 'Ready'.

```
root@vbox:/# kubectl get nodes  
NAME                STATUS    ROLES          AGE    VERSION  
dev-cluster-control-plane Ready    control-plane  28h    v1.33.1  
dev-cluster-worker   Ready    <none>         28h    v1.33.1  
dev-cluster-worker2   Ready    <none>         28h    v1.33.1  
root@vbox:/#
```

5. Instale o ArgoCD

Se ainda não estiver rodando, dentro da máquina provisionada aplique:

- kubectl create namespace argocd
- kubectl apply -n argocd -f <https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifests/install.yaml>

(Não quebre linha, faça cada comando na mesma linha.)

Verifique os namespaces criados com:

```
sudo kubectl get namespaces
```

```
root@vbox:/# kubectl get namespaces
NAME                STATUS   AGE
argocd              Active   28h
default             Active   28h
kube-node-lease     Active   28h
kube-public         Active   28h
kube-system         Active   28h
local-path-storage  Active   28h
root@vbox:/# _
```

6. Configure o ArgoCD para os repositórios

Acesse o **ArgoCD** via interface web com **Port Forward**. Na máquina provisionada, aplique o comando:

```
kubectl port-forward svc/argocd-server -n argocd --address 0.0.0.0
8080:443
```

Coloque a porta de acordo com a que foi configurada por você, o comando acima é apenas um exemplo.

Você receberá uma tela de login do ArgoCD:

Usuário -> Admin

Senha -> Para obter a senha, abra o terminal na máquina hospedeira e aplique o comando:

```
kubectl get secret argocd-initial-admin-secret -n argocd -o jsonpath="{.data.password}" |
base64 -d && echo
```

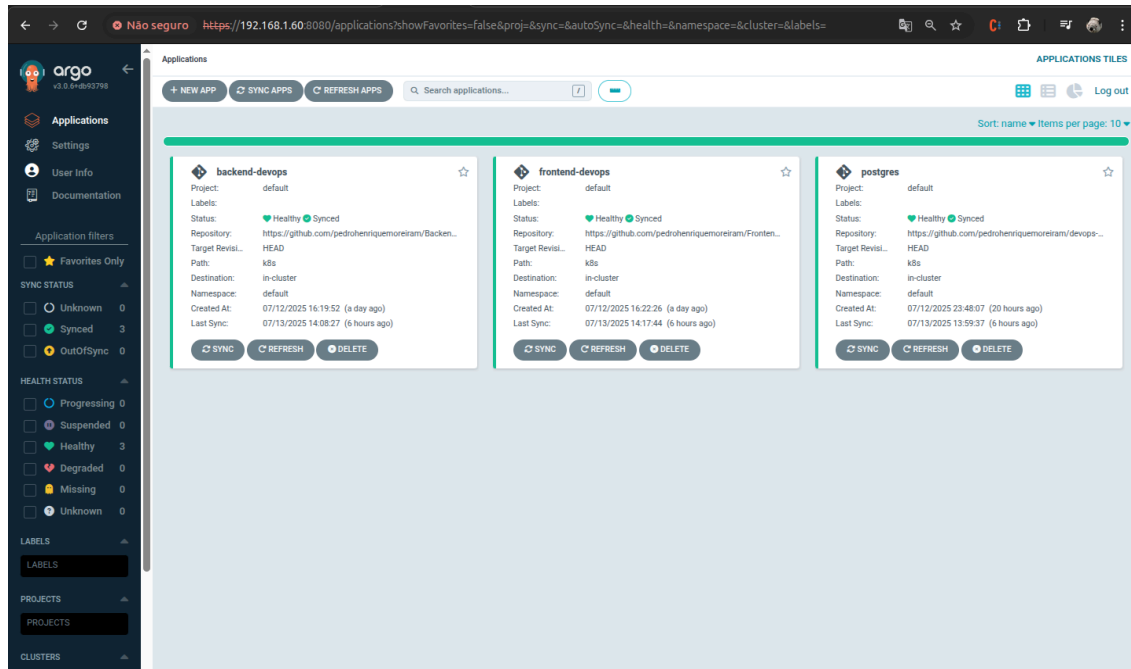
Irá aparecer um código após a execução do comando, essa é a sua senha. Após o login, você irá ver uma tela inicial do **ArgoCD** sem nada adicionado.

A print abaixo mostra um exemplo de tela, mas no seu caso, não terá nada adicionado.

Adicionar apps no ArgoCD:

- Faça os commits dos repositórios clonados para seu repositório no GitHub.
- Vá no seu ArgoCD, clique em **"New app"**.
- Em **"Application Name"**, escolha o nome do seu app (ex:backend-devops)
- Project Name deixe em **default**
- Em **Sync Options** marque **"Prune Last"**, **"Apply out of sync only"** e **"Auto-create namespace"**.
- Em **"SOURCE"** adicione seu repositório do GitHub

Após essas configurações, clique em **“Create”** e pronto, você adicionou seu app ao ArgoCD.



7. Verifique os serviços

Dentro da máquina provisionada, rode o comando abaixo para verificar os serviços:

```
kubectl get svc
```

```
vagrant@vbox:~$ sudo kubectl get svc
NAME                TYPE          CLUSTER-IP      EXTERNAL-IP      PORT(S)          AGE
backend             NodePort      10.96.76.112    <none>           80:30001/TCP     26h
frontend            NodePort      10.96.244.227   <none>           80:30090/TCP     23h
kubernetes           ClusterIP     10.96.0.1       <none>           443/TCP          28h
postgres            NodePort      10.96.109.216   <none>           5432:30003/TCP   20h
vagrant@vbox:~$
```

A print acima mostra serviços que já foram configurados em uma máquina provisionada, na sua pode ser que não tenham todos os serviços rodando ainda.