NixOS: Crie Seu Próprio Laboratório DEV, Atômico e Reprodutível.

Google Developer Groups

2024

DevFest

Presidente Prudente

Objetivo do Curso

O objetivo desse curso é apresentar o **NixOS** e suas vantagens para ambientes de desenvolvimento reprodutíveis e consistentes.

1. Introdução ao Nix e NixOS
2. Instalação e Primeira Configuração
3. Gerenciamento de pacotes com Nix
4. Ambiente de Desenvolvimento Reprodutíveis
5. Introdução ao Nix Flakes
6. Modularização no NixOS

Origem do Nix e NixOS

- **Primeiro Paper sobre Nix(2003)**
  - Escrito por **Eelco Dolstra** durante seu doutorado, o paper introduziu o conceito de um sistema de gerenciamento de pacotes com controle total de dependências.
  - A ideia central era um sistema onde todos os pacotes e suas dependências fossem **imutáveis** e **reprodutíveis**, criando um ambiente de software seguro e estável.

Origem do Nix e NixOS

- **Objetivo Inicial**
  - Resolver problemas clássicos de "**dephell**" (inferno das dependências) e inconsistências em sistemas operacionais, garantindo que as instalações de pacotes fossem idênticas, independentemente de variáveis externas.

Evolução do Projeto e Criação do NixOS

- **2010: Lançamento do NixOS**
  - O NixOS foi lançado como o primeiro sistema operacional baseado no Nix.
  - **Principais Inovações**: Sistema operacional declarativo, totalmente gerenciado pelo Nix, onde o estado do sistema é controlado por arquivos de configuração.

Evolução do Projeto e Criação do NixOS

- **Por Que o NixOS ?**
  - Para estender o conceito de **imutabilidade e reprodutibilidade** do Nix para o sistema inteiro, não apenas para pacotes.
  - Tornou-se ideal para ambientes de desenvolvimento, servidores e aplicações que exigem consistência.

Motivação: O Que Torna o NixOS Diferente?

- **Imutabilidade:** Cada configuração é isolada e previsível, sem efeitos colaterais.
- **Reprodutibilidade:** O sistema pode ser recriado em qualquer ambiente, independente de fatores externos.
- **Gerenciamento Declarativo:** Em vez de alterar o sistema diretamente, os usuários descrevem o estado desejado, e o NixOS aplica essas mudanças de forma controlada e segura.

Diferença entre Nix/NixOS e outros sistemas

- **Instalando pkgs no NixOS:**
  - Como todo tipo de instalação no nixOS é feito de forma declarativa, você sempre terá uma lista que do que tem na sua máquina, te permitindo manter rastreio dos pkgs que você usa.
  - Por conta do nixOS usar o gerenciador de pacotes **nix**, todo pacote instalado vai parar na **nixStore** onde ganha uma chave hashing criptografada que te permite ter várias versões de um mesmo app ou até mesmo ter apps que usam as mesmas dependências mas sem dar conflito. Criando assim pkgs imutáveis e atômicos

Diferença entre Nix/NixOS e outros sistemas - Exemplo

- **Instalando ambiente Gráfico Plasma no Arch Linux:**
  - sudo pacman -S plasma-desktop
  - sudo pacman -S sddm
  - sudo systemctl enable sddm
  - sudo systemctl start sddm

- **Instalando Plasma no NixOS:**

```
services ={
  xserver.enalbe = true;
  display.desktopManager.plasma6.enable = true;
};
```

Diferença entre Nix/NixOS e outros sistemas - Exemplo

- **Removendo Plasma do Arch:**
  - É recomendado que você **saia da sessão gráfica atual** e execute o processo em um terminal TTY para evitar problemas.
  - Use **CTRL + ALT + F2** para abrir um TTY e faça login
  - Pare o serviço de gerenciamento de login
    - sudo systemctl stop sddm
    - sudo systemctl disable sddm
  - Desinstale o pacote do KDE Plasma
    - sudo pacman -Rns plasma-desktop kde-applications
  - Remova arquivos de configuração restantes
    - rm -rf ~/.config/plasma
    - rm -rf ~/.config/kde
    - rm -rf ~/.kde

Diferença entre Nix/NixOS e outros sistemas - Exemplo

- **Removendo Plasma do NixOS:**
  - Mude de **plasma6** para o nome do gerenciador que você deseja:

```
services ={
  xserver.enalbe = true;
  display.desktopManager.plasma6.enable = true;
};
```

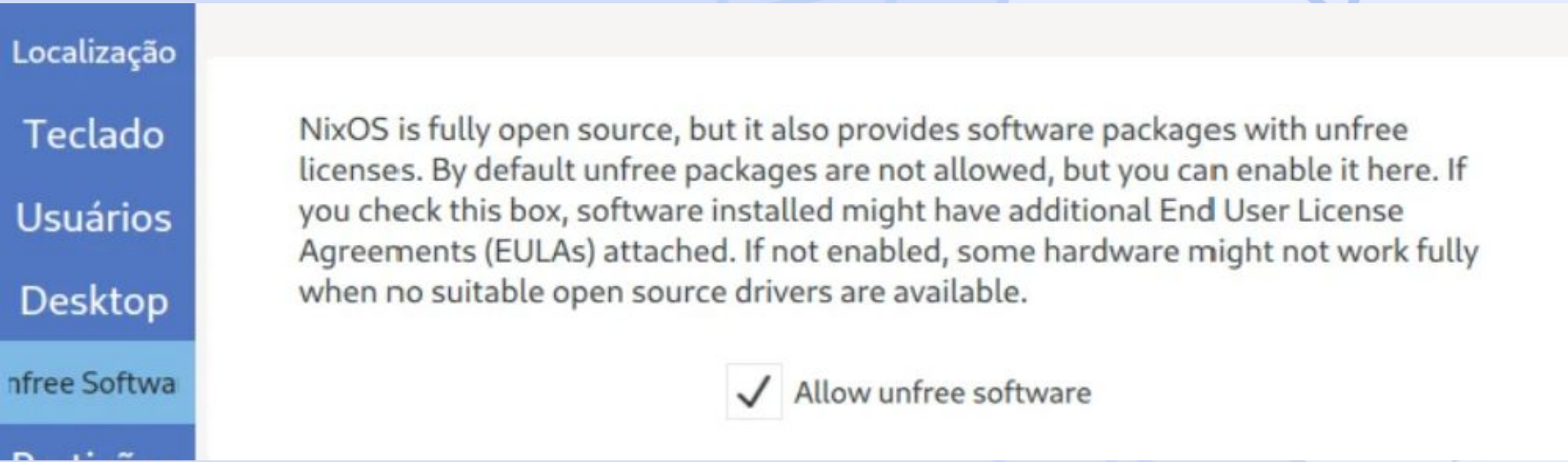
```
services ={
  xserver.enalbe = true;
  display.desktopManager.gnome.enable = true;
};
```

Instalação e Primeira Configuração

- **Instalando o NixOS:**
  - Pré-requisitos, máquina virtual ou bare metal e muita força de vontade.
  - A forma mais fácil instalar o **NixOS** é indo em nixos.org/download/ e baixando a iso **GNOME, 64-bit Intel/AMD**
    - Essa iso é a que tem o passo a passo mais fácil de se seguir, e no final ela ainda te permite escolher se quer manter o ambiente gráfico gnome, colocar outro ambiente gŕafico ou até mesmo deixar sem nenhum.
  - Utilizando o a iso nixOS-GNOME tem duas telas na instalação que é importante destacar.

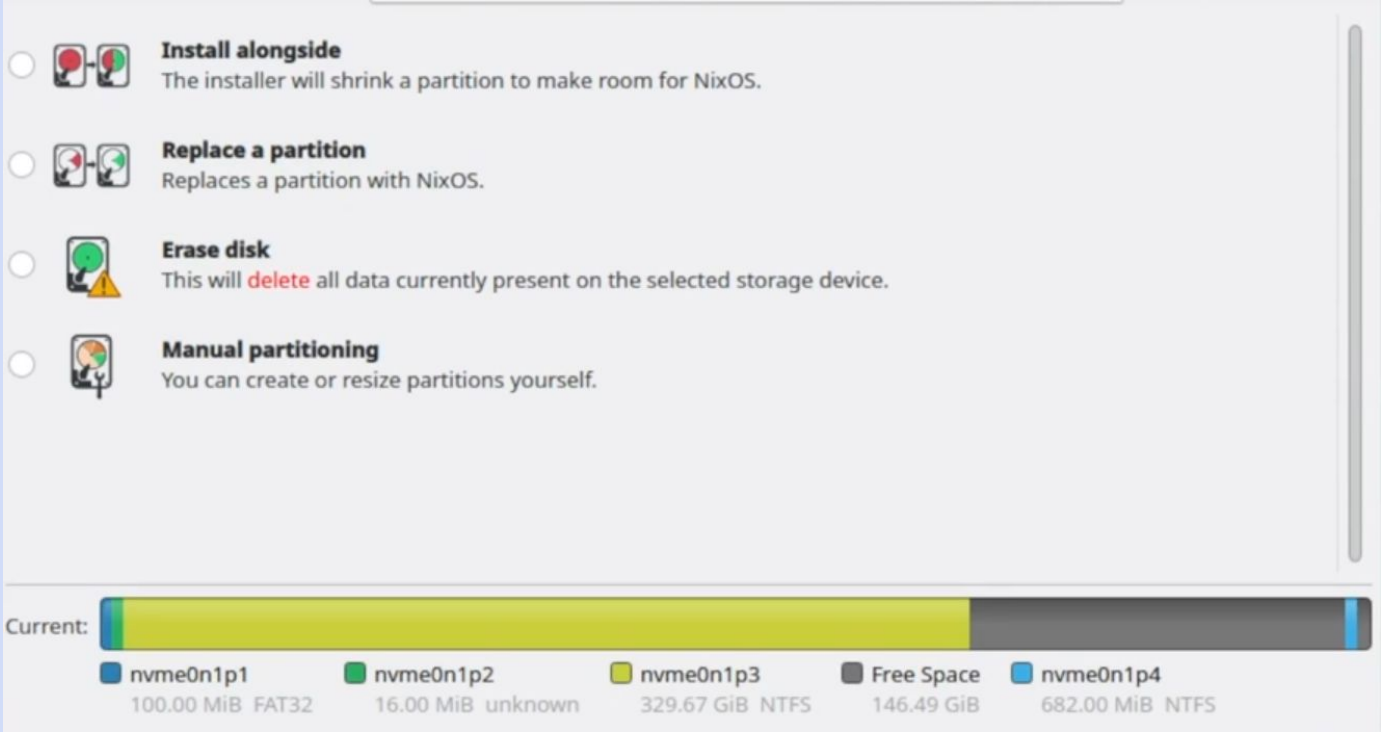
Instalação e Primeira Configuração

- **Instalando o NixOS - Tela Allowunfree:**
  - A opção allow unfrre te permite baixar softwares proprietários como os driver NVDIA que são disponíveis pela própria NVIDIA, se você deixar essa opção desmarcada, você só podera utilizar sofwares Open Source.



Instalação e Primeira Configuração

- **Instalando o NixOS - Tela de Particionamento:**



Instalação e Primeira Configuração

- **Instalando o NixOS - Tela de Particionamento:**
  - **Install Alongside:** Permite instalar o NixOS ao lado de outro OS existente, mantendo ambos. Essa opção é útil para quem deseja experimentar o nixOS sem remover o sistema atual.
  - **Replace a Partition:** Escolhendo essa opção, o usuário pode selecionar uma partição específica para substituir com a instalação do NixOS. Isso resulta na exclusão dos dados existentes na partição selecionada.
  - **Erase Disk:** Esta opção apaga todas as partições no disco selecionado, permitindo uma instalação limpa do NixOS. É ideal para quem deseja iniciar do zero e garantir que não haja conflitos com OS's anteriores.
  - **Manual Partitioning:** Oferece ao usuário a flexibilidade de criar e gerenciar partições de forma personalizada. É recomendado para usuários avançados que desejam controlar a configuração do disco de acordo com suas necessidades específicas.

Primeira Configuração

- **O que é o configuration.nix?**
  - O **configuration.nix** é o arquivo central do nixOS, onde são definidas todas as configurações do sistema. Ele permite a declaração de usuários, pacotes e serviços de maneira declarativa, facilitando a reprodutibilidade e a gestão de configuração. Qualquer alteração nesse arquivo pode ser aplicada de forma simples com o comando **nixos-rebuild switch**.
  - É nele que DECLARAMOS usuários, pacotes de usuário e de sistema, configuramos internet, wifi, HD's e outros.

Gerenciamento de pacotes com Nix - Conceito de Store

- **Como o Nix Armazena Pacotes**
  - O Nix utiliza a **Nix Store** (/nix/store) para armazenar todos os pacotes e dependências do sistema.
  - Cada pacote é ar## Conceito de "Store" e Pacotes no Nix - Como o Nix armazena pacotes - Reprodutibilidade e isolamentomazenado em uma pasta com hash exclusivo **(/nix/store/<hash>-nome-do-pacote)**, o que permite várias versões e configurações do mesmo pacote coexistirem sem conflitos.
  - Esse modelo facilita o controle de versões e remoção segura de pacotes sem comprometer o sistema.

- **Reprodutibilidade e Isolamento**
  - **Reprodutibilidade:** Com a Nix Store, é possível recriar um ambiente de forma idêntica em qualquer máquina que use Nix. Isso garante que as dependências e versões dos pacotes sejam idênticas, evitando problemas de inconsistência.
  - **Isolamento:** Como cada pacote é isolado em seu próprio diretório na Nix Store, não há interferência entre diferentes versões ou configurações de pacotes, eliminando conflitos de dependências.

Gerenciamento de pacotes com Nix - Conceito de Store

```
[menezess42@nixos:~]$ ls -d /nix/store/*python* | head -n 10
/nix/store/00545nw0qdwq2df3nyci3qxf5j5x5rpz-python3.11-cffi-1.16.0.drv
/nix/store/01arsai3zfndxr8x3a832n2rigr68mgw-python3.11-pyelftools-0.31.drv
/nix/store/01z83bfmgzbjg61pblvdcwkn8db3k13d-python3.11-gunicorn-21.2.0.drv
/nix/store/05rdppnd36a1xqlq8g2198r8sxn8b1dq-python3.11-mako-1.3.3.drv
/nix/store/06335lzwrkcx635c4pf98ac6bxqiwadj-python3-3.11.9-env
/nix/store/07q2a5255ndjawg2wnm9k53hjjabr6s0-python3.11-immutables-0.20.drv
/nix/store/0ars0imknddfjkqj0p0nfnfy9b2x510c-python3.11-cattrs-23.2.3.drv
/nix/store/0b0a0nv5mrgsc10ynpvffy9rgdmszvlh-python3.11-platformdirs-4.2.0.drv
/nix/store/0cn4nkg2fcbv5h5k8yg59qwnlqgyccmj-python3.11-markupsafe-2.1.5.drv
/nix/store/0fkvmcsi8lwqagvcjd4v700wdh9cy23m-python3.11-constantly-23.10.4.drv
```

Ambiente de Desenvolvimento Reprodutíveis

- **Configurando um dev env com nix - introdução a default.nix:**
  - Nix oferece o default.nix para configurar ambientes de desenvolvimento personalizados. Esses arquivos permitem definir dependências e configurar o ambiente necessário, garantindo que toda uma equipe tenha acesso às mesmas ferramentas.

```
{ pkgs ? import <nixpkgs> {} }:

pkgs.mkShell {
  name = "python-data-science-env";

  # Lista de pacotes Python para Data Science
  buildInputs = with pkgs; [
python311Packages.pandas
python311Packages.numpy
python311Packages.matplotlib
python311Packages.scipy
python311Packages.scikit-learn
python311Packages.jupyterlab  # para notebooks interativos
  ];

  # Opcional: variáveis de ambiente adicionais
  shellHook = ''
    echo "Bem-vindo ao ambiente Python para Data Science!"
  '';
}
```

Ambiente de Desenvolvimento Reprodutíveis

- **Vantagens:**
  - **Reprodutibilidade Básica:** Todos terão o mesmo ambiente ao usar o mesmo default.nix
  - **Facilidade de Uso:** um único comando (nix-hsell) para configurar o ambiente.
- **Desvantagens:**
  - **Versões não travadas de dependências:** Eles podem não fixar a versão exata do nixpkgs, o que pode levar a inconsistências entre desenvolvedores ao longo do tempo.

```
{ pkgs ? import <nixpkgs> {} }:

pkgs.mkShell {
  name = "python-data-science-env";

  # Lista de pacotes Python para Data Science
  buildInputs = with pkgs; [
python311Packages.pandas
python311Packages.numpy
python311Packages.matplotlib
python311Packages.scipy
python311Packages.scikit-learn
python311Packages.jupyterlab  # para notebooks interativos
  ];

  # Opcional: variáveis de ambiente adicionais
  shellHook = ''
    echo "Bem-vindo ao ambiente Python para Data Science!"
  '';
}
```

Nix Flakes - Resolvendo o problema de modularização e Controle de versão

- **O que São Nix Flakes ?**
  - Nix Flakes são uma forma de organizar e compartilhar configurações e dependências de um projeto no Nix, trazendo mais estrutura e padronização ao ecossistema.
- **Benefícios:**
  - Reprodutibilidade Melhorada: flakes permitem versionar e congelar dependências, tornando os ambientes mais previsíveis.
  - Facilidade de Compartilhamento: Com a estrutura flakes, é fácil compartilhar a configuração de um projeto com outras pessoas, que podem reproduzir o mesmo ambiente localmente.
  - Gestão de Dependências Simplificadas: Ao definir todas as dependências no flake, você garante que elas estarão disponíveis em qualquer máquina onde o projeto for rodado, com a mesma versão.

Nix Flakes - Resolvendo o problema de modularização e Controle de versão

flake.nix

```
{
        description = "Day Planner and Investment Tracker";

        inputs = {
                nixpkgs.url = "github:NixOS/nixpkgs/nixpkgs-unstable";
                flake-utils.url = "github:numtide/flake-utils";
        };

        outputs = { self, nixpkgs, flake-utils }:
                flake-utils.lib.eachDefaultSystem (system:
                                let
                                pkgs = import nixpkgs { inherit system; };
#enableIDE = true;
#idePackages = if enableIDE then import ./ide-packages.nix {inherit pkgs; } else [];
                                in {
                                devShell = pkgs.mkShell {
                                name = "day-planner-env";
                                buildInputs = with pkgs; [
                                curl
                                direnv
                                sqlite
                                nodejs
                                python311
# Project Lib
                                python311Packages.pip
                                python311Packages.pandas
                                python311Packages.openpyxl
                                python311Packages.matplotlib
                                python311Packages.flask
                                python311Packages.sqlalchemy
                                python311Packages.flask-sqlalchemy
                                python311Packages.flask-migrate
                                python311Packages.flask-login
                                python311Packages.flask-bcrypt
                                python311Packages.pytest
                                python311Packages.pytest-flask
                                ];
                                shellHook = ''
                                        echo "Welcome to the Day Planner and Investment Tracker environment!"
                                        '';
                                };
                                }
                                );
}
```

Nix Flakes - Resolvendo o problema de modularização e Controle de versão

flake.lock

Nix Flakes - Explicação da anatomia

1. **description**:
   a. Uma breve descrição do flake. Facilita a identificação do propósito
2. **inputs:**
   a. Define as dependências externas, como nixpkgs, necessário para acessar pacotes Nix.

```nix
{
  description = "Meu Flake de Exemplo";

  # Inputs: Definem as dependências do flake (outros flakes ou canais Nix).
  inputs = {
    nixpkgs.url = "github:NixOS/nixpkgs"; # Referência ao repositório Nixpkgs
  };

  # Outputs: Define o que o flake "produz" — como pacotes, configurações e módulos.
  outputs = { self, nixpkgs, ... }:
    let
      pkgs = import nixpkgs { system = "x86_64-linux"; };
    in
    {
      # Packages: Pacotes que o flake fornece.
      packages.default = pkgs.hello;

      # Apps: Define aplicativos executáveis que podem ser expostos pelo flake.
      apps.hello = {
        type = "app";
        program = "${pkgs.hello}/bin/hello";
      };

      # DevShell: Configurações para um ambiente de desenvolvimento.
      devShells.default = pkgs.mkShell {
        buildInputs = [ pkgs.hello ];
      };
    };
}
```

Nix Flakes - Explicação da anatomia

3. **Outputs:**
   a. Onde estão definidos os elementos principais do flake, incluindo pacotes(**pakgs**), aplicativos(**apps**), e ambiente de desenvolvimento (**devShells**). É a seção principal, onde os módulos importados e a lógica são especificados.

```
{
  description = "Meu Flake de Exemplo";

  # Inputs: Definem as dependências do flake (outros flakes ou canais Nix).
  inputs = {
    nixpkgs.url = "github:NixOS/nixpkgs"; # Referência ao repositório Nixpkgs
  };

  # Outputs: Define o que o flake "produz" — como pacotes, configurações e módulos.
  outputs = { self, nixpkgs, ... }:
    let
      pkgs = import nixpkgs { system = "x86_64-linux"; };
    in
    {
      # Packages: Pacotes que o flake fornece.
      packages.default = pkgs.hello;

      # Apps: Define aplicativos executáveis que podem ser expostos pelo flake.
      apps.hello = {
        type = "app";
        program = "${pkgs.hello}/bin/hello";
      };

      # DevShell: Configurações para um ambiente de desenvolvimento.
      devShells.default = pkgs.mkShell {
        buildInputs = [ pkgs.hello ];
      };
    };
}
```

Nix Flakes - usando **direnv** para ativar Flakes atuomaticamente

- **O que é direnv ?**
  - direnv é uma ferramenta que carrega/descarrega variáveis de ambiente automaticamente ao entrar e sair de diretórios.
  - Ideal para automatizar o ambiente de desenvolvimento sem precisar ativar manualmente os flakes toda vez.
- **Como instalar ?**
  - Adicione o direnv aos pacotes do seu nixOS no configuration.nix
  - Execute o rebuild como vimos antes.
  - habilite o Hook do direnv no shell. Vá no seu .bashrc e adicione a seguinte linha **eval “$(direnv hook bash)”**.
  - Crie um arquivo .envrc na pasta do flake e adicione o seguinte comando a esse arquivo **use flake**
  - Para finalizar execute um **direnv allow**.

Nix Flakes - Rodando código Java.

- **Para compilar:**
  - Para compilar execute o comando abaixo:
    - java hello.java
- **Para executar:**
  - Para rodar execute o comando abaixo:
    - java hello

Nix Flakes - Rodando código C/C++.

- **Para compilar:**
  - Para compilar execute o comando abaixo:
    - gcc hello.c -o hello
- **Para executar:**
  - Para rodar execute o comando abaixo:
    - ./hello

Modularizando o configuration.nix

- **Por que Modularizar?**:
  - Modularizar o `configuration.nix` facilita a organização e manutenção de configurações complexas. Permite dividir o arquivo principal em partes menores e mais específicas, tornando mais fácil localizar e atualizar configurações específicas.
- **Como Funciona**:
  - No `configuration.nix`, podemos importar arquivos de configuração separados, cada um contendo uma configuração ou serviço específico. Por exemplo, você pode ter um módulo só para configurar redes, outro para usuários, e outro para serviços.

Explicando Comandos da Aula de ontem

- **Nix-shell Execução Instantânea:**
  - O nix-shell permite executar aplicativos temporariamente, sem a necessidade de instalação. É ideal para testes rápidos ou uso pontual de uma ferramenta, sem deixar rastros no sistema.
- **Como Funciona para testes de aplicações**:
  - Para iniciar um aplicativo de forma temporária, basta usar:
    - **nix-shell -p nomeDoApp**
- **Como funciona para rodar devShells:**
  - Estando dentro da pasta de desenvolvimento(pasta onde se localiza o default.nix), rode nix-shell.

Explicando Comandos da Aula de ontem

- **sudo nvim /etc/nixos/configuration.nix:**
  - Aqui, abrimos o arquivo de configuração principal do NixOS usando permissões de Superusuário(**sudo**).
- **sudo nixos-rebuild switch:**
  - Após adicionarmos novos pacotes a configuração, esse comando aplica as alterações no sistema. O switch reconstrói e ativa a nova configuração, permitindo verificar imediatamente os resultados sem precisar reiniciar.
- **nix flake init:**
  - Esse comando inicializa um novo flake no diretório atual, gerando uma configuração básica do Nix flakes.
- **nix flake lock:**
  - Após inicializar o flake, o **nix flake lock** fixa versões de dependências, garantindo que o projeto tenha versões específicas e consistentes.

Ariel Menezes

- **GitHub:**
  - github.com/Menezess42
- **Linkedin:**
  - www.linkedin.com/in/menezess42
- **Instagram:**
  - @ariel_meenzess42
- **Server do Discord:**
  - https://discord.gg/eFaUw6D8