KHARAGPUR OPEN SOURCE SOCIETY

UNDERSTANDING NIXOS: THE DECLARATIVE LINUX DISTRO

Presented by –
Saurav Raj
24CS10133

What is NixOS?

NixOS is a Linux distribution that uses the Nix package manager to manage packages and configurations declaratively.

KEY FEATURES

DECLARATIVE CONFIGURATION

DEFINE EVERYTHING IN /ETC/NIXOS/CONFIGURATION.NIX, MAKING SYSTEM SETUPS FULLY REPRODUCIBLE.

```
{ config, pkgs, ... }: {
  # Install packages
  environment.systemPackages = with pkgs; [ vim firefox git ];

  # Enable a service
  services.openssh.enable = true;

  # Define a user
  users.users.myuser.isNormalUser = true;
}
```

ATOMIC UPGRADES AND ROLLBACKS

UPDATES WON'T BREAK THE SYSTEM. IF SOMETHING GOES WRONG, JUST ROLLBACK INSTANTLY.

```
# Upgrade system
nixos-rebuild switch --upgrade

# Rollback if something breaks
nixos-rebuild switch --rollback
```

REPRODUCIBLE SYSTEM CONFIGURATIONS

WHETHER YOU REINSTALL, SWITCH MACHINES, OR DEPLOY ON A SERVER, YOUR SYSTEM STAYS IDENTICAL.

```
# Clone your system config from GitHub
git clone https://github.com/user/my-nixos.git
cd my-nixos

# Rebuild system exactly as defined
nixos-rebuild switch --flake .#hostname
```

NO DEPENDENCY HELL

THANKS TO NIX'S PACKAGE MANAGER, MULTIPLE VERSIONS OF SOFTWARE CAN COEXIST WITHOUT CONFLICTS.

```
# Run Python 3.10 in an isolated shell
nix-shell -p python310 --run "python3 --version"

# Now switch to Python 3.9 without conflicts
nix-shell -p python39 --run "python3 --version"
```

HOW IS NIXOS ANY BETTER THAN OTHER OPTIONS...

	NIXOS	OTHER DISTROS
DECLARATIVE CONFIGURATION	✓	✗ (MANUAL PACKAGE MANAGEMENT)
ATOMIC UPGRADES & ROLLBACKS	✓	✗ (PRONE TO BREAKING AFTER UPDATES)
NO DEPENDENCY CONFLICTS	✓	✗ (DLL/PACKAGE VERSION HELL)
REPRODUCIBILITY	✓	✗ (VARIES BETWEEN INSTALLATIONS)
ISOLATED ENVIRONMENTS	✓	✗ (SHARED DEPENDENCIES)

HOW TO CREATE YOUR OWN PACKAGE IN NIX

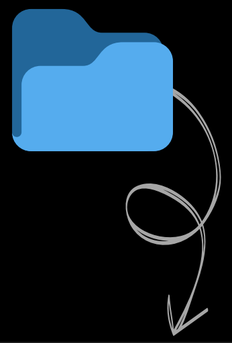
WHAT IS A NIX PACKAGE ?

A Nix package is software defined and built using Nix's declarative approach, ensuring it’s reproducible, isolated, and conflict-free.

Why care about creating your own package ??

- **Package software your way**: Whether it's a personal project, a specific version of a tool, or a patched software, Nix lets you customize how things are built and installed.
- **Eliminate “works on my machine” issues**: Nix guarantees that if it works on your system, it will work exactly the same elsewhere.
- **Keep your system clean**: Packages live in their own isolated environments, avoiding conflicts with system-wide dependencies.
- **Easily roll back changes**: If something breaks, you can quickly revert to a previous working version.

1 CREATE A NIX EXPRESSION FILE



```
{ stdenv, fetchurl }:
stdenv.mkDerivation {
  name = "mypackage";
  src = fetchurl {
    url = "https://example.com/mypackage.tar.gz";
    sha256 = "0b2...";
  };
  buildInputs = [];
  installPhase = "cp -r * $out";
}
```

Define your package using mkDerivation, which sets up the build process.

2 BUILD THE PACKAGE

```
nix-build mypackage.nix
```

This command fetches sources, builds in isolation, and outputs a derivation (.drv)

3 CHECK THE OUTPUT

```
ls result/
```

The built package is available inside the result/ directory.

4 USE THE PACKAGE

```
./result/bin/mypackage
```

Run the package directly from the Nix store!

NIX LANGUAGE

NIX IS A PURELY FUNCTIONAL, DECLARATIVE LANGUAGE USED TO DEFINE AND MANAGE PACKAGE BUILDS AND SYSTEM CONFIGURATIONS

- DECLARATIVE – DEFINE WHAT YOU WANT, NOT HOW.
- FUNCTIONAL – NO SIDE EFFECTS, JUST PURE EXPRESSIONS.
- REPRODUCIBLE – ENSURES THE SAME RESULT EVERYWHERE

```
let
  a = 10;
  b = 5;
in
{ sum = a + b; }
```

HOW DOES NIX WORK

Step 1
Parse & Evaluate

Step 2
Fetch dependencies

Step 3
Buid and store in
/nix/store/

```
{ pkgs }: {
  hello = pkgs.hello;
}
```

This simple code piece tells Nix to fetch and use hello package declaratively !!

NIX FLAKES

NIX FLAKES INTRODUCE A MORE RELIABLE, STRUCTURED WAY TO MANAGE NIX PROJECTS WITH BETTER DEPENDENCY HANDLING AND REPRODUCIBILITY.

WHY CARE ABOUT FLAKES...

- Better reproducibility (locks dependencies)
- Simpler Package Management (self-contained flake.nix)
- Easier Sharing & Git Integration (pin dependencies with flake.lock)

```
nix flake init
nix flake update
nix run .#mypackage
```

FIRST, INITIALIZE A NEW FLAKE. THEN UPDATE AND RUN YOUR PACKAGE!

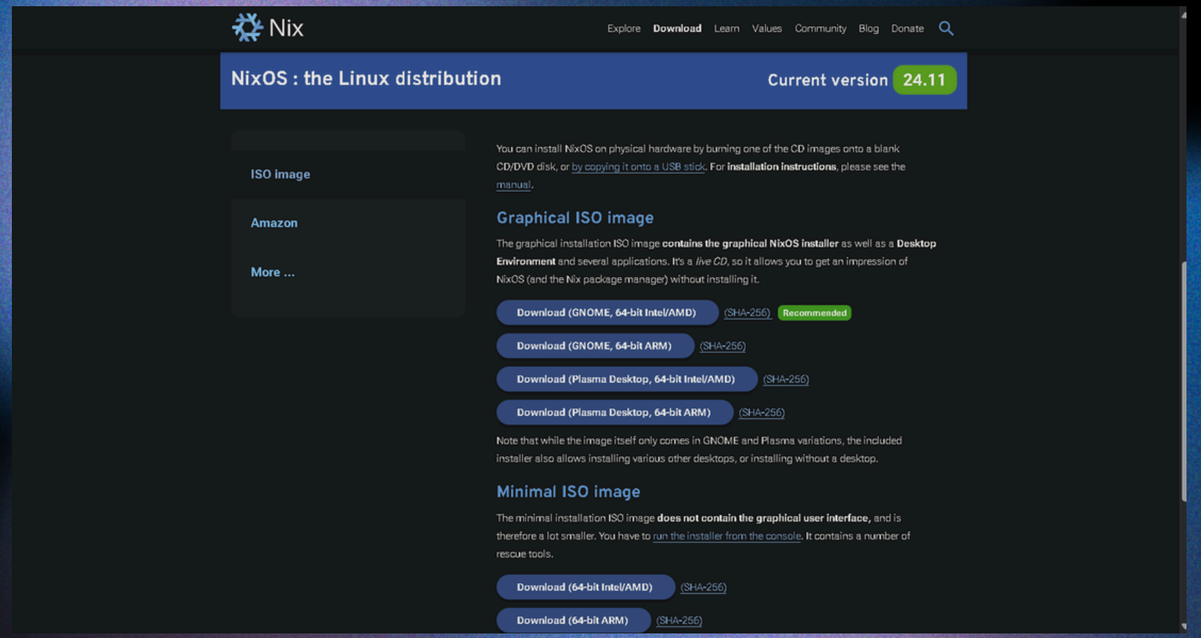
	TRADITIONAL NIX	NIX FLAKES
LOCK DEPENDENCIES	❌ NO	✅ YES (FLAKE.LOCK)
GIT INTEGRATION	❌ MANUAL	✅ BUILT-IN
DEFAULT PACKAGES	🔄 IMPLICIT	✅ EXPLICIT & STRUCTURED
REPRODUCIBILITY	⚠️ CAN DRIFT	✅ FULLY LOCKED

LET'S INSTALL NIXOS!!

1 GO TO NIXOS.ORG AND GRAB THE LATEST ISO.

GRAPHICAL (GNOME/KDE) – IF YOU WANT A USER-FRIENDLY EXPERIENCE.

- **MINIMAL ISO – FOR TERMINAL NINJAS.**



2 MAKE A BOOTABLE USB

WINDOWS USER ??
Go with Rufus(super easy)

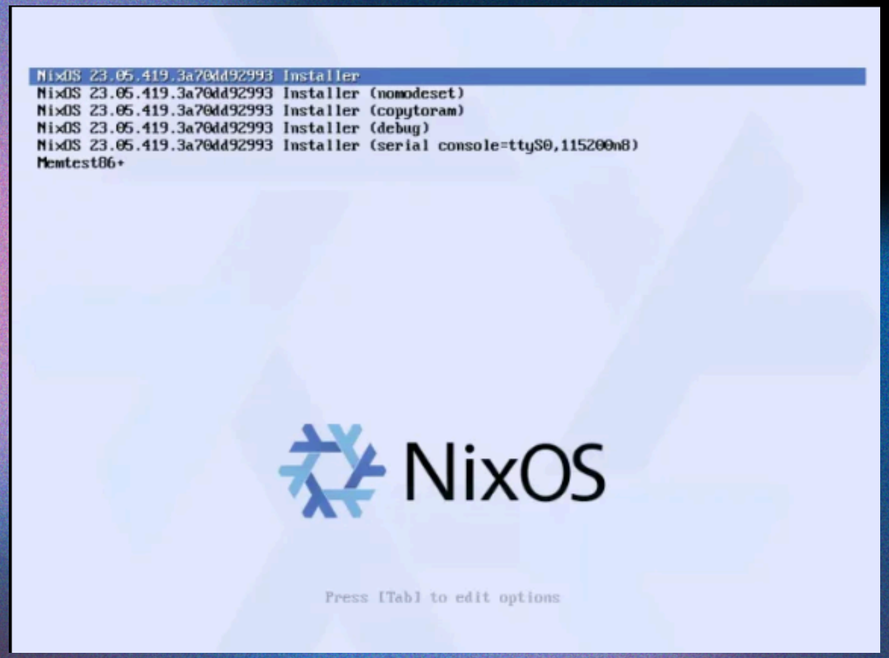
LINUX/MAC??
Run this in your terminal

```
sudo dd if=nixos.iso of=/dev/sdX bs=4M status=progress
```

3 BOOT INTO NIXOS LIVE MODE

Restart your computer and boot from the USB (usually by pressing F12 or ESC).

- You'll see the NixOS menu.
  Select "Boot NixOS" and wait.



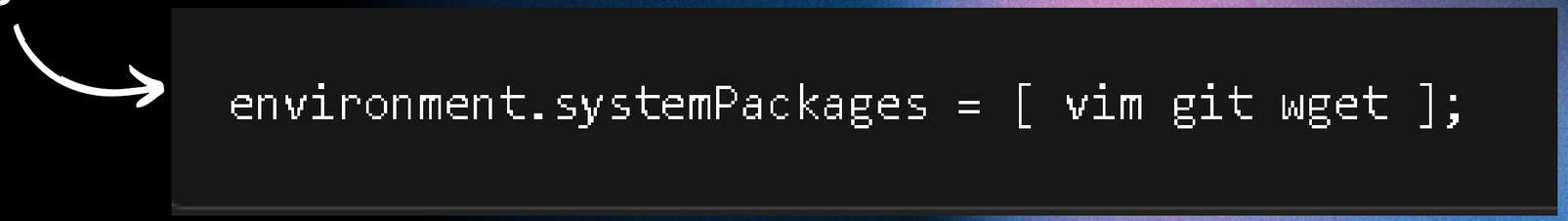
4 If you're installing fresh,you might need to set up partitions... (use this)

```
nixos-generate-config --root /mnt
```

5 CONFIGURE YOUR SYSTEM

OPEN /mnt/etc/nixos/configuration.nix.

(This is where you tell NixOS what to install)
for eg.



```
environment.systemPackages = [ vim git wget ];
```

6 RUN THE MAGIC COMMAND

```
nixos-install
```

Welcome to the OS where nothing makes sense!!

HERE'S YOUR SURVIVAL KIT...

`NIX-SHELL -P <PKG>`	Run a package temporarily without installing it.
`NIXOS-REBUILD SWITCH`	Apply config changes from /etc/nixos/configuration.nix.
`NIX-ENV -IA NIXPKGS.<PKG>`	Install a package for the current user.
`NIX-COLLECT-GARBAGE -D`	Free up disk space by removing old packages.
`NIX SEARCH NIXPKGS <PKG>`	Find available packages.

Thank You!