Demystifying the Nix Store

The Giant Immutable LEGO® Set

Made with Marp by Leonard Sheng Sheng Lee.

Have You Ever Looked Inside `/nix/store` ?

```
├── 0a5jirwm929lxp126lhsaivlcjzj56dl-nixd-2.8.2
├── k7gnyhjpdngfjv77hlh28zhrpv7gkar9-glibc-2.42-47.drv
├── 7m2c9rgvk9hnd8ngvp4c1hhl43c97jxs-gcc-15.2.0
```

The Immediate Reaction

What are these hashes?

Why is everything immutable?

Where's my `/usr/bin` ?

How does anything even work?

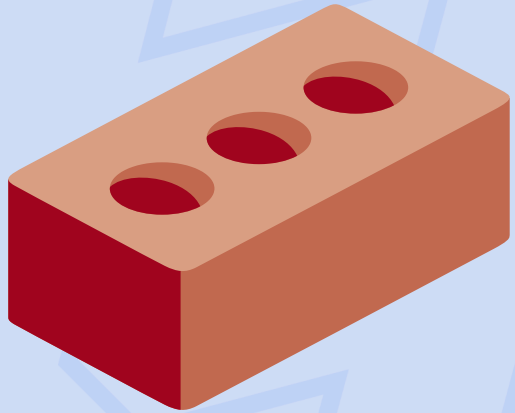
Today's Session

Explore the LEGO® Metaphor.

Look Inside Expressions & Derivations.

Put Nix into Practice.

Explore the LEGO® Metaphor



Made with Marp by Leonard Sheng Sheng Lee.

Conventional Package Managers

Everything mixed together.

Upgrading might break.

Hard to undo changes.

Dependency hell.

Giant LEGO® Set

For each Nix package:

The binaries and libraries are like a set of bricks.

The derivation that builds it is like a instruction manual.

The content-addressed hash is like a unique set number.

LEGO® Metaphor ≈ Nix Rules

Bricks snap together perfectly.

Instructions are reproducible.

Sets never change once built.

You can combine sets safely.

You can't modify a built set.

Look Inside Expressions & Derivations

Made with Marp by Leonard Sheng Sheng Lee.

What Is a Nix Expression? (1/2)

The source code contained in a `.nix` file.

A **pure, functional language** for describing packages.

Every `.nix` file is a function that returns a value.

Expressions declare **what** to build, not **how** to run it.

What Is a Nix Expression? (2/2)

> **The LEGO® design sketch you draw on paper.**

Expressions -> Derivations (1/2)

Nix *evaluates* **the expression** to produce a **derivation**.

The derivation is a fully resolved build recipe with every dependency pinned to an exact store path.

Expressions -> Derivations (2/2)

> **Read the sketch, prints the instruction manual.**

What Is a Derivation? (1/2)

A **recipe** that describes:

- **Inputs**: What you need, such as dependencies.
- **Build steps**: How to build it.
- **Outputs**: What you get, including binaries and libraries.

What Is a Derivation? (2/2)

> **The LEGO® instruction booklet inside your set.**

Example of Nix Expression

```
# show-utc-datetime.nix

{  pkgs ? import <nixpkgs> { }, }:
# ----------------------------------------
pkgs.runCommand "show-utc-datetime"
  { nativeBuildInputs = [ pkgs.uutils-coreutils-noprefix ]; }
# ----------------------------------------
  ''
    mkdir --parents $out/bin
    cat > $out/bin/show-utc-datetime <<'EOF'
    #!${pkgs.runtimeShell}
    exec ${pkgs.uutils-coreutils-noprefix}/bin/date --universal +"%Y%m%dT%H%M%SZ"
    EOF
    chmod +x $out/bin/show-utc-datetime
  ''
```

Declaring Inputs (1/2)

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

`pkgs` is the function argument.

`import ...` loads the entire Nix packages collection.

Declaring Inputs (2/2)

> **Find the bricks you need in a LEGO® catalog.**

Choosing the Builder

```
pkgs.runCommand "show-utc-datetime"
  { nativeBuildInputs = [ pkgs.uutils-coreutils-noprefix ]; }
```

The `runCommand` creates a simple derivation without needing a full `stdenv.mkDerivation` function.

The Build Script

```
  ''
    mkdir --parents $out/bin
    cat > $out/bin/show-utc-datetime <<'EOF'
    #!${pkgs.runtimeShell}
    exec ${pkgs.uutils-coreutils-noprefix}/bin/date --universal +"%Y%m%dT%H%M%SZ"
    EOF
    chmod +x $out/bin/show-utc-datetime
  ''
```

The shell script that calls `date` from the package.

String Interpolation

```
#!${pkgs.runtimeShell}
exec ${pkgs.uutils-coreutils-noprefix}/bin/date \
--universal +"%Y%m%dT%H%M%SZ"
```

After Nix evaluates above expression, it becomes:

```
#!/nix/store/...8s9kxnp-bash-5.3p9/bin/bash
exec /nix/store/...093gp61-uutils-coreutils-0.6.0/bin/date \
  --universal +"%Y%m%dT%H%M%SZ"
```

Every dependency is pinned to an **exact store path**.

Build The Expression (*.nix) File

```
nix build --file show-utc-datetime.nix ...
```

Evaluate the expression to produce a `.drv` file.

Compute a hash from inputs.

Run the build script in an isolated sandbox.

Store the output at the computed path.

Derivation: Build Output

```
this derivation will be built:
  /nix/store/...gn1360z-show-utc-datetime.drv
```

Evaluate the `.nix` expression to produce a `.drv` file.

Compute the hash `...gn1360z` from all inputs.

Execute the `.drv` file as the actual build recipe.

Derivation: Build Output / Get Dependencies

```
this path will be fetched (3.22 MiB download, 12.22 MiB unpacked):
  /nix/store/...093gp61-uutils-coreutils-0.6.0

copying path '...-uutils-coreutils-0.6.0' from 'https://cache.nixos.org'
```

The `uutils-coreutils-0.6.0` package is needed but not in the local store.

Derivation: Build Output / Executing Build

```
show-utc-datetime> building
  '/nix/store/...gn1360z-show-utc-datetime.drv'
```

Run the build script inside a sandbox.

Allow only declared dependencies.

Block network access and access to `/usr` or `/bin`.

Place the output at the computed store path.

Derivation: Build Output / Dependency Graph

```
┌─ Dependency Graph:
│  ✓ show-utc-datetime
├─ Builds
└─ ∑ ▸ 0 | ✓ 1 | ⏸ 0 | Finished after 3s
```

Optional: `nix-output-monitor`

The Derivation (*.drv) File

```
nix derivation show --file show-utc-datetime.nix
```


```
{
    "/nix/store/...gn1360z-show-utc-datetime.drv": {
        "outputs": {
            "out": {
                "path": "/nix/store/...v8h9rcs-show-utc-datetime"
            }
        }
    }
}
```


Map a derivation hash to a deterministic output path.

Inside the `.drv` : Builder and Arguments

```
{
    "builder": "/nix/store/...8s9kxnp-bash-5.3p9/bin/bash",
    "args": [
        "-e",
        "/nix/store/...ic6ynpg-source-stdenv.sh",
        "/nix/store/...ny02r39-default-builder.sh"
    ]
}
```

Every path is absolute.

Nothing comes from the host system.

Inside the `.drv` : Input Derivations

```
{
    "inputDrvs": {
        ".../bash-5.3p9.drv": { "outputs": ["out"] },
        ".../uutils-coreutils-0.6.0.drv": {
            "outputs": ["out"]
        },
        ".../stdenv-darwin.drv": { "outputs": ["out"] }
    }
}
```

Three input derivations must be built first.

Inside the `.drv` : Build Command

```
{
  "env": {
    "buildCommand": "mkdir --parents $out/bin\n
      cat > $out/bin/show-utc-datetime <<'EOF'\n
      #!/nix/store/...8s9kxnp-bash-5.3p9/bin/bash\n
      exec /nix/store/...093gp61-uutils-coreutils-0.6.0/bin/date
        --universal +\"%Y%m%dT%H%M%SZ\"\n
      EOF\nchmod +x $out/bin/show-utc-datetime\n",
    "name": "show-utc-datetime",
    "system": "aarch64-darwin"
  }
}
```

Resolve all interpolations in the build script.

Examining the Output

```
$ tree /nix/store/...v8h9rcs-show-utc-datetime
/nix/store/...v8h9rcs-show-utc-datetime
└── bin
    └── show-utc-datetime

$ cat /nix/store/...v8h9rcs-show-utc-datetime/bin/show-utc-datetime
#!/nix/store/...8s9kxnp-bash-5.3p9/bin/bash
exec /nix/store/...093gp61-uutils-coreutils-0.6.0/bin/date \
  --universal +"%Y%m%dT%H%M%SZ"
```

Running the Result

```
$ /nix/store/...v8h9rcs-show-utc-datetime/bin/show-utc-datetime
20260223T184340Z

$ ./result/bin/show-utc-datetime
20260223T184340Z
```

Create a `./result` symlink to the store path.

Build Process in a Nutshell

1. **Evaluate the expression**, the `.nix` file.
2. **Produce the derivation**, the `.drv` file.
3. **Gather all inputs**, including dependencies.
4. **Create isolated environment.**
5. **Run build steps.**
6. **Store output** at the computed path.

The Store Path Structure (1/2)

```
[hash]-[name]-[version]
└──────────┬──────────┘
     Unique Identifier
```

Example：

```
1xy62wrp3m91snd3cazxgg0yrplb6sav-git-2.52.0
```

The Store Path Structure (2/2)

$$f(\text{inputs}) = /\text{nix}/\text{store}/\ \ldots$$

```
  /nix/store/v4bvnkm0p5x41fhybskr0cf2zvkgyrvv-cargo-1.92.0
  |---------| |------------------------------| |----------|
Store Directory            Digest                  Name
```

Why Hashes Matter

Reproducibility: Same inputs = same hash.

Caching: Already built? Reuse it!

Isolation: Different versions coexist peacefully.

Atomic upgrades: New hash = new path.

Immutable Store

Once built, **never changes**.

Upgrades create **new paths**.

Old versions remain until garbage collected.

Rollbacks are just **switching symlinks**.

Example: Upgrading Python

```
/nix/store/iki3g1iyxydm65k7hm0r3ssm8l6mvlb6-python3-3.12.8
/nix/store/8bwmgvfcyys3kfia055ih7gask3fid7s-python3-3.14.2
```

Both exist simultaneously!

Your programs use whatever version they need.

Atomic Operations

```
$ nix-env --install firefox
...
$ nix-env --rollback  # Instant undo!
```

Update symbolic link at `/nix/var/nix/profiles/default` path.

Total Isolation

During a Nix build:

Block network access in most cases.

Block access to `/usr`, `/bin`, etc.

Prevent environment variable leakage.

Allow only declared dependencies.

Result: Reproducible builds!

Exploring Dependencies

```
$ nix-store --query --references /nix/store/...-hello
/nix/store/...-glibc-2.35
/nix/store/...-gcc-11.3.0-lib

$ nix-store --query --referrers /nix/store/...-glibc
/nix/store/...-hello
/nix/store/...-bash
/nix/store/...-coreutils
```

Closure (1/2)

A package with all its dependencies, resolved recursively.

```
$ nix-store --query --requisites /nix/store/...-hello
/nix/store/...-hello
/nix/store/...-glibc-2.35
/nix/store/...-gcc-11.3.0-lib
/nix/store/...-linux-headers-5.19
```

This is everything needed to run the program.

Closure (2/2)

> The LEGO set that completely built.

Made with Marp by Leonard Sheng Sheng Lee.

Closure Size

```
$ nix path-info --closure-size --human-readable nixpkgs#hello
/nix/store/...-hello-2.12.2  31.8 MiB
```

Why does "Hello World" need 31.8 MiB?

`glibc-2.42-47` : 29.0 MiB

...

`hello-2.12.2` : 268.2 KiB

Garbage Collection

```
nix-collect-garbage --delete-older-than 30d
```

Put Nix into Practice

Made with Marp by Leonard Sheng Sheng Lee.

Flakes: Modern Nix

```
{
  inputs.nixpkgs.url = "github:NixOS/nixpkgs/nixos-25.11";

  outputs = { self, nixpkgs }: {
    packages.x86_64-linux.default = nixpkgs.legacyPackages.x86_64-linux.hello;
  };
}
```

Explicit inputs with locked versions.

Better reproducibility.

Easier to share and compose.

Our Flake: Full Expression

```
{
  description = "Show UTC Date & Time";
  inputs.nixpkgs.url = "github:NixOS/nixpkgs/nixpkgs-unstable";
  outputs = { self, nixpkgs }:
    let
      forAllSystems = f:
        nixpkgs.lib.genAttrs
          [ "x86_64-linux" ... "aarch64-darwin" ]
          (system: f nixpkgs.legacyPackages.${system});
    in
    { packages = forAllSystems (pkgs: {
        default = pkgs.runCommand "show-utc-datetime" { ... } ''...'';
      });
    };
}
```

Flake: Metadata and Inputs (1/2)

```
description = "Show UTC Date & Time";

inputs.nixpkgs.url = "github:NixOS/nixpkgs/nixpkgs-unstable";
```

Declare external dependencies with exact sources.

Create a `flake.lock` to pin the exact revision.

Flake: Metadata and Inputs (2/2)

> **Order from which LEGO® catalog edition.**

Flake: Multi-Platform Support

```
outputs = { self, nixpkgs }:
  let
    forAllSystems = f:
      nixpkgs.lib.genAttrs
        [ "x86_64-linux" "aarch64-linux"
          "x86_64-darwin" "aarch64-darwin" ]
        (system: f nixpkgs.legacyPackages.${system});
  in
```

The `outputs` is a function receiving resolved inputs.

Flake: Package Definition

```
{
  packages = forAllSystems (pkgs: {
    default =
      pkgs.runCommand "show-utc-datetime"
        # ...
  });
};
```

The build logic is identical to our regular expression.

Flake Outputs and Derivations

```
nix build

nix flake show

nix flake metadata

nix run
```

> **Everything is still derivations under the hood!**

Analyzing Flake Dependencies

```
nix why-depends .#default \
  "$(nix path-info --recursive .#default | grep bash)"

# /nix/store/42c3md0x...-show-utc-datetime
# └───/nix/store/vlfjhc97...-bash-5.3p9

nix path-info --recursive .#default

nix path-info --closure-size .#default
```

Explore your dependency graph.

Real-World Impact

Development: Consistent, reproducible environments.

CI/CD: Hermetic, cacheable builds.

Production: Atomic deployments. Declarative configuration. SBOM.

Multi-user: Isolated user environments. No dependency conflicts.

The Trade-Offs

Benefits:

Extreme reliability.

Perfect reproducibility.

Drawbacks:

Disk space from storing many versions.

Steep learning curve.

Key Takeaways

Expressions ≈ Design Sketches

Derivations ≈ Instruction Manuals

Hashes ≈ Set Numbers

Next Steps

Explore your own `/nix/store`.

Read some `.drv` files.

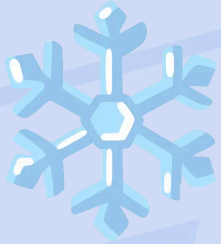
Trace dependencies with `why-depends`.

Write your first Nix expression.

Join the Nix community.

Thanks!

sheeeng.github.io/slides



Made with Marp by Leonard Sheng Sheng Lee.