Fediversity Implementation and planning

Actors

- Maintainers

  The group developing and maintaining this project. We are creating the deployment workflows and service configurations, and curate changes proposed by contributors.

- Developers

  People with the technical background to engage with our work, and may contribute back, build on top of, remix, or feel inspired by our work to create something better.

- Hosting provider

  They provide and maintain the physical infrastructure, and run the software in this repository, through which operators interact with their deployments. Hosting providers are technical administrators for these deployments, ensuring availability and appropriate performance.

  We target small- to medium-scale hosting providers with 20+ physical machines.

- Operator

  They select the applications they want to run. They don't need to own hardware or deal with operations. Operators administer their applications in a non-technical fashion, e.g. as moderators. They pay the hosting provider for registering a domain name, maintaining physical resources, and monitoring deployments.

- User

  They are individuals using applications run by the operators, and e.g. post content.

Glossary

- Fediverse

  A collection of social networking applications that can communicate with each other using a common protocol.

- Application

  User-facing software (e.g. from Fediverse) configured by operators and used by users.

- Configuration

  A collection of settings for a piece of software.

Example: Configurations are deployed to VMs.

- Provision

  Make a resource, such as a virtual machine, available for use.

- Deploy

  Put software onto computers. The software includes technical configuration that links software components.

- Migrate

  Move service configurations and deployments (including user data) from one hosting provider to another.

- Run-time backend

  A type of digital environment one can run operating systems such as NixOS on, e.g. bare-metal, a hypervisor, or a container run-time.

- Provider

  An interface against which we deploy to a run-time backend.

- Provider configuration

  A configuration that specifies resources made available to deploy to and how to access these.

- Resource

  A resource is any external entity that we need for our set-up This may include e.g. hypervisors, file systems, DNS entries, VMs or object storage instances.

Technologies used

This is an incomplete and evolving list of core components planned to be used in this project. It will grow to support more advanced use cases as the framework matures.

Nix and NixOS

NixOS is a Linux distribution with a vibrant, reproducible and security-conscious ecosystem. As such, we see NixOS as the only viable way to reliably create a reproducible outcome for all the work we create.

Considered alternatives include:

- containers: do not by themselves offer the needed reproducibility

Proxmox

Proxmox is a hypervisor, allowing us to create VMs for our applications while adhering to our goal of preventing lock-in. In addition, it has been packaged for Nix as well, simplifying our requirements to users setting up our software.

Considered alternatives include:

- OpenNebula: seemed less mature

Garage

Garage is a distributed object storage service. For compatibility with existing clients, it reuses the protocol of Amazon S3.

Considered alternatives include:

- file storage: less centralized for backups

Architecture

At the core of Fediversity lies a NixOS configuration module for a set of selected applications.

- We will enable using it with **different run-time environments**, such as a single NixOS machine or a ProxmoX hypervisor.
- Depending on the targeted run-time environment, deployment may involve NixOps4 or OpenTofu as an **orchestrator**.
- We further provide demo front-end for **configuring applications** and configuring **run-time backends**.

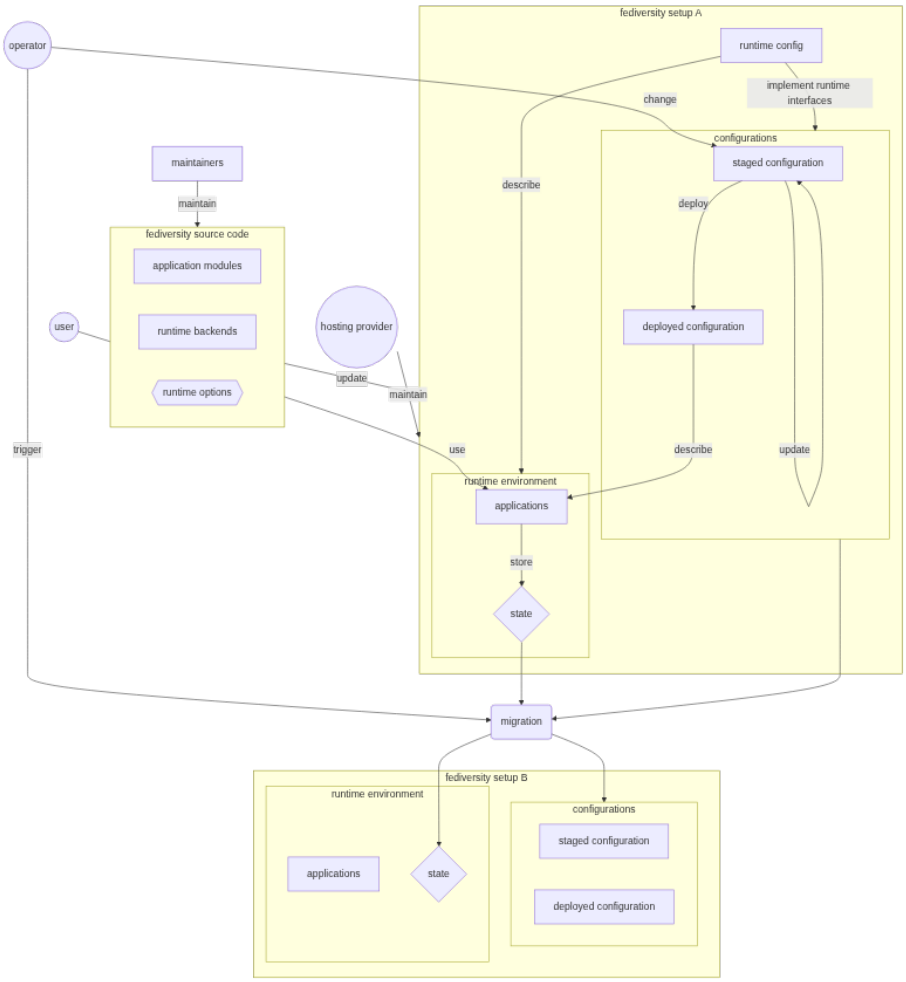
To ensure reproducibility, all software will be packaged with Nix.

To reach our goals, we aim to implement the following interactions.

The used legend is as follows:

- Circle: actor
- Angled box: type
- Rectangle: value
- Rounded box: function
- Diamond: state
- Arrow: points towards dependant

For further info on components see the glossary.



Configuration data flow

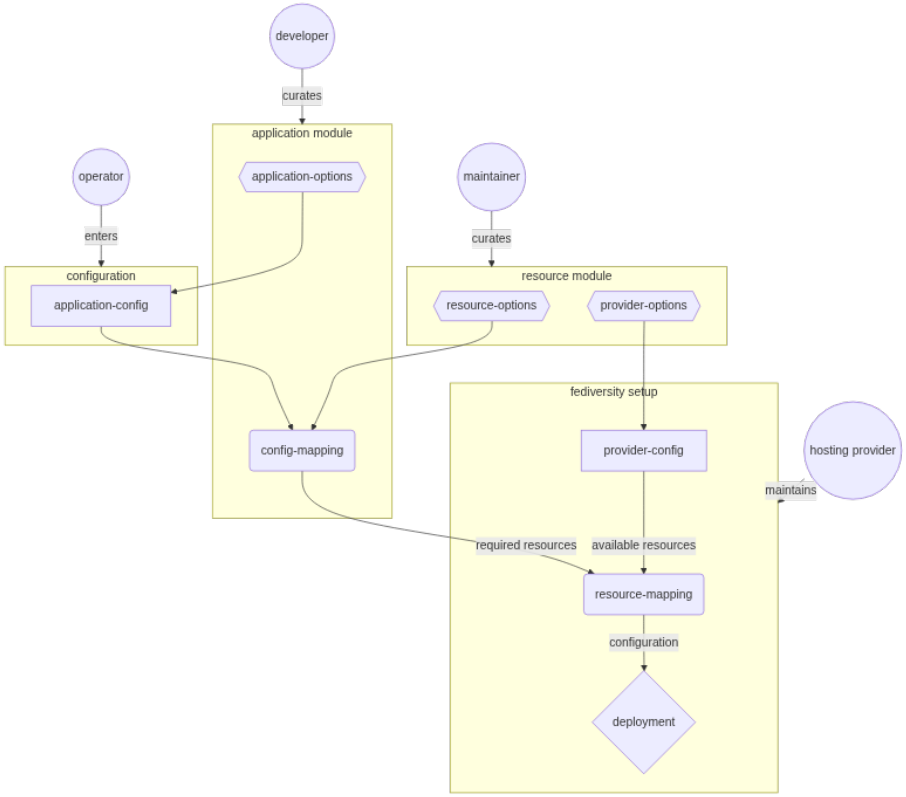
This data flow diagram refines how a deployment is obtained from an operator's application configuration and a hosting provider's runtime setup.

An **application module** specifies operator-facing **application options**, and a **configuration mapping** which determines the application's underlying implementation. Application modules can be supplied by external developers, which would curate application modules against that interface.

For its runtime setup, a hosting provider has to supply a **resource mapping** that would take their self-declared **provider configuration** (which determines the *available* resources) and the output of an application's resource mapping (which determine resource *requirements*) and produce a **configuration**. This configuration ships with a mechanism to be *deployed* to the infrastructure (which

is described by the environment, and features the required resources), where it will accumulate **application state**.

Applications and runtime environments thus interface through **resources**, the properties of which are curated by Fediversity maintainers.



Service portability

The process of migrating one's applications to a different host encompasses:

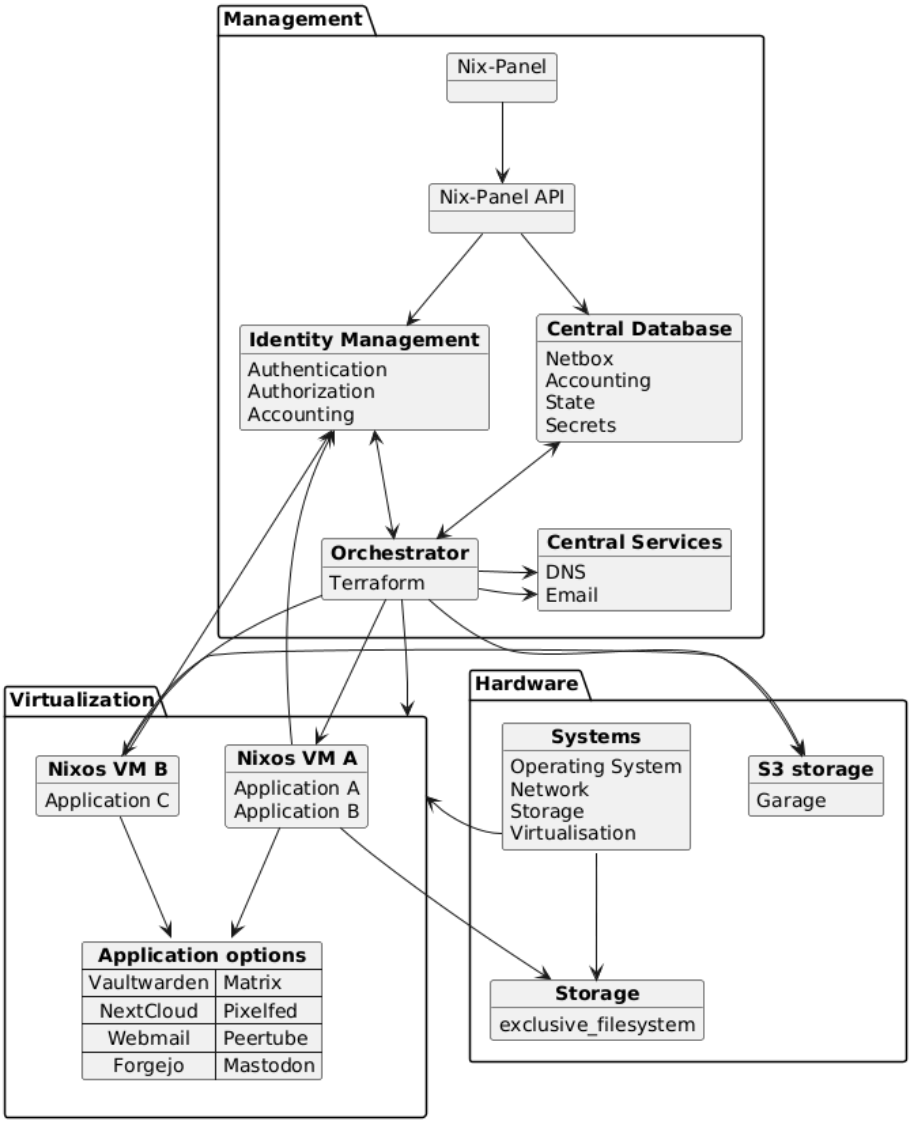
1. Domain registration: involves a (manual) update of DNS records at the registrar
2. Deploy applications: using the reproducible configuration module
3. Copy application data:
   - Run back-up/restore scripts
   - Run application-specific migration scripts, to e.g. reconfigure connections/URLs

Data model

Whereas the bulk of our configuration logic is covered in the configuration schema, implemented here and tested here, our reference front-end applications will store data. The data model design for the configuration front-end needed support the desired functionality is as follows, using the crow's foot notation to denote cardinality:

Host architecture

Whereas the core abstraction in Fediversity is a NixOS configuration module, a more full-fledged example architecture of the web host use-case we aim to support as part of our exploitation would be as follows, where virtual machines in question run Fediversity to offer our selected applications:



CI / CD

In our simplest set-up, continuous integration and continuous deployment pipelines are handled using Forgejo's built-in runner, with relevant secrets handled using Forgejo secrets. Jobs we handle using CI include linting, formatting, testing, and a periodic life-cycle management job to keep our dependencies up-to-date. In a future iteration, we may make use of Gerrit to better manage our review process for incoming merge requests.