openIDL V1 - Architecture - DevOps

Developing for openIDL

openIDL is an open source project. We will follow best practices for such projects.

> **General Guidelines**
>
> This page describes the practices for developing / contributing to openIDL
>
> All commits should correspond to an issue in the GItHub repository.
>
> All commits should mention the issue number at the beginning of the commit. Use the format
>
> `#8 Updated the x to fix y`
>
> All pull/merge requests must reference the issue number.

Pull Requests

All pull requests must be reviewed by someone other than the developer.

All pull requests require DCO - this requires local machine pull signoff and push. Follow directions in the pull request DCO details.

Tagging and Versioning

Use github releases to manage the iterations of the project in the different branches.

See https://docs.github.com/en/github/administering-a-repository/releasing-projects-on-github/managing-releases-in-a-repository

GitFlow

Let's follow GitFlow: https://datasift.github.io/gitflow/IntroducingGitFlow.html

the branches are:

- main - protected branch from which the UAT and Production builds are sourced
- test - protected branch from which the integration test environment is built
- dev - each developer has their own branch which may correspond to a cloud account

pull requests are used to promote from dev to test and from test to main

github actions are used to build, run unit tests, and deploy to the environment

README.md files provide help in. understanding each sub project and how to test it locally

the repository is a monorepo. It holds all the different components in subdirectories. Each one can be built and tested separately.

common components are developed following this process:

Common Component Development

Development on Local Machine

Create a branch

make update to the common code

make dependency from dependent code as relative path

run tests

commit to branch in git

Testing with Dependent Apps

identify all apps that are dependent and should be updated

for each we need to test them

pull the changed common component code

use relative path dependency

run tests

Deploying as Package

update version of common lib package.json

pull request

merge into main

CI/CD auto deploys new version to github packages

Update and retest dependents

for each dependent including original app that drove the change

update package.json to use the versioned dependency

rerun tests

commit and push updates to dependent app

pull request

merge

CI/CD auto whatever

DevOps Diagram

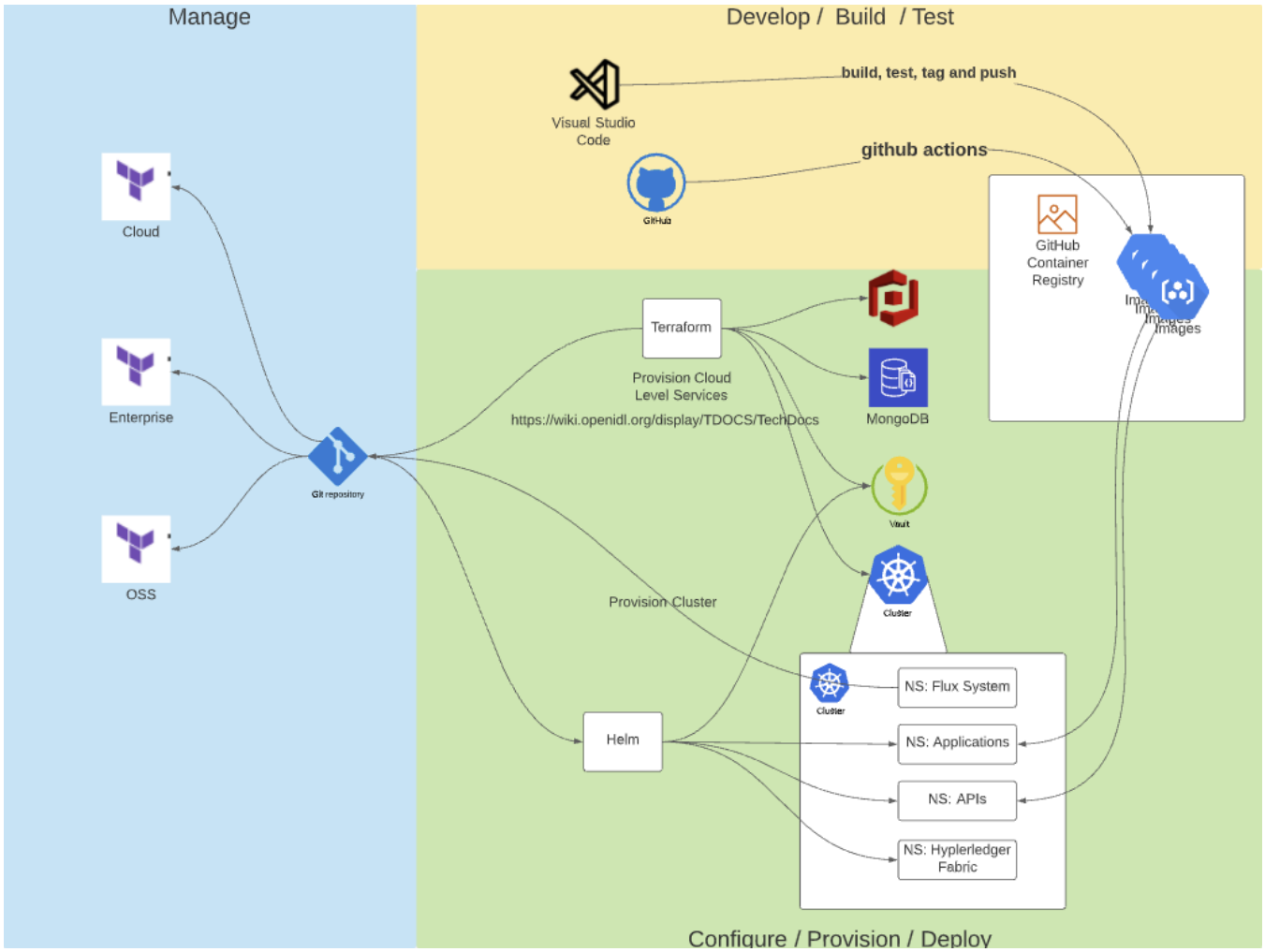
Technologies

Hashicorp Terraform

Blockchain Automation Framework

Sets up the blockchain nodes etc.

Helm



blocked URL

Here we discuss the DevOps for openIDL.

Since this is a distributed collection of nodes there are network and node level needs.

There are three areas to the dev ops architecture for openIDL

1. Manage
2. Develop / Build / Test
3. Configure / Provision / Deploy

Manage

Managing the configuration in Git is one of the tenets of the architecture.

The git repository is multi-part. The network administrator manages a reference configuration that is used to set up any of the types of nodes. There is one reference configuration for each type of node. In the data call app, there is a Multi-Tenant Node, a Carrier Node and an Analytics Node.

The node owners will each establish a repository to hold their configuration. This should be linked somehow to the administrator repository, so that updates can be merged using git workflows.

Updates to the configuration for a node in git trigger a workflow that can include a consent step for the node owner before configuration kicks off.

Develop / Build / Test

The Develop / Build / Test phase produces usable artifacts for the Deploy stage.

Developing for openIDL can be done locally. You can create images and save them in your minikube docker or you can just run the particular component out of code.

Configure / Provision / Deploy

The Deploy stage assumes that the assets required (terraform, helm, container images) are all built and properly packaged.