idpbuilder: Build and iterate Locally, deploy to Cloud

Same code in Dev locally as in Production on the Cloud. Faster iteration.

colum.mccoole@analect.com
colum-mccoole-746b946a
analect/technical-docs-hierarchy

Topic Navigation

How does this topic relate to other relevant content in this space.

Developer Tooling for Platforms: idpbuilder Concept

`idpbuilder` is an Internal development platform binary launcher developed under the auspices of CNOE. Essentially, it is looking to make it easier to get started with assembling a mix of applications that go to make up a platform. Getting all these working in conjunction with eachother in a kubernetes context can be challenging. The group behind CNOE thought it would be useful to settle on key open-source technologies that get used for IDPs and to set these up in a way that gave users a starting-point for building and working with such systems. Wrapping these inside idpbuilder helps with a faster developer loop.

As an idpbuilder user, you can spin-up a complete internal developer platform (IDP) using industry standard technologies like Kubernetes, Argo, and Backstage with only Docker required as a dependency.

This can be useful in several ways:

1. **Rapid Deployment:** Create a reference implementation of an IDP with minimal setup time.
2. **CI Integration:** Easily incorporate idpBuilder into your continuous integration workflows for comprehensive testing.
3. **Local Development:** Provide IDP engineers with a consistent and easily reproducible local development environment.

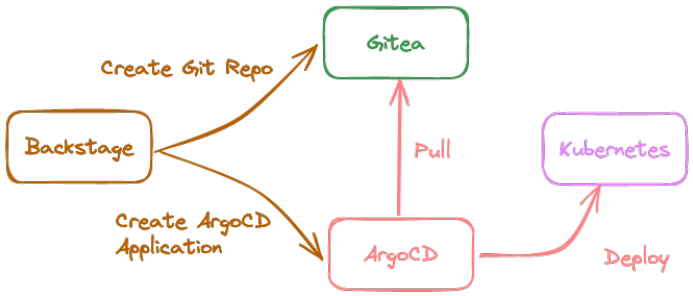

Figure 1. Backstage on top of idpbuilder infrastructure

idpbuilder Simplified Architecture

`idpbuilder` comes with a set of technologies that enables GitOps workflows all contained within the ephemeral environment. It does this by provisioning a `kind` cluster, `Gitea` server, `ArgoCD`, and `ingress-nginx`.

- **ArgoCD** is the GitOps solution to deploy manifests to Kubernetes clusters. In this project, a package is an ArgoCD application.
- **Gitea** server is the in-cluster Git server that ArgoCD can be configured to sync resources from. You can sync from local file systems to this.
- **Ingress-nginx** is used as a method to access in-cluster resources such as ArgoCD UI and Gitea UI.

Once installed, `idpbuilder` passes control over these packages to ArgoCD by storing manifests in Gitea repositories then creating ArgoCD applications. From here on, ArgoCD manages them based on manifests checked into Git repositories.

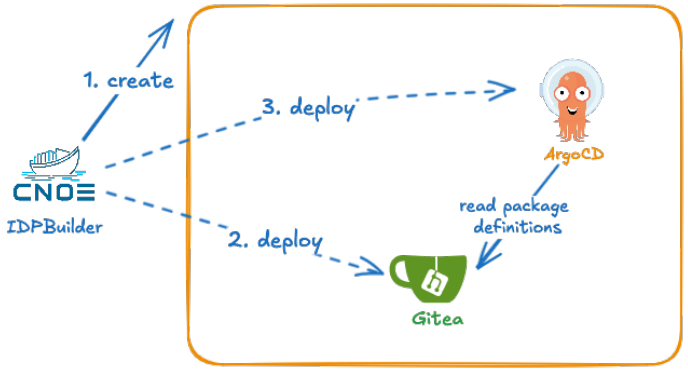

Figure 2. idpbuilder Simplified Architecture

idpbuilder with Backstage, Argo Workflows, Spark

Getting a reference-implementation up-and-running in 5 minutes involves running this one-liner (below). Reference Figure 1, let's review what happens:

```
idpbuilder create --use-path-routing --package https://github.com/cnoe-io/stacks//ref-implementation
```

1. Backstage creates a git repo, then pushes templated contents to it.
2. Backstage creates an ArgoCD Applic. and points it to the git repository.
3. Backstage registers the app as a component in Backstage.
4. ArgoCD deploys the manifests stored in the repo to the cluster.
5. Backstage retrieves the app's health from ArgoCD API, then displays it.

CNOE (Cloud Native Operational Excellence) Tenets

The goal for the CNOE framework is to bring together a cohort of enterprises operating at the same scale so that they can navigate their operational technology decisions together, de-risk their tooling bets, coordinate contribution, and offer guidance to large enterprises on which CNCF (cloud-native computing foundation) technologies to use together to achieve the best cloud efficiencies.

The CNOE working group will operate based on the following tenets:

1. **Open source first:** use of open source technology is prioritized over proprietary technology for each of the technology verticals discussed later in the doc. This helps ensure alignment across all the participating members by allowing them to coordinate on collaborations while having the freedom to update and modify a given technology to their needs
2. **Community driven:** Decisions on the direction of the working group is driven by the community and its governing body. This involves the selection of technologies, level of commitment, and level of contribution.
3. **Tools and not Practices:** CNOE offers suggestions on which tools to use and with what configurations. What practices a given company builds around and above those tools is out of scope for CNOE.
4. **Powered by Kubernetes, but not limited to orchestrate to Kubernetes:** The CNOE working group relies heavily on the success of the CNCF community to choose technologies that are deemed useful to the type of operations required by the community. As such, Kubernetes is considered the de-facto environment to operate CNOE tooling. However, choosing of Kubernetes as the operating environment, does not require for it to be the environment to orchestrate against. Using the right infrastructure as code tooling, the CNOE community can choose to orchestrate against any compute platform of their choice.
5. **Standardized to the infrastructure, customizable by the developers:** CNOE aims at addressing the usability requirements of its stakeholders. While the requirements of the platform could be enforced by the security engineers and infrastructure operators, the usability of it needs to be guaranteed by platform operators and application developers.
6. **Built to be shared:** All CNOE deliverables including the reference architecture and the deployment packages will be developed out in the open and by collaboration of all its participating members and with the goal of making it sharable and usable by the larger open source community of interest.