An Ideal CI/CD System

Dec 10, 2022

A good CI/CD system means developer productivity. What an ideal CI/CD system looks like today.

Serverless – A CI/CD system should be serverless. There's no reason to be managing Jenkins or individual nodes. Managing CI/CD machines makes it really easy to introduce configuration and environment drift, which causes hard to trace down bugs. Container-native is usually a good choice, especially if you often deploy software inside containers, but anything that provides a real ephemeral environment is good.

On-prem (Cloud-prem) – For small projects, it's fine to use hosted CI/CD services, but for anything bigger than a toy project, you should probably be spinning up your CI/CD runners inside your own cloud account.

Minimal permissions / native IAM – Workers should authenticate in a cloud-native way (such as OIDC) and be deployed with a minimal set of permissions. While it might seem like a hassle to expand the scope of workers while projects deploy a wider variety and number of services, limiting the blast radius and separating out environments will save you from a whole class of bugs and mistakes.

Easy to debug – Easy to debug means that it's possible to run the pipeline in a meaningful way locally. It also means that logs and artifacts are easily accessible from runs.

Triggers, but not complicated ones – GitHub Actions provides a good model for running workflows automatically (from pull requests or merge events) as well as manual triggers. However, don't build a complicated business process around the manual triggers.

Code, not YAML – Following my TypeScript for Infrastructure post, I believe it's much easier to design pipelines in a full programming language rather than a configuration one. You get reusability, control flow, strong typing, and other properties that you probably want when you're describing a dependency graph.

DAG – There's been some experimentation with having a fully event-driven CI/CD system. Generally, I think it's best to mostly have a static DAG that encodes the dependency graph. Events might trigger the initial workflow and might be emitted at the competition (or failure) of the DAG, but don't fire in between.

Subscribe for daily posts on startups & engineering.

larry@google.com

Subscribe