Case Study: Switch from Jenkins CI to Circle CI

See an up to date, HTML version of this case study on andrewhowden.com

Skills

This case study required:

- Several years of experience with CI/CD systems and how they evolve over time
- Implementation of CI/CD in several servers
- Ability to use, build, store and manipulate containers
- Ability to understand Linux within the context of a container
- Understanding of CPU / memory usage and how that affects workloads, including how the CFS scheduler limits workloads.
- Using professional networks to reach out to the CircleCI engineering team

Requirements

The application on which I worked was deployed via the continuous integration server Jenkins, which functioned reasonably well. However, it suffered from some issues which meant it required an uncomfortable amount of time and money to support.

Specifically:

- It had not been updated in a very long time
- It was running on bare metal infrastructure not managed by infrastructure as code techniques
- The definitions themselves were only expressed in groovy script in the UI
- The small helper scripts and other tooling to launch the system was written in a variety of languages, none of which the organisation was familiar.

To address this, the organisation planned to move away from the Jenkins CI serer to another server. After interviewing many of the team leads, team members and systems team members I wrote a proposal in the form of an RFC[1] with the judgement criteria for success being:

- Reduce maintenance on the CI/CD pipeline
- Reduced time to ship changes to production
- Increased number of developers who had made changes to the CI pipeline
- Decreased perceived complexity of the pipeline

Plan

In the past I had written CI/CD pipelines on several different servers, including DroneCI, TravisCI, BitBucket Pipelines, Google Cloud Build and Jenkins. In my experience to ensure the pipeline was predictable and maintainable it needed to be subject to the same software delivery life cycle as the application itself, such as including a testing period, code review, QA and so fourth.

Further, the advent of Docker had created a whole series of CI tooling for which environments were both cheap and could be discarded at the end of the build. This prevented the aggregation of tooling that no one understands, as well as dependency problems over year-long periods.

Initially my goal was to use the open source DroneCI, deployed as a helm chart on top of Google Container Engine but there was concern from the systems team that hosting the CI software at all would create too great a maintenance burden. The next CI that most closely matched my experience was CircleCI.

Implementation

The implementation was completed in stages:

Lints

In the first iteration of CircleCI a new test, not previously implemented in the CI pipeline was introduced into CircleCI to see whether the system was stable and provided value. More specifically the linting tool PHPCS was added to the pipeline to enforce PSR2 lints.

The pipeline itself functioned admirably well after a small period adjusting on which files the linter ran, however the results of the linter indicated the way in which the code was written did not adhere to the guidelines it was supposed to and there was some tension in determining whether to refactor old code to meet these guidelines.

The pipeline aspect of this had no issues, so the decision was made to go ahead.

Unit Tests, Integration Tests

The next task was to implement both Unit Tests and Integration Tests into the pipeline.

These tests were somewhat flaky, and had dependencies on the Jenkins environment in which they were being executed that were not well documented such as the PHP version, direct access to Docker and so fourth.

This part of work was completed primarily by a colleague who showed a particular interest in learning more about the CI/CD process but had extensive knowledge of the application and its design.

Together we implemented the unit tests and integration tests.

Deployment

Unfortunately, the deployment was not successfully implemented. The deployment relied on a redesign of the infrastructure slated for completion but not completed for some period of time while the new system was running.

Evaluation

The move to CircleCI was moderately successful. However, there was an issue in which the tests would take a much longer period of time (+100%) on the CircleCI machines than on the bare metal servers.

This was investigated and the likely determinant being the clock speed on the CircleCI build servers being considerably slower (~2.8 - 3.2GHz*) than the bare metal servers (~4Ghz). That, in conjunction with the multi-tenant nature of CircleCI meant that the builds were slower.

Ultimately the systems team decided that this speed loss was too great, and the only way to address it was to put the service on AWS build service where the amount of compute available was potentially greater. The CircleCI service was torn down and replaced with AWS build service which reduced time spent in the integration test period by ~20 - 40% from the increased CircleCI period.

At the time of writing it is believed that the new deployments were still not completed.

- If memory serves. It was ~ 3Ghz.