Case 04: Technical and Business Processes

You may have seen this situation before. The customer experience is that pushing to production is scary, because you never know when things might break. This customer could only push to production once a month. There was significant risk of downtime. And when there is downtime, the application breaks, and that impacts revenue. So the summary is… we need to be able to develop and deploy features, need to present to production, without it being an event.

A customer had this interesting business requirement...

- Pushing to Prod is a big event and happens once a month.
- Significant risk of downtime due to unforeseen issues.
- Downtimes exceeding SLA have a revenue impact.
- Need to develop and deploy features without burning the house down. Pushing to Prod should be a non-event.

There are two passes through the problem in the architectural process. First, there is the business pass, which includes both the business challenge and the people processes - understanding what roles there are and what actions the people need to be able to take. And then there is the technical pass which is mapping all these needs and procedures to a technical solution.

We mapped that to technical requirements like this…

Establish CI/CD pipeline:

- Single source repo per product; git-flow as branching model.
- Automate build, self-testing, rapid.
- Automate deployment.
- Setup robust monitoring, logging and alerting for visibility.

Promote team culture:

- Test Driven Development.
- Push often, address broken builds immediately.
- Transparency.
- Change management / Release process.

To push to production on demand, we needed to analyze the development process. We figured out that a single source repo made the most sense for the whole team. We decided to go with git-flow as a branching model instead of other kinds of development. We automated the build process. But also we made sure that the builds were self-testing using SALT test coverage. And we measured the build process and made sure that it was rapid. We also automated the

deployment of the solution software. Couple this automation with monitoring, logging, and alerting, and you get a nice build-and-deploy system that can be handed off to a team.

But that doesn't solve the entire problem. The system has to be used. And this team was not going to be accustomed to using the development paradigm we had implemented. So we also needed to enhance their processes. This primarily had to do with how test and development worked together. In the new paradigm they would start writing the testing along with development. The new way was to push to production often and push early, and as soon as something breaks -- fix it. This meant a new team culture, one of accountability and transparency. Where all the stakeholders could participate in identifying and resolving a problem. And that meant change management and leadership buy-in was necessary for the technical solution to be successful.

And this is how we implemented that technical requirement.

- Cloud Source Repositories - for hosting their repositories
- Container Builder - that builds the docker container images
- Container Registry - that hosts those container images
- Google Kubernetes Engine + Helm - for running and managing
- Spinnaker - for CDP - continuous deliver
- Cloud Load Balancing, Google Cloud’s operations suite, Cloud IAM + Service Accounts - management constructs for proper visibility