Principles of Software Construction: Objects, Design, and Concurrency

DevOps

Charlie Garrod    Chris Timperley
Administrivia

• Homework 6 has been released
  – Sequential implementation due by Tuesday, Nov. 26
  – Parallel implementation due by Wednesday, Dec. 4
Outline

• DevOps and CI/CD
• Large-Scale Version Control
• Release Management
Devs, Ops, and The Wall of Confusion

OLD WAY:

CODE → BUILD → TEST → RELEASE → OPERATE

DEVELOPERS

HANDOFF

HANDOFF

HANDOFF

Wall of Confusion

DEVELOPMENT
- Deliver new features
- Product oriented
- Innovation

OPERATIONS
- Guarantee stability
- Service oriented
- Rationalization

https://www.plutora.com/blog/what-is-enterprise-devops
https://www.yudiz.com/welcome-devops-prevent-defects/
DevOps: Development / Operations

https://blog.gds-gov.tech/that-ci-cd-thing-principles-implementation-tools-aa8e77f9a350
Principle: Automation Everywhere

```bash
#!/bin/bash

pip install "$1" &
easy_install "$1" &
brew install "$1" &
npm install "$1" &
yum install "$1" & dnf install "$1" &
docker run "$1" &
pkg install "$1" &
apt-get install "$1" &
sudo apt-get install "$1" &
steamcmd +app_update "$1" validate &
git clone https://github.com/"$1"/"$1" &
cd "$1"; ./configure; make; make install &
curl "$1" | bash &
```

https://blog.chef.io/automate-all-the-things/
Principle: Code as Configuration

- Manage configuration files in your version control system
  - Travis, Gradle, Jenkins, ...
- Packaging and installation
  - Docker, package.json, setup.py, pom.xml, ...
- Infrastructure and deployment
  - Docker Compose, Ansible, Puppet, Kubernetes
  - Manage servers and resources
- ...
Installation and configuration can be annoying

- Build flags
- Build order
- Static dependencies
- Dynamic dependencies
- Environment variables
- Configuration files
- DLL hell
- ...

https://llvm.org/docs/GettingStarted.html
https://blog.codinghorror.com
- Uses lightweight containerization
- Full setup including configuration
- Separate container for each service
  - web server, database, logic, …
  - reduced attack surface
- Used in development and deployment

FROM ubuntu:18.04
RUN apt-get update \
  && apt-get install -y \
      apt-transport-https \
      ca-certificates \
      curl \
      docker \
      software-properties-common \
      git \
      python \
      python-pip \
      python-dev \
      patchelf \
      python3 \
      python3-pip \
      openjdk-8-jdk \
      locales \
      vim \
  && pip install pipenv \
&& curl -fsSL https://download.docker.com/linux/ubuntu/gpg | apt-key add - \
&& add-apt-repository \
  "deb [arch=amd64] https://download.docker.com/linux/ubuntu \
  $(lsb_release -cs) \
  stable" \
  && apt-get update \
&& apt-get install -y docker-ce \
&& apt-get autoremove -y \
&& apt-get clean \
&& rm -rf /var/lib/apt/lists/* /tmp/* /var/tmp/*
RUN sed -i -e 's/# en_US.UTF-8 UTF-8/en_US.UTF-8 UTF-8/en_US.UTF-8 UTF-8/' /etc/locale.gen \
    && locale-gen
ENV LANG en_US.UTF-8
ENV LANGUAGE en_US:en
ENV LC_ALL en_US.UTF-8

Containerized Applications

- App A
- App B
- App C
- App D
- App E
- App F

Docker

Host Operating System

Infrastructure
Docker and DockerHub

- Build an image for each release
- Quickly rollback to stable versions

```
$ docker pull mysql:8.0
$ docker push christimperley/darjeeling
```
Principle: Rapid Releases and Feedback

• Remove the manual and ceremonial aspects from releases
  – Possibly continuous releases
  – Incremental rollout; quick rollback

• Get feedback on your changes ASAP
  – Continuously measure quality, refine implementation, and rerelease
Principle: Shared Responsibility

- Breakdown the “Wall of Confusion”
- Improve collaboration between dev. and ops. teams
- Reduce “throw it over the fence” syndrome
- Treat failures as a learning experience...
Aside: Postmortems

Example Postmortem

Shakespeare Sonnet++ Postmortem (incident #465)

Date: 2015-10-21
Authors: jennifer, martym, agoogler
Status: Complete, action items in progress
Summary: Shakespeare Search down for 66 minutes during period of very high interest in Shakespeare due to discovery of a new sonnet.
Impact: Estimated 1.21B queries lost, no revenue impact.
Root Causes: Cascading failure due to combination of exceptionally high load and a resource leak when searches failed due to terms not being in the Shakespeare corpus. The newly discovered sonnet used a word that had never before appeared in one of Shakespeare's works, which happened to be the term users searched for. Under normal circumstances, the rate of task failures due to resource leaks is low enough to be unnoticed.
Trigger: Latent bug triggered by sudden increase in traffic.
Resolution: Directed traffic to sacrificial cluster and added 10x capacity to mitigate cascading failure. Updated index deployed, resolving interaction with latent bug. Maintaining extra capacity until surge in public interest in new sonnet passes. Resource leak identified and fix deployed.
Detection: Borgmon detected high level of HTTP 500s and paged on-call.

Action Items:

<table>
<thead>
<tr>
<th>Action Item</th>
<th>Type</th>
<th>Owner</th>
<th>Bug</th>
</tr>
</thead>
<tbody>
<tr>
<td>Update playbook with instructions for responding to cascading failure</td>
<td>mitigate</td>
<td>jennifer</td>
<td>n/a DONE</td>
</tr>
<tr>
<td>Use flux capacitor to balance load between clusters</td>
<td>prevent</td>
<td>martym</td>
<td>Bug 5554823 TBD</td>
</tr>
</tbody>
</table>

https://blog.codinghorror.com/the-project-postmortem/
https://www.developer.com/design/article.php/3637441
https://landing.google.com/sre/books/
Two sides to DevOps

Operations-oriented
- Manage servers automatically
- Easier to identify and fix bugs
- Automatic logging, monitoring, and operations

Developer-oriented
- Agile releases!
- Easier to share and understand code
- Faster onboarding
- Safely push code through CI/CD pipeline
Continuous Integration and Continuous Deployment

https://dzone.com/articles/learn-how-to-setup-a-cicd-pipeline-from-scratch
Continuous Integration
Continuous Integration at Google

Google workflow

- Sync user workspace to repo
- Write code
- Code review
- Commit

- All code is reviewed before commit (by humans and automated tooling)
- Each directory has a set of owners who must approve the change to their area of the repository
- Tests and automated checks are performed before and after commit
- Auto-rollback of a commit may occur in the case of widespread breakage

Additional tooling support

<table>
<thead>
<tr>
<th>Critique</th>
<th>Code review</th>
</tr>
</thead>
<tbody>
<tr>
<td>CodeSearch*</td>
<td>Code browsing, exploration, understanding, and archeology</td>
</tr>
<tr>
<td>Tricorder**</td>
<td>Static analysis of code surfaced in Critique, CodeSearch</td>
</tr>
<tr>
<td>Presubmits</td>
<td>Customizable checks, testing, can block commit</td>
</tr>
<tr>
<td>TAP</td>
<td>Comprehensive testing before and after commit, auto-rollback</td>
</tr>
<tr>
<td>Rosie</td>
<td>Large-scale change distribution and management</td>
</tr>
</tbody>
</table>

** See "Tricorder: Building a program analysis ecosystem", In International Conference on Software Engineering (ICSE), 2015
Aside: Sapienz and SapFix at Facebook

Outline

• DevOps and CI/CD
• Large-Scale Version Control
• Release Management
How do you scale to 2 billion lines of code?

Google repository usage

Human users
- 25 thousand Googlers in dozens of offices around the world

On an average workday
- 15 thousand commits by humans
- 30 thousand commits by automated systems
- Billions of file read requests* (800K QPS at daily peak)

*Google recently open-sourced a subset of the internal build system, see [www.bazel.io](http://www.bazel.io)

Millions of changes committed (cumulative)

Some perspective

Linux kernel
- 15 million lines of code in 40 thousand files (total)

Google repository
- 15 million lines of code in 250 thousand files changed per week, by humans
- 2 billion lines of code, in 9 million source files (total)
A recent history of code organization

https://www.toptal.com/front-end/guide-to-monorepos
Monolithic repositories (Monorepos)

A single version control repository containing multiple:
- Projects
- Applications
- Libraries
Scaling Mercurial at Facebook: Insights from the Other Side
Durham Goode, Facebook

With thousands of commits a week across hundreds of thousands of files, Facebook’s main source repository is enormous—many times larger than even the Linux kernel, which checked in at 17 million lines of code and 44,000 files in 2013. Given our size and complexity—and Facebook’s practice of shipping code twice a day—improving our source control is one way we help our engineers move fast.

Choosing a source control system

Two years ago, as we saw our repository continue to grow at a staggering rate, we sat down and extrapolated our growth forward a few years. Based on those projections, it appeared likely that our then-current technology, a Subversion server with a Git mirror, would become a productivity bottleneck very soon. We looked at the available options and found none that were both fast and easy to use at scale.

Our code base has grown organically and its internal dependencies are very complex. We could have spent a lot of time making it more modular in a way that would be friendly to a source control tool, but there are a number of benefits to using a single repository. Even at our current scale, we often make large changes throughout our code base, and having a single repository is useful for continuous modernization. Splitting it up would make large, atomic refactorings more difficult. On top of that, the idea that the scaling constraints of our source control system should dictate our code structure just doesn’t sit well with us.

We realized that we’d have to solve this ourselves. But instead of building a new system from scratch, we decided to take an existing one and make it scale. Our engineers were comfortable with Git and we...
The largest Git repo on the planet

Brian
May 24th, 2017

It's been 3 months since I first wrote about our efforts to scale Git to extremely large projects and teams with an effort we called "Git Virtual File System". As a reminder, GVFS, together with a set of enhancements to Git, enables Git to scale to VERY large repos by virtualizing both the git folder and the working directory. Rather than download the entire repo and checkout all the files, it dynamically downloads only the portions you need based on what you use.

A lot has happened and I wanted to give you an update. Three months ago, GVFS was still a dream. I don't mean it didn't exist — we had a concrete implementation, but rather, it was unproven. We had validated on some big repos but we hadn't rolled it out to any meaningful number of engineers so we had only conviction that it was going to work. Now we have proof.

Today I want to share our results. In addition, we're announcing the next steps in our GVFS journey for customers, including expanded open sourcing to start taking contributions and improving how it works for us at Microsoft, as well as for partners and customers.

Windows is live on Git

Over the past 3 months, we have largely completed the rollout of Git/GVFS to the Windows team at Microsoft.

As a refresher, the Windows code base is approximately 35M files and, when checked into a Git repo, results in a repo of about 300GB. Further, the Windows team is about 4,000 engineers and the engineering system produces 1,769 daily "lab builds" across 440 branches in addition to thousands of pull request validation builds. All 3 of the dimensions (file count, repo size and activity), independently, provide daunting scaling challenges and taken together they make it unbelievably challenging to create a great experience. Before the move to Git, in Source Depot, it was spread across 40+ depots and we had a tool to manage operations that spanned them.

As of my writing 3 months ago, we had all the code in one Git repo, a few hundred engineers using it and a small fraction (<10%) of the daily build load. Since then, we have rolled out in waves across the engineering team.

The first, and largest, jump happened on March 22nd when we rolled out to the Windows OneCore team of about 2,000 engineers. Those 2,000 engineers worked in Source Depot on Friday, went home for the weekend and came back Monday morning to a new experience based on Git. People on my team were holding their breath that whole weekend, praying we weren't going to be pummeled by a mob of angry engineers who showed up Monday unable to get any work done.
Monorepos are also used by open source projects
Monorepos tend to use a common build system

**Bazel**

**Buck**

**Pants: A fast, scalable build system**

Pants is a build system designed for codebases that:

- Are large and/or growing rapidly.
- Consist of many subprojects that share a significant amount of code.
- Have complex dependencies on third-party libraries.
- Use a variety of languages, code generators and frameworks.

Pants supports Java, Scala, Python, C/C++, Go, Javascript/Node, Thrift, Protobuf and Android code. Adding support for other languages, frameworks and code generators is straightforward.

Pants is a collaborative open-source project, built and used by Twitter, Foursquare, Square, Medium and other companies.
Why do these companies use monorepos?
Benefits of Monorepos

• **Cheaper code reuse**
  – Extract reusable code into a new component
  – Easily use that code from elsewhere! No need for more repos.

• **Browse, read, and search through the entire codebase**
  – Works with grep, IDEs, and special tools out of the box

• **Atomic refactorings with a single commit**
  – Switch from an old API to a new API in a single commit

• **Easier to test, debug, review, and deploy projects that span multiple applications**
  – Easier to collaborate across projects and teams.
  – No more internal dependency management!
Drawbacks of Monorepos

• Require collective responsibility for team and developers
• Require trunk-based development
  – More on that later...
• Force you to have only one version of everything
• Scalability requirements for the repository
• Can be hard to deal with updates around things like security issues
• Build and test bloat without very smart build system
• Slow VCS without very smart system
• Permissions?
Outline

- DevOps and CI/CD
- Large-Scale Version Control
- Release Management
How and when should software be released?

Google repository

- 15 million lines of code in 250 thousand files \textit{changed per week, by humans}
- 2 billion lines of code, in 9 million source files (total)
Principle: Quick to Deploy; Slow to Release

“Get your **** together; fix it in production.”
Chuck Rossi, former Release Engineering Director at Facebook
Trunk-based development at Google

Combined with a centralized repository, this defines the monolithic model:

- Piper users work at “head”, a consistent view of the codebase
- All changes are made to the repository in a single, serial ordering
- There is no significant use of branching for development
- Release branches are cut from a specific revision of the repository
Trunk-based development

Whole dev-team shared branch called Trunk (or master)

Each committer (preferably a pair-programming duo) in this Trunk-Based Development way of working is streaming small commits straight into the trunk (or master) with a pre-integration step of running the build first (which must pass).

Key

Developers active here

Developers NOT active here

Whole dev-team shared branch called Trunk (or master)

Trunk-Based Development at scale is best done with short-lived feature branches: one person over a couple of days (max) and flowing through Pull-Request style code-review & build automation before “integrating” (merging) into the trunk (or master)

https://trunkbaseddevelopment.com
Cherrypicking

https://www.atlassian.com/blog/git/the-essence-of-branch-based-workflows
Fresh release branch every week

https://engineering.fb.com/web/rapid-release-at-massive-scale/
The number of commits in a branch cut became unsustainable.
Quasi-continuous push from master (1,000+ devs, 1,000 diffs/day); 10 pushes/day
Principle: Every feature is an experiment
CANYON TESTING

USER

ROUTER

MOST USERS (95%)

SOME USERS (5%)

OLD VERSION

WEB SERVER

APPLICATION SERVER

DATABASE SERVER

NEW VERSION

WEB SERVER

APPLICATION SERVER

DATABASE SERVER
Dark Launching

- Similar to canary testing
- Focuses on user response to frontend changes rather than performance of backend
- Measure user response via *metrics: engagement, adoption*
Aside: Opt-In Beta
Automated canary analysis at Netflix

- ~60,000 configuration changes per day, 
  ~4000 commits per day
- Bake an Amazon Machine Image (AMI) for each commit
- Deploy via Spinnaker and Kayenta
- Perform automated canary analysis.
  - If okay, switch to new version.
  - If bad, rollback to old version.

https://octopus.com/blog/blue-green-red-black
Control deployments at run-time using feature flags

https://martinfowler.com/articles/feature-toggles.html
Warning! Feature flags can be dangerous

In laymen’s terms, Knight Capital Group realized a $460 million loss in 45-minutes. Remember, Knight only has $365 million in cash and equivalents. In 45-minutes Knight went from being the largest trader in US equities and a major market maker in the NYSE and NASDAQ to bankrupt.

Summary

- DevOps brings development and operations together
  - Automation, Automation, Automation
  - Infrastructure as code
- Release management
  - Versioning and branching strategies
- Continuous deployment is increasingly common
- Exploit opportunities of continuous deployment; perform testing in production and quickly rollback
  - Experiment, measure, and improve