Principles of Software Construction: Objects, Design, and Concurrency

DevOps

Charlie Garrod **Chris Timperley**

Carnegie Mellon University
School of Computer Science

institute for SOFTWARE RESEARCH

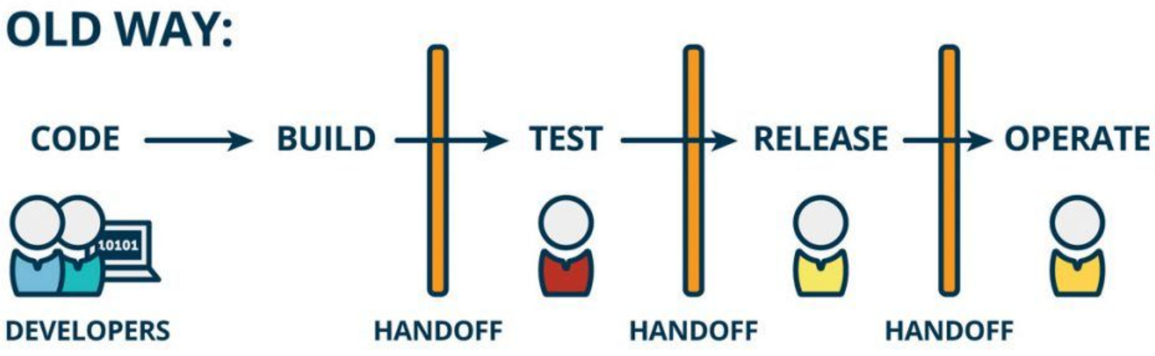
Administrivia

- 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

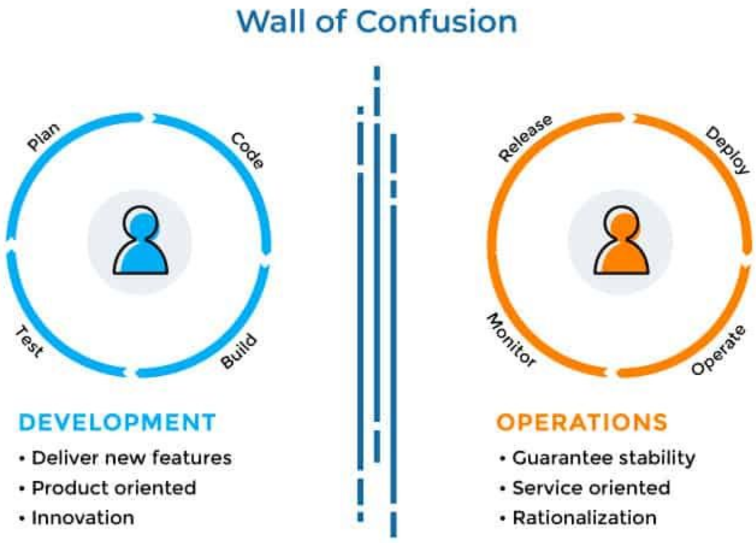
10101

DEVELOPERS

HANDOFF

HANDOFF

HANDOFF



Wall of Confusion

Plan

Code

Build

Test

Release

Deploy

Operate

Monitor

DEVELOPMENT

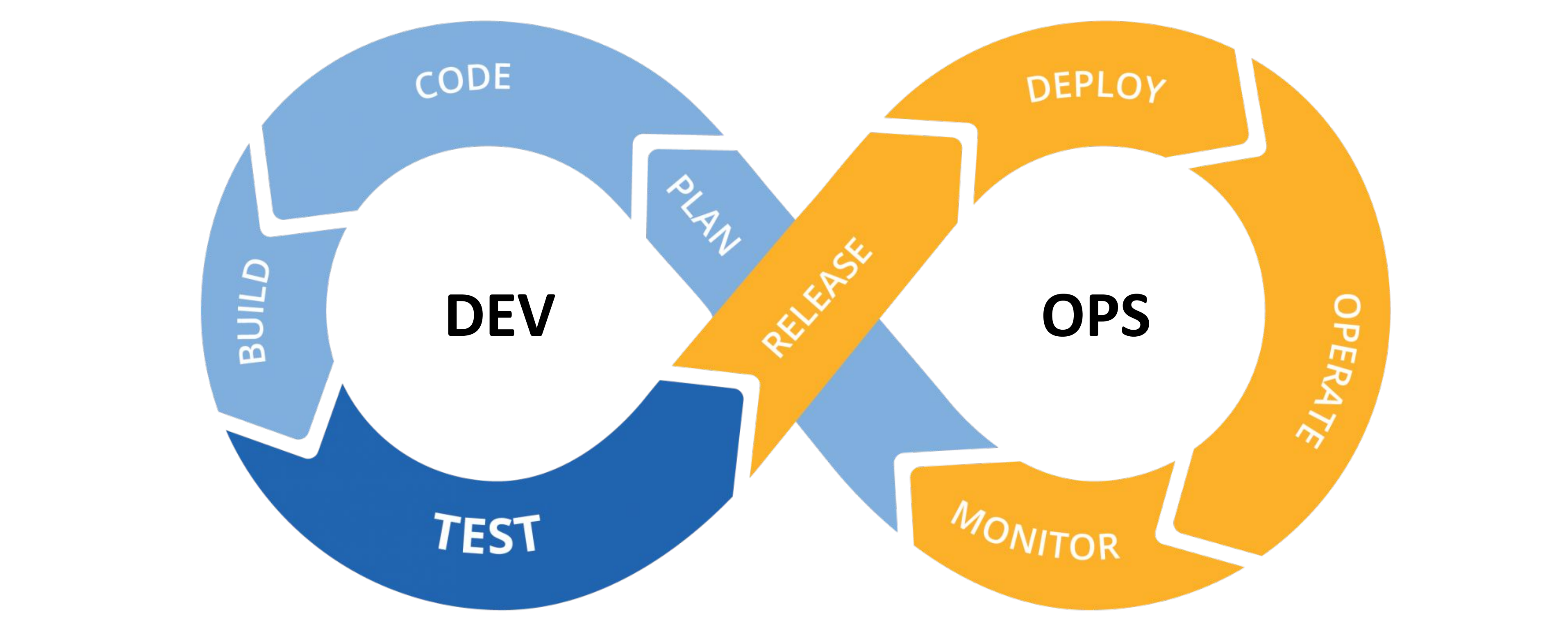
- Deliver new features
- Product oriented
- Innovation

OPERATIONS

- Guarantee stability
- Service oriented
- Rationalization

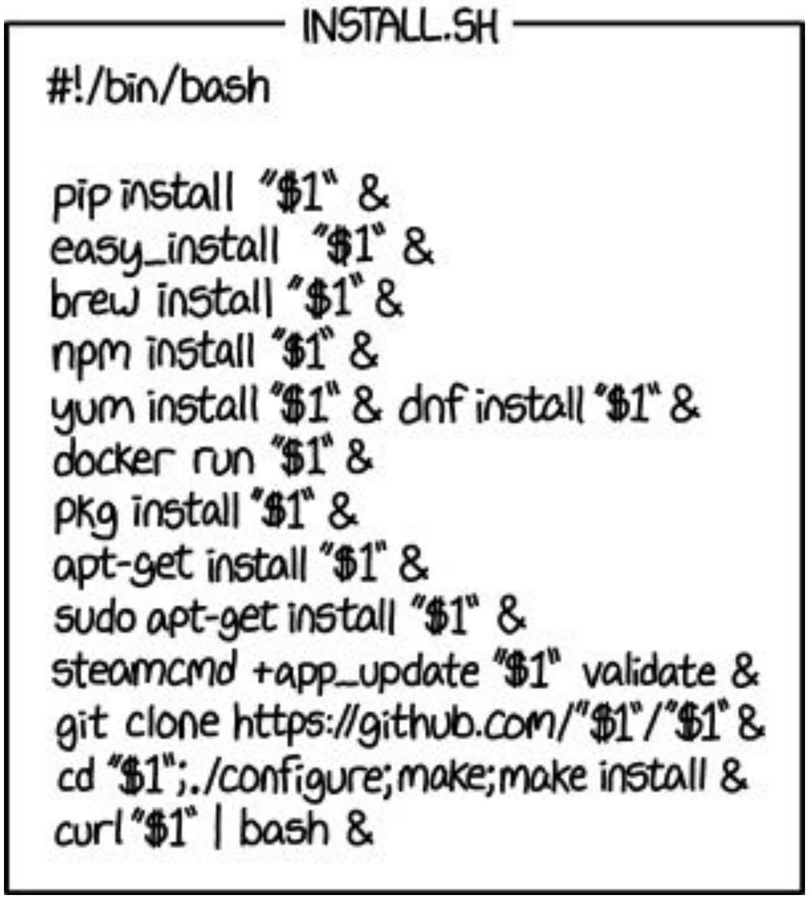
https://www.plutora.com/blog/what-is-enterprise-devops
https://www.yudiz.com/welcome-devops-prevent-defects/

DevOps: Development / Operations



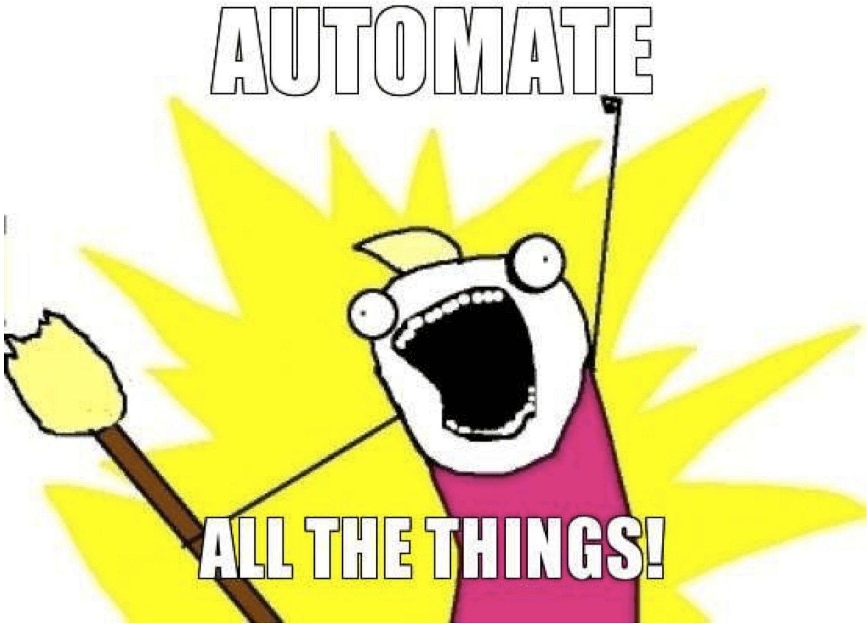
https://blog.gds-gov.tech/that-ci-cd-thing-principles-implementation-tools-aa8e77f9a350

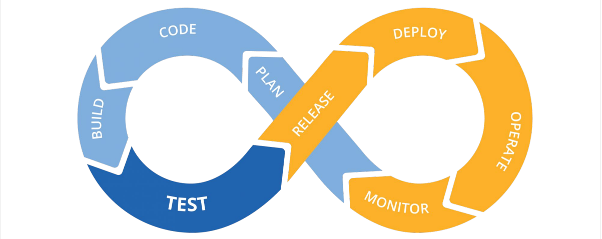
Principle: Automation Everywhere



INSTALL.SH

```
#!/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 &
```

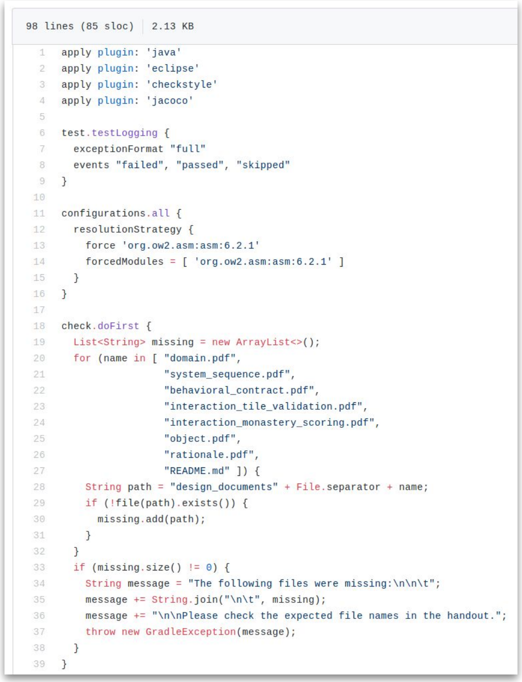




https://blog.chef.io/automate-all-the-things/

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
- …

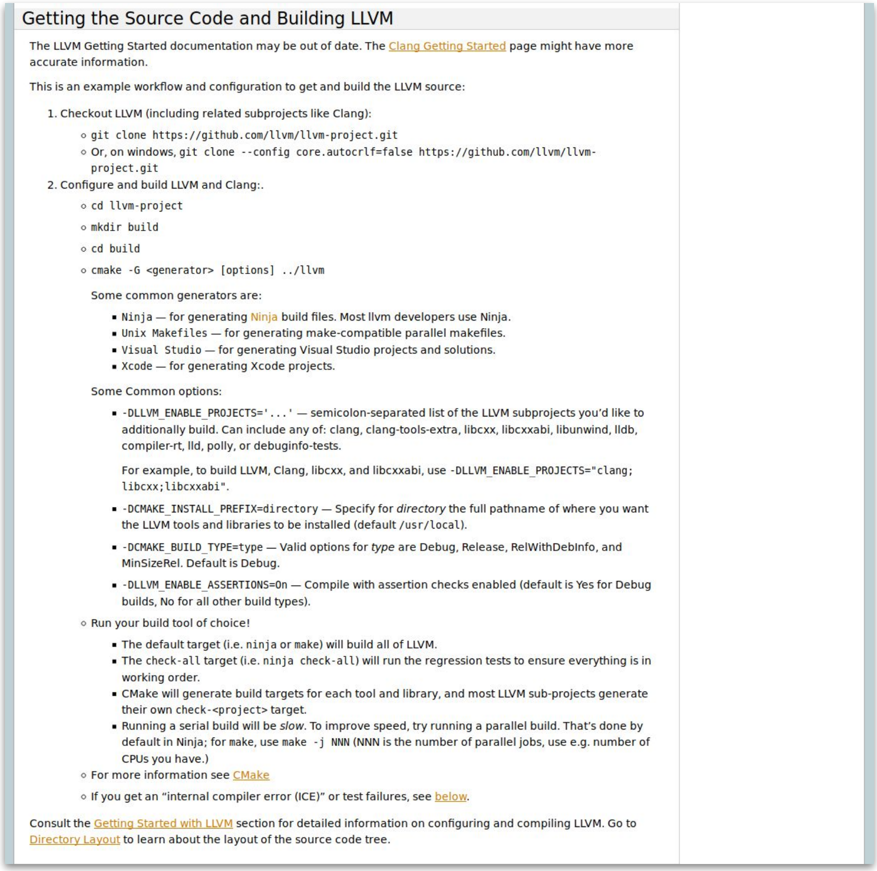


98 lines (85 sloc) | 2.13 KB

```
apply plugin: 'java'
apply plugin: 'eclipse'
apply plugin: 'checkstyle'
apply plugin: 'jacoco'

test.testLogging {
  exceptionFormat "full"
  events "failed", "passed", "skipped"
}

configurations.all {
  resolutionStrategy {
    force 'org.ow2.asm:asm:6.2.1'
    forcedModules = [ 'org.ow2.asm:asm:6.2.1' ]
  }
}

check.doFirst {
  List<String> missing = new ArrayList<>();
  for (name in [ "domain.pdf",
                 "system_sequence.pdf",
                 "behavioral_contract.pdf",
                 "interaction_tile_validation.pdf",
                 "interaction_monastery_scoring.pdf",
                 "object.pdf",
                 "rationale.pdf",
                 "README.md" ]) {
    String path = "design_documents" + File.separator + name;
    if (!file(path).exists()) {
      missing.add(path);
    }
  }
  if (missing.size() != 0) {
    String message = "The following files were missing:\n\n\t";
    message += String.join("\n\t", missing);
    message += "\n\nPlease check the expected file names in the handout.";
    throw new GradleException(message);
  }
}
```

Installation and configuration can be annoying

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



Getting the Source Code and Building LLVM

The LLVM Getting Started documentation may be out of date. The Clang Getting Started page might have more accurate information.

This is an example workflow and configuration to get and build the LLVM source:

1. Checkout LLVM (including related subprojects like Clang):
   - `git clone https://github.com/llvm/llvm-project.git`
   - Or, on windows, `git clone --config core.autocrlf=false https://github.com/llvm/llvm-project.git`
2. Configure and build LLVM and Clang:.
   - `cd llvm-project`
   - `mkdir build`
   - `cd build`
   - `cmake -G <generator> [options] ../llvm`

     Some common generators are:
     - `Ninja` — for generating Ninja build files. Most llvm developers use Ninja.
     - `Unix Makefiles` — for generating make-compatible parallel makefiles.
     - `Visual Studio` — for generating Visual Studio projects and solutions.
     - `Xcode` — for generating Xcode projects.

     Some Common options:
     - `-DLLVM_ENABLE_PROJECTS='...'` — semicolon-separated list of the LLVM subprojects you'd like to additionally build. Can include any of: clang, clang-tools-extra, libcxx, libcxxabi, libunwind, lldb, compiler-rt, lld, polly, or debuginfo-tests.

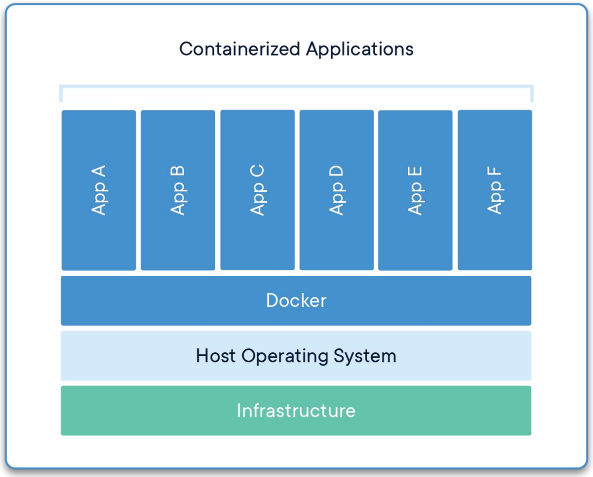
       For example, to build LLVM, Clang, libcxx, and libcxxabi, use `-DLLVM_ENABLE_PROJECTS="clang;libcxx;libcxxabi"`.
     - `-DCMAKE_INSTALL_PREFIX=directory` — Specify for *directory* the full pathname of where you want the LLVM tools and libraries to be installed (default `/usr/local`).
     - `-DCMAKE_BUILD_TYPE=type` — Valid options for *type* are Debug, Release, RelWithDebInfo, and MinSizeRel. Default is Debug.
     - `-DLLVM_ENABLE_ASSERTIONS=On` — Compile with assertion checks enabled (default is Yes for Debug builds, No for all other build types).
   - Run your build tool of choice!
     - The default target (i.e. `ninja` or `make`) will build all of LLVM.
     - The `check-all` target (i.e. `ninja check-all`) will run the regression tests to ensure everything is in working order.
     - CMake will generate build targets for each tool and library, and most LLVM sub-projects generate their own `check-<project>` target.
     - Running a serial build will be *slow*. To improve speed, try running a parallel build. That's done by default in Ninja; for `make`, use `make -j NNN` (NNN is the number of parallel jobs, use e.g. number of CPUs you have.)
   - For more information see CMake
   - If you get an "internal compiler error (ICE)" or test failures, see below.

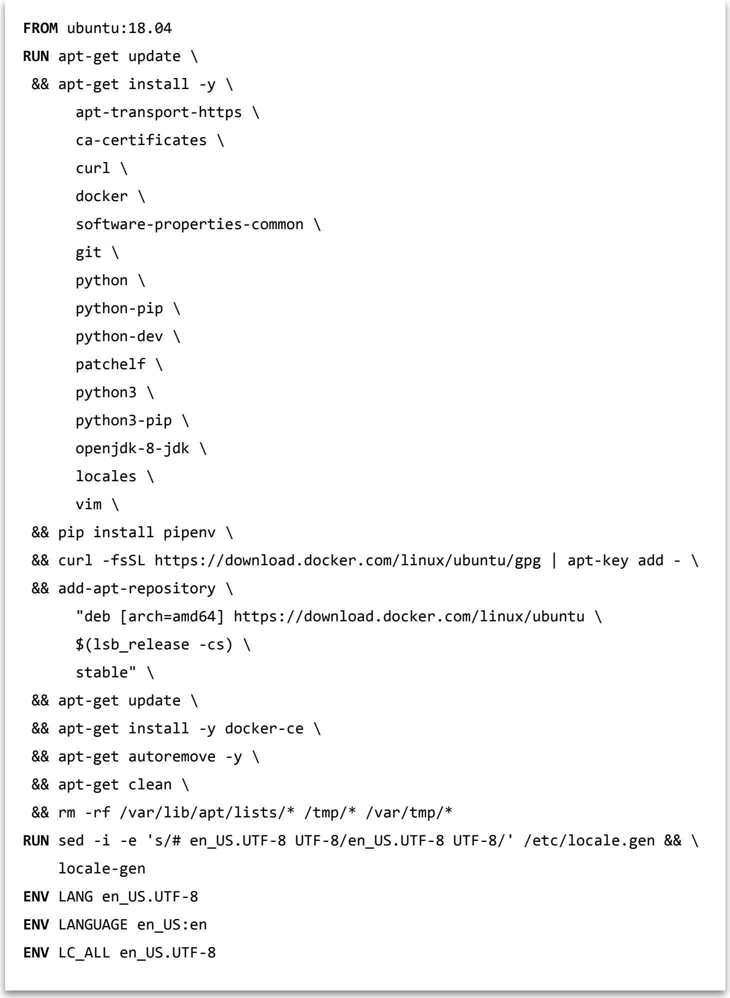
Consult the Getting Started with LLVM section for detailed information on configuring and compiling LLVM. Go to Directory Layout to learn about the layout of the source code tree.

https://llvm.org/docs/GettingStarted.html
https://blog.codinghorror.com

docker

- Uses lightweight containerization
- Full setup including configuration
- Separate container for each service
  - web server, database, logic, …
  - reduced attack surface
- Used in development and deployment



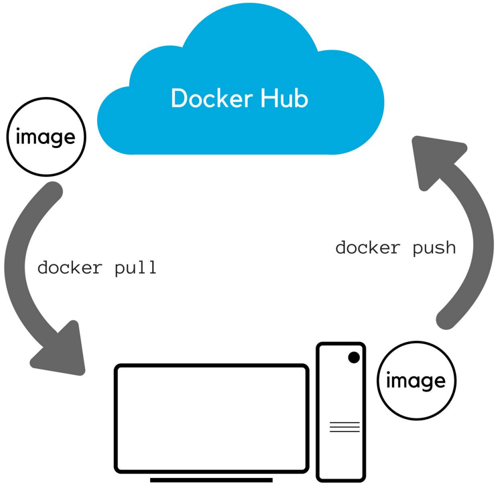


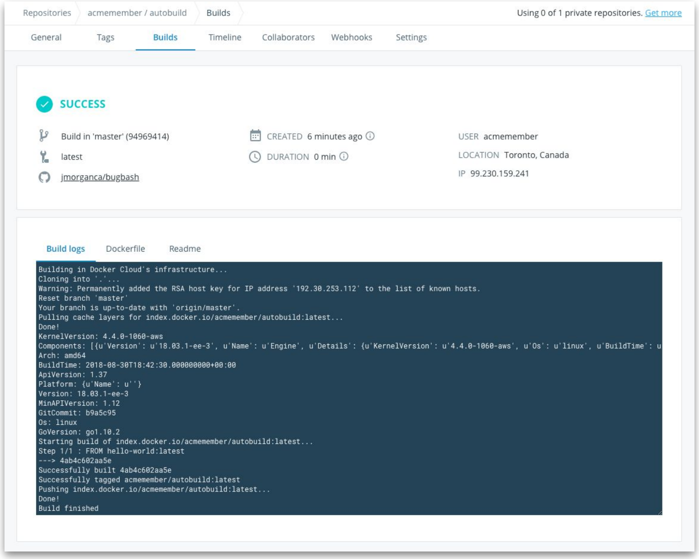
```
FROM ubuntu:18.04
RUN apt-get update \
 && apt-get install -y \
      apt-transport-https \
      ca-certificates \
      curl \
      docker \
      software-properties-common \
      git \
      python \
      python-pip \
      python-dev \
      patchelf \
      python3 \
      python3-pip \
      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/' /etc/locale.gen && \
    locale-gen
ENV LANG en_US.UTF-8
ENV LANGUAGE en_US:en
ENV LC_ALL en_US.UTF-8
```

Docker and DockerHub

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

```
$ docker pull mysql:8.0
$ docker push christimperley/darjeeling
```





SUCCESS

Build in 'master' (94969414)

latest

jmorganca/bugbash

CREATED 6 minutes ago

DURATION 0 min

USER acmemember

LOCATION Toronto, Canada

IP 99.230.159.241

```
Building in Docker Cloud's infrastructure...
Cloning into '.'...
Warning: Permanently added the RSA host key for IP address '192.30.253.112' to the list of known hosts.
Reset branch 'master'
Your branch is up-to-date with 'origin/master'.
Pulling cache layers for index.docker.io/acmemember/autobuild:latest...
Done!
KernelVersion: 4.4.0-1060-aws
Components: [{u'Version': u'18.03.1-ee-3', u'Name': u'Engine', u'Details': {u'KernelVersion': u'4.4.0-1060-aws', u'Os': u'linux', u'BuildTime': u
Arch: amd64
BuildTime: 2018-08-30T18:42:30.000000000+00:00
ApiVersion: 1.37
Platform: {u'Name': u''}
Version: 18.03.1-ee-3
MinAPIVersion: 1.12
GitCommit: b9a5c95
Os: linux
GoVersion: go1.10.2
Starting build of index.docker.io/acmemember/autobuild:latest...
Step 1/1 : FROM hello-world:latest
---> 4ab4c602aa5e
Successfully built 4ab4c602aa5e
Successfully tagged acmemember/autobuild:latest
Pushing index.docker.io/acmemember/autobuild:latest...
Done!
Build finished
```

https://docs.docker.com/docker-hub/builds/
https://static.packt-cdn.com/products/9781789137231/graphics/99abf1ea-4efe-4ccd-93c3-b36e80f3263c.png

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

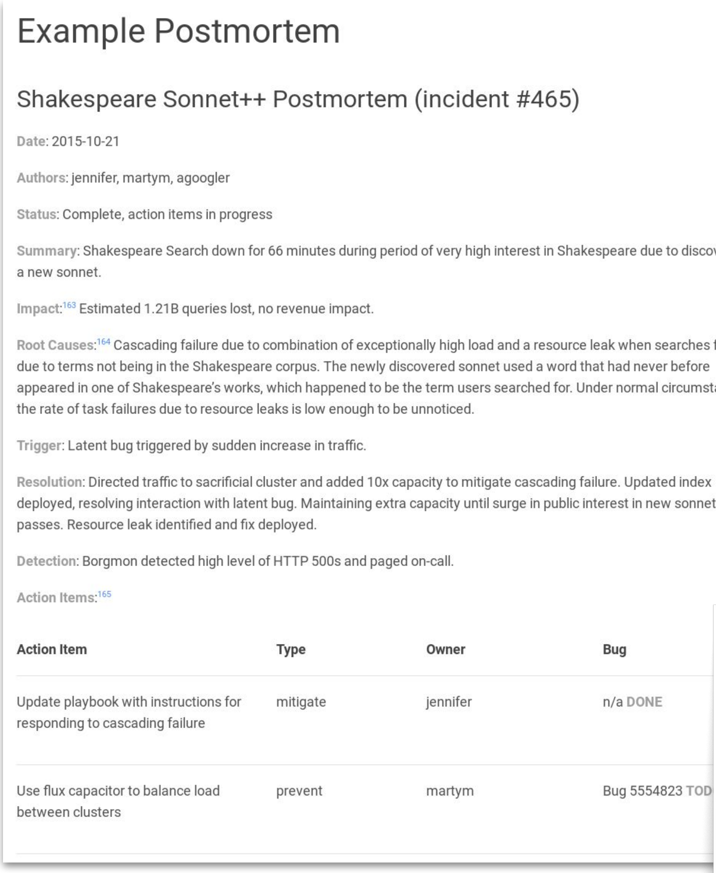
Travis CI

PHABRICATOR

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:[163] Estimated 1.21B queries lost, no revenue impact.

Root Causes:[164] 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.

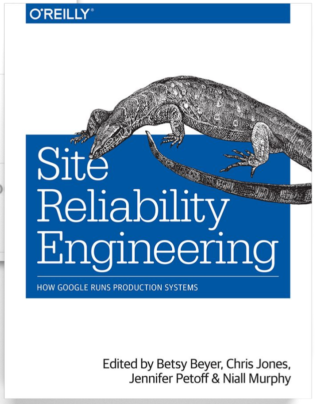
Trigger: Latent bug triggered by sudden increase in traffic.

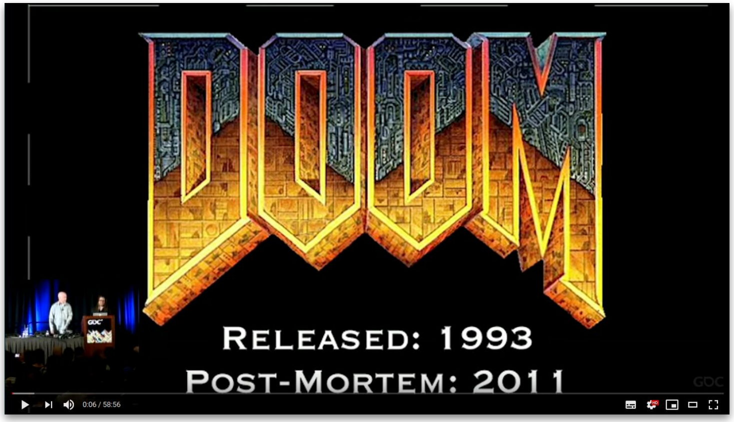
Resolution: Directed traffic to sacrificial cluster and added 10x capacity to mitigate cascading failure. Updated index deployed, resolving interaction with latent bug. Maintaining extra capacity until surge in public interest in new sonnet passes. Resource leak identified and fix deployed.

Detection: Borgmon detected high level of HTTP 500s and paged on-call.

Action Items:[165]

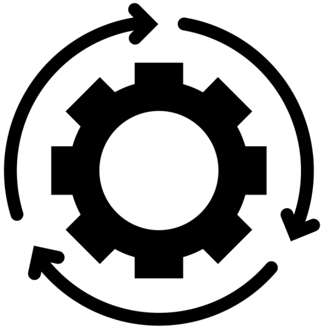
Action Item	Type	Owner	Bug
Update playbook with instructions for responding to cascading failure	mitigate	jennifer	n/a DONE
Use flux capacitor to balance load between clusters	prevent	martym	Bug 5554823 TOD


O'REILLY
Site Reliability Engineering
HOW GOOGLE RUNS PRODUCTION SYSTEMS
Edited by Betsy Beyer, Chris Jones, Jennifer Petoff & Niall Murphy

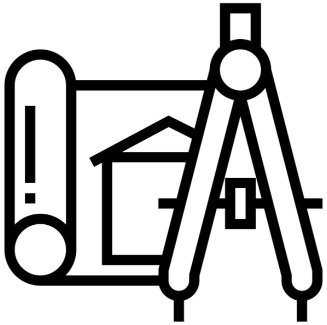


https://blog.codinghorror.com/the-project-postmortem/
https://www.developer.com/design/article.php/3637441
https://landing.google.com/sre/books/

Two sides to DevOps



Created by Shocho
from Noun Project



Created by Eucalyp
from Noun Project

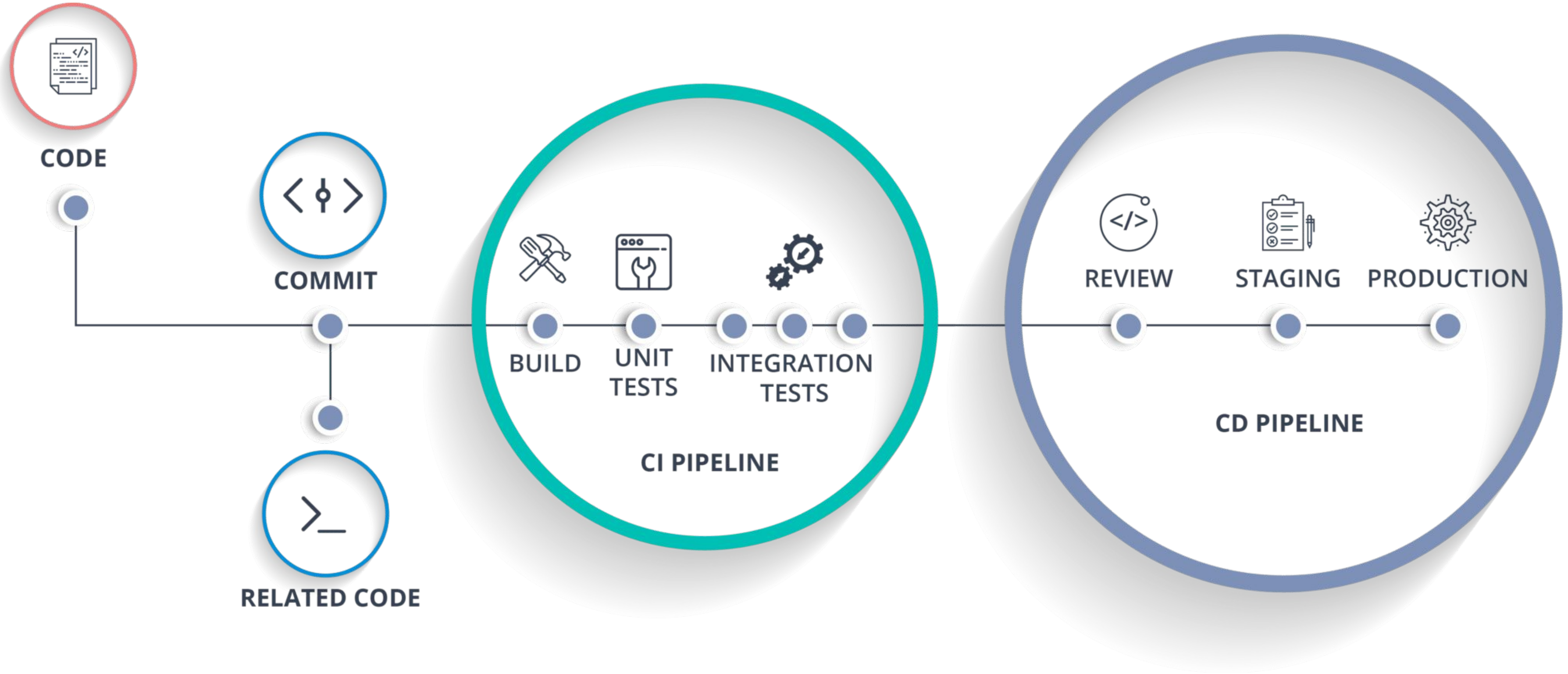
Operations-oriented

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

Developer-oriented

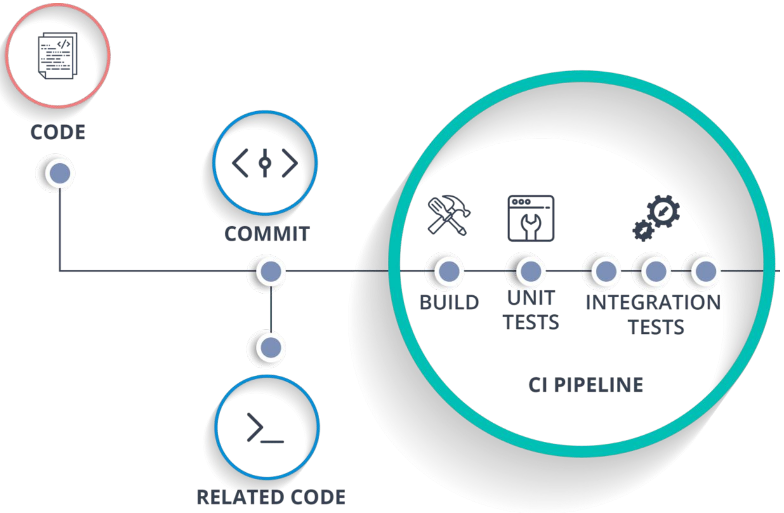
- Agile releases!
- Easier to share and understand code
- Faster onboarding
- Safely push code through CI/CD pipeline

Continuous Integration and Continuous Deployment



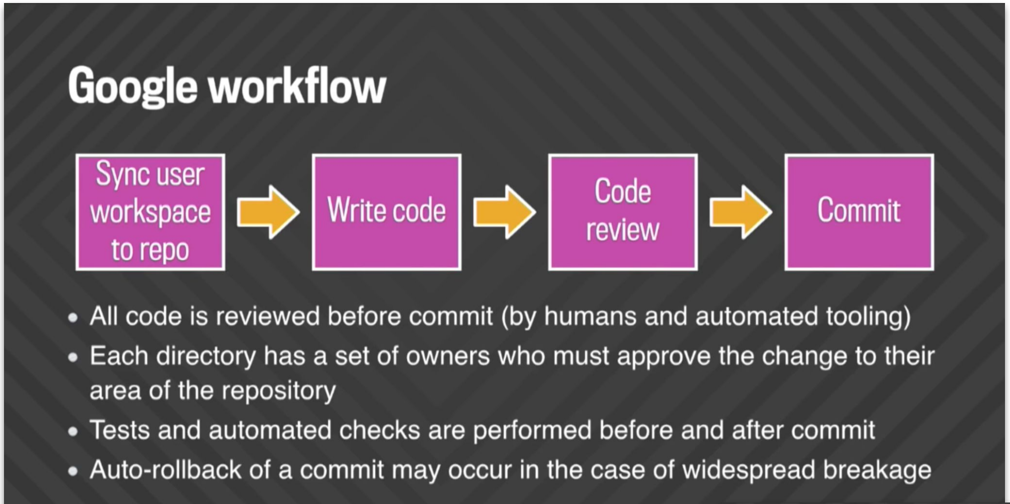
https://dzone.com/articles/learn-how-to-setup-a-cicd-pipeline-from-scratch

Continuous Integration



Travis CI JUnit 5 Gradle GitHub SpotBugs

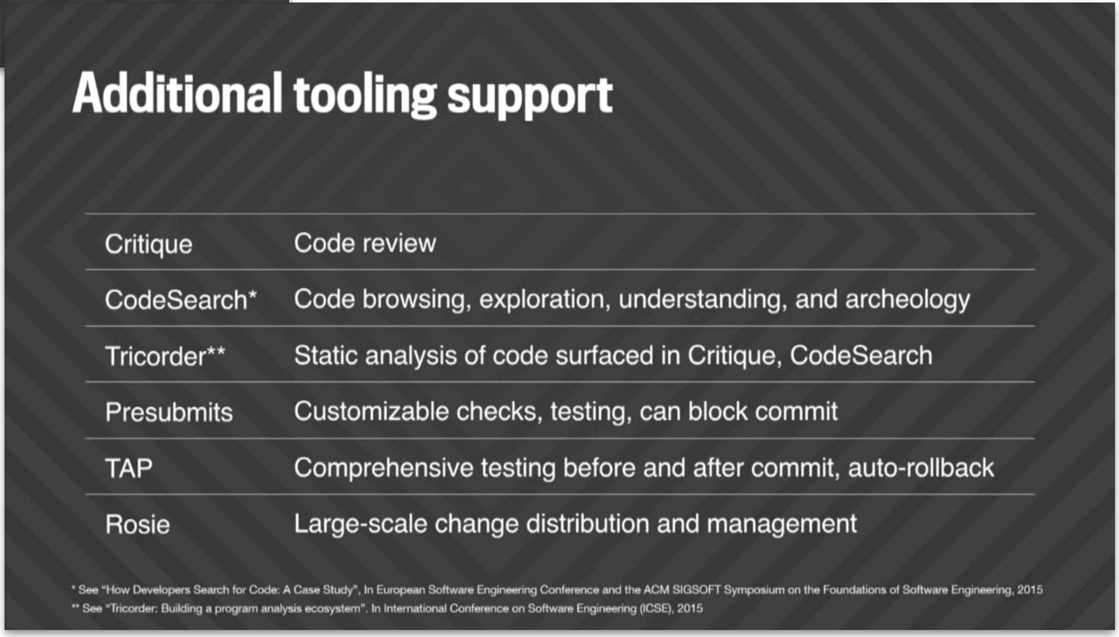
Continuous Integration at Google



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



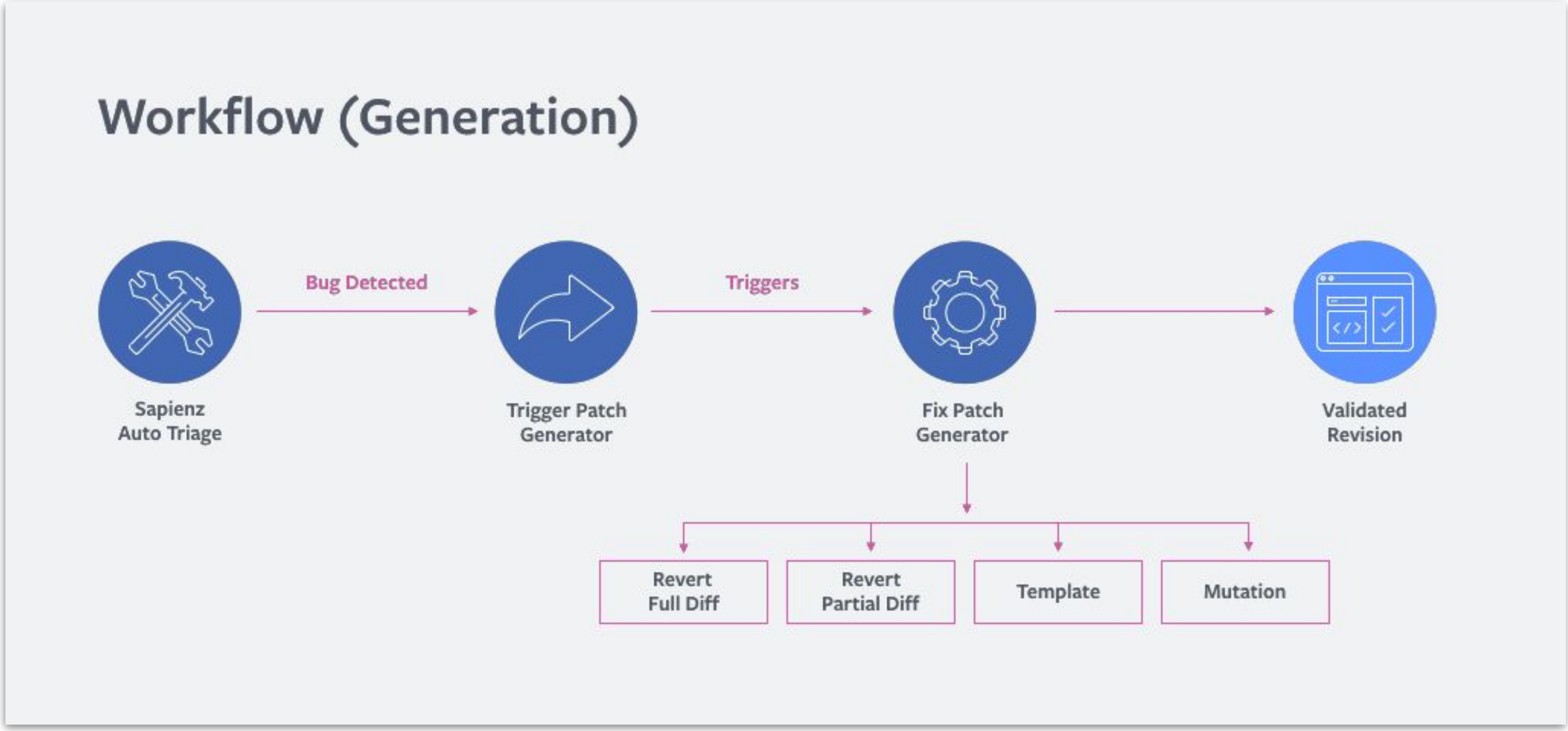
Additional tooling support

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

* See "How Developers Search for Code: A Case Study", In European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, 2015

** See "Tricorder: Building a program analysis ecosystem". In International Conference on Software Engineering (ICSE), 2015

Aside: Sapienz and SapFix at Facebook

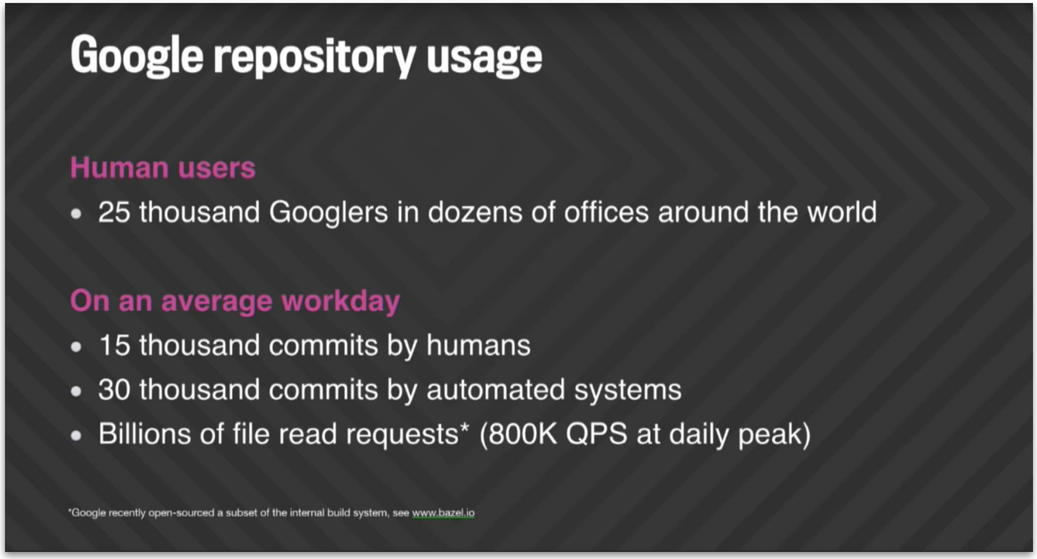


https://engineering.fb.com/developer-tools/finding-and-fixing-software-bugs-automatically-with-sapfix-and-sapienz/

Outline

- DevOps and CI/CD
- Large-Scale Version Control
- Release Management

How do you scale to 2 billion lines of code?



Google repository usage

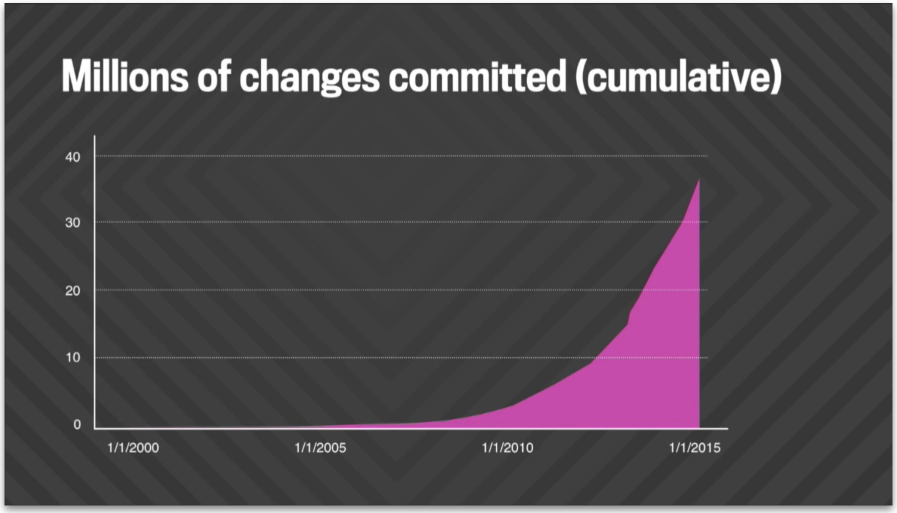
Human users

- 25 thousand Googlers in dozens of offices around the world

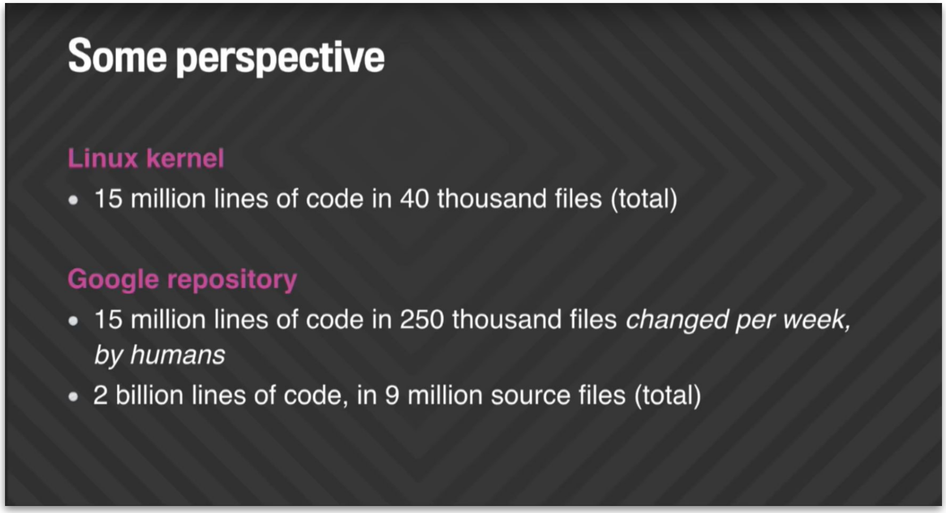
On an average workday

- 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



Millions of changes committed (cumulative)



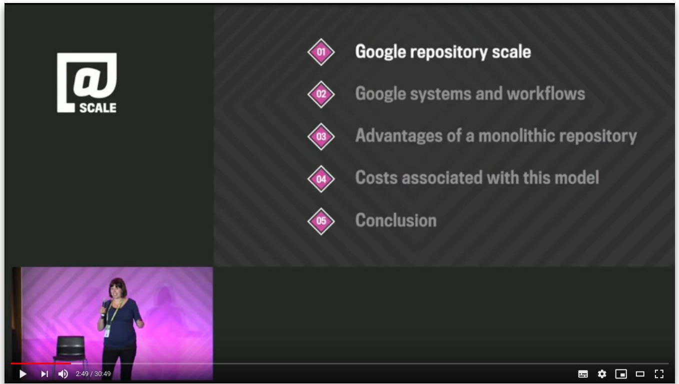
Some perspective

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)



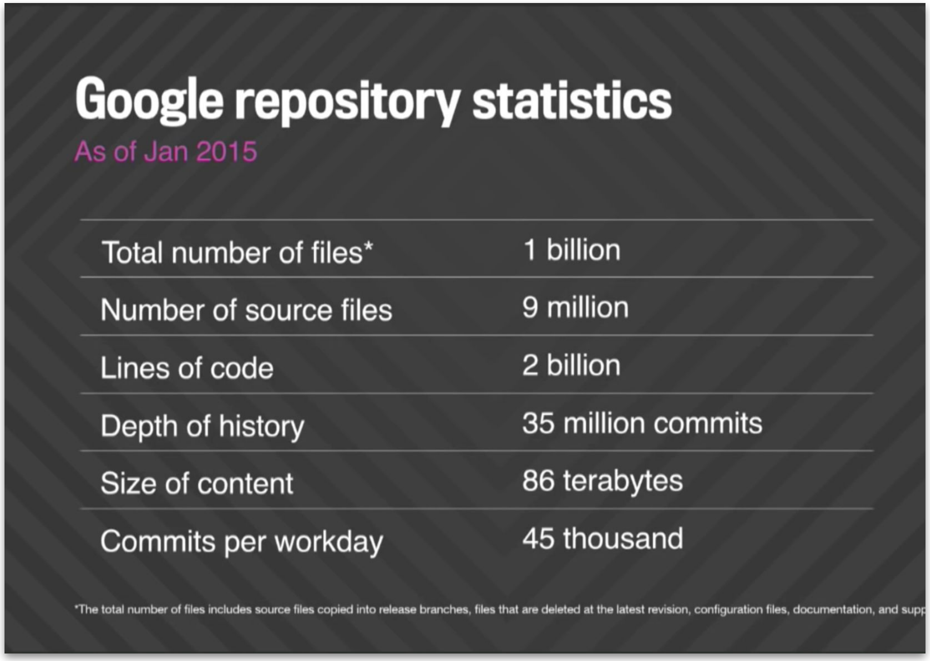
01 Google repository scale

02 Google systems and workflows

03 Advantages of a monolithic repository

04 Costs associated with this model

05 Conclusion

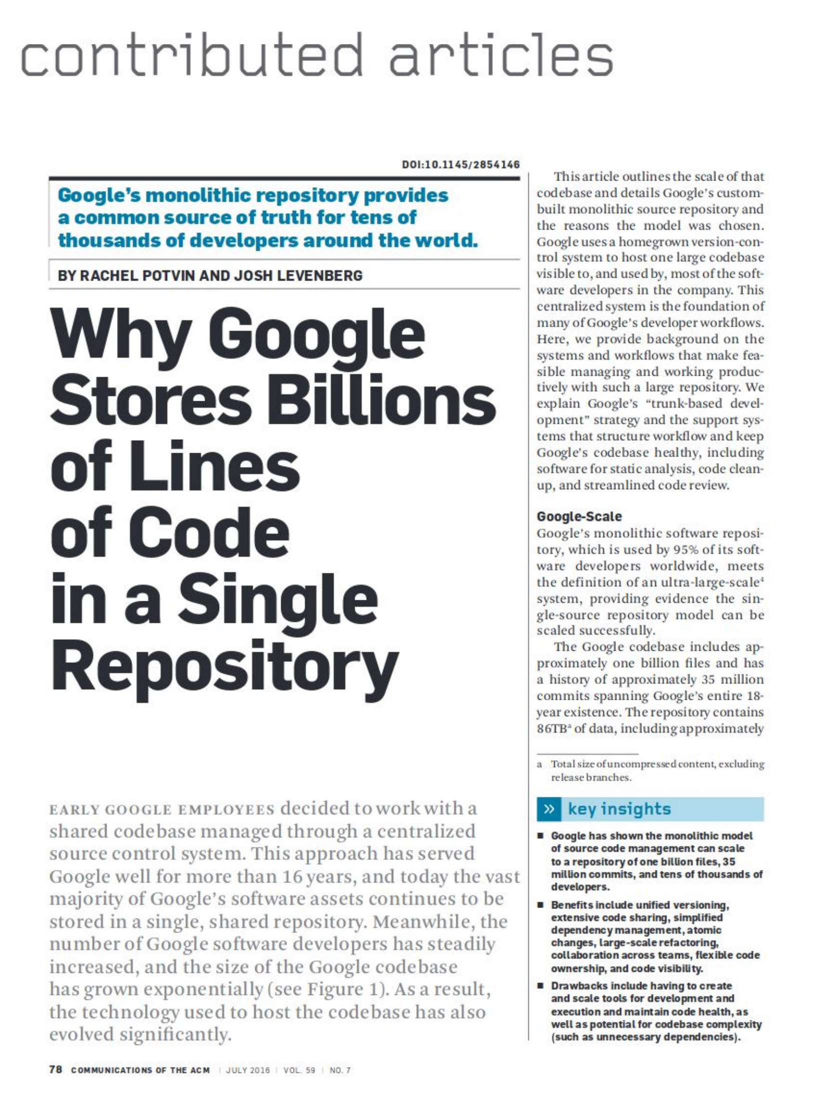


Google repository statistics

As of Jan 2015

Total number of files*	1 billion
Number of source files	9 million
Lines of code	2 billion
Depth of history	35 million commits
Size of content	86 terabytes
Commits per workday	45 thousand

*The total number of files includes source files copied into release branches, files that are deleted at the latest revision, configuration files, documentation, and supp



contributed articles

DOI:10.1145/2854146

Google's monolithic repository provides a common source of truth for tens of thousands of developers around the world.

BY RACHEL POTVIN AND JOSH LEVENBERG

Why Google Stores Billions of Lines of Code in a Single Repository

EARLY GOOGLE EMPLOYEES decided to work with a shared codebase managed through a centralized source control system. This approach has served Google well for more than 16 years, and today the vast majority of Google's software assets continues to be stored in a single, shared repository. Meanwhile, the number of Google software developers has steadily increased, and the size of the Google codebase has grown exponentially (see Figure 1). As a result, the technology used to host the codebase has also evolved significantly.

This article outlines the scale of that codebase and details Google's custom-built monolithic source repository and the reasons the model was chosen. Google uses a homegrown version-control system to host one large codebase visible to, and used by, most of the software developers in the company. This centralized system is the foundation of many of Google's developer workflows. Here, we provide background on the systems and workflows that make feasible managing and working productively with such a large repository. We explain Google's "trunk-based development" strategy and the support systems that structure workflow and keep Google's codebase healthy, including software for static analysis, code clean-up, and streamlined code review.

Google-Scale

Google's monolithic software repository, which is used by 95% of its software developers worldwide, meets the definition of an ultra-large-scale[4] system, providing evidence the single-source repository model can be scaled successfully.

The Google codebase includes approximately one billion files and has a history of approximately 35 million commits spanning Google's entire 18-year existence. The repository contains 86TB[a] of data, including approximately

[a] Total size of uncompressed content, excluding release branches.

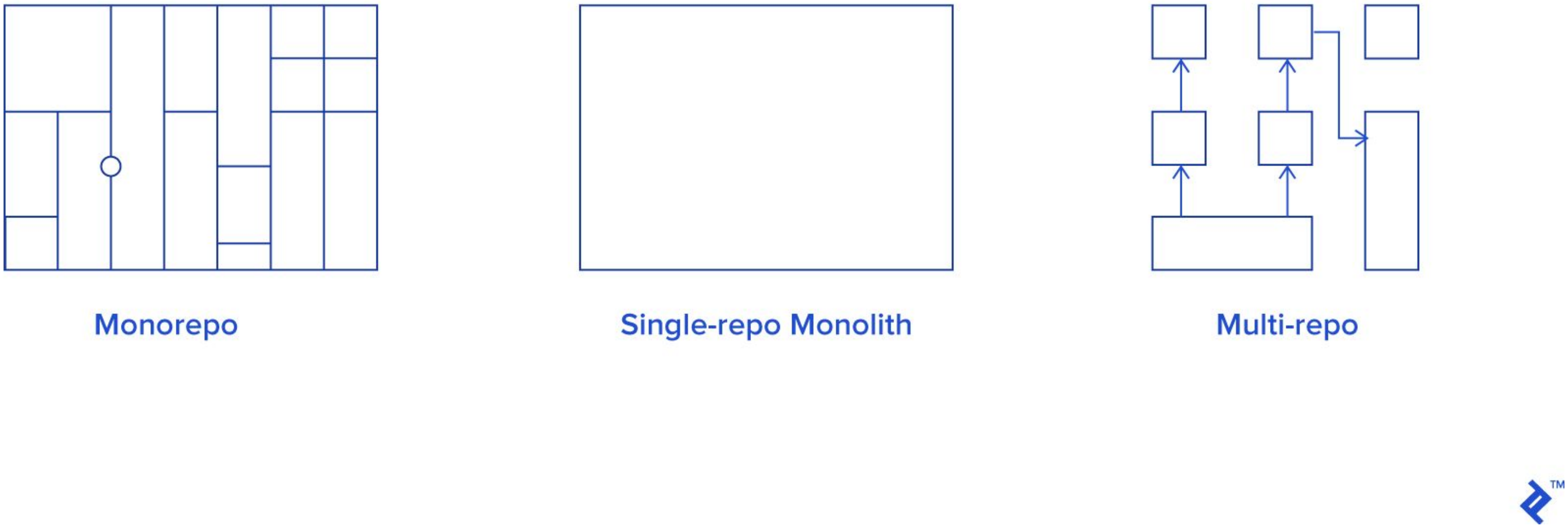
» key insights

- Google has shown the monolithic model of source code management can scale to a repository of one billion files, 35 million commits, and tens of thousands of developers.
- Benefits include unified versioning, extensive code sharing, simplified dependency management, atomic changes, large-scale refactoring, collaboration across teams, flexible code ownership, and code visibility.
- Drawbacks include having to create and scale tools for development and execution and maintain code health, as well as potential for codebase complexity (such as unnecessary dependencies).

78 COMMUNICATIONS OF THE ACM | JULY 2016 | VOL. 59 | NO. 7

R. Potvin and J. Levenberg, "The Motivation for a Monolithic Codebase: Why Google stores billions of lines of code in a single repository", in Communications of the ACM, vol. 59, no. 7, 2016.

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

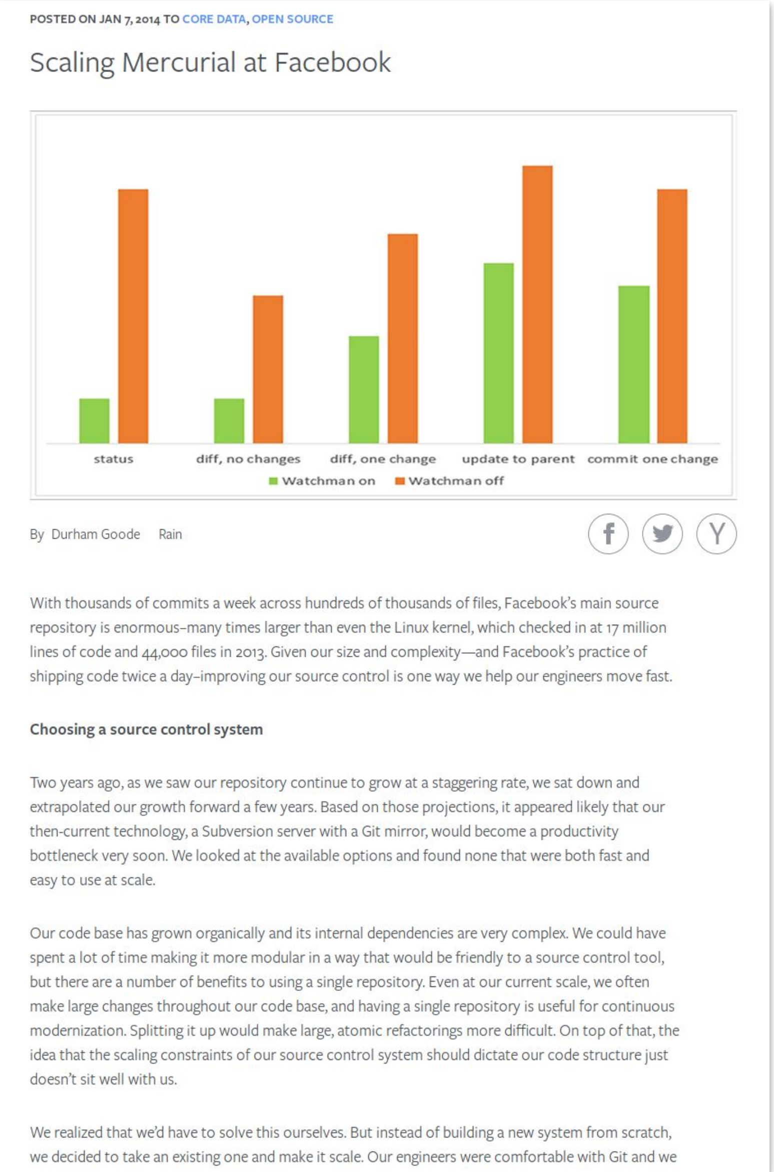
Scaling Mercurial at Facebook: Insights from the Other Side

Durham Goode, *Facebook*

Watchman

A file watching service

mercurial



POSTED ON JAN 7, 2014 TO CORE DATA, OPEN SOURCE

Scaling Mercurial at Facebook

By Durham Goode Rain

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.

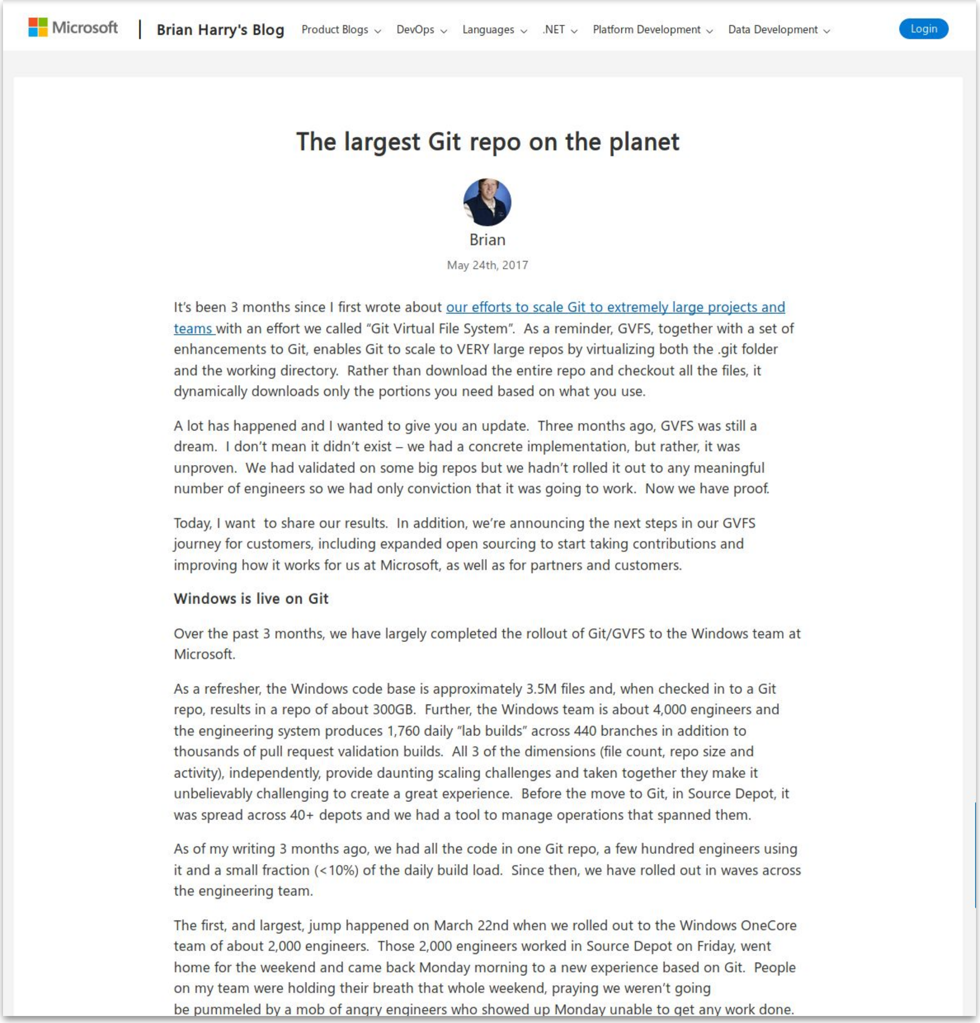
Choosing a source control system

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

Microsoft



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.

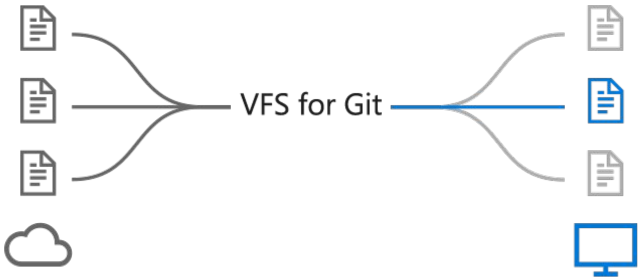
Windows is live on Git

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

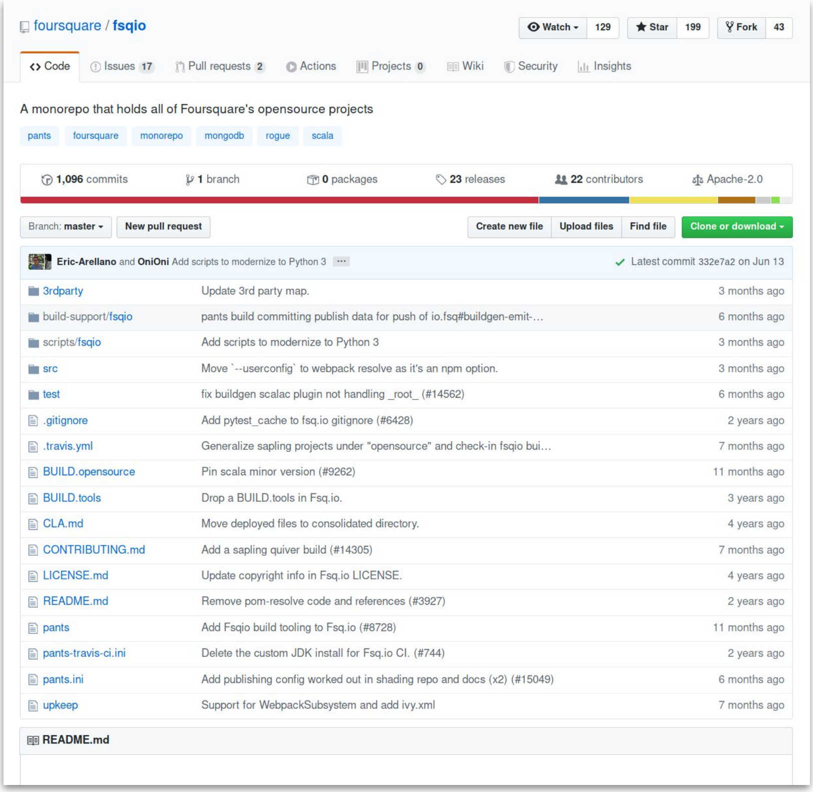
As a refresher, the Windows code base is approximately 3.5M files and, when checked in to a Git repo, results in a repo of about 300GB. Further, the Windows team is about 4,000 engineers and the engineering system produces 1,760 daily "lab builds" across 440 branches in addition to thousands of pull 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 be pummeled by a mob of angry engineers who showed up Monday unable to get any work done.



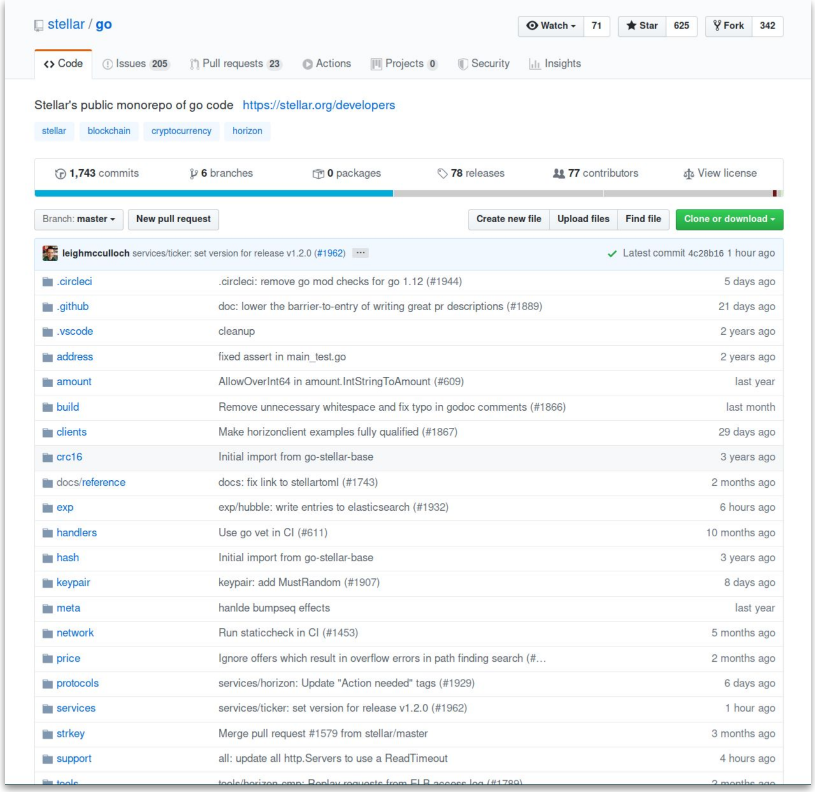
Monorepos are also used by open source projects



A monorepo that holds all of Foursquare's opensource projects

Eric-Arellano and OniOni Add scripts to modernize to Python 3 ··· ✓ Latest commit 332e7a2 on Jun 13

3rdparty	Update 3rd party map.	3 months ago
build-support/fsqio	pants build committing publish data for push of io.fsq#buildgen-emit-…	6 months ago
scripts/fsqio	Add scripts to modernize to Python 3	3 months ago
src	Move `--userconfig` to webpack resolve as it's an npm option.	3 months ago
test	fix buildgen scalac plugin not handling _root_ (#14562)	6 months ago
.gitignore	Add pytest_cache to fsq.io gitignore (#6428)	2 years ago
.travis.yml	Generalize sapling projects under "opensource" and check-in fsqio bui…	7 months ago
BUILD.opensource	Pin scala minor version (#9262)	11 months ago
BUILD.tools	Drop a BUILD.tools in Fsq.io.	3 years ago
CLA.md	Move deployed files to consolidated directory.	4 years ago
CONTRIBUTING.md	Add a sapling quiver build (#14305)	7 months ago
LICENSE.md	Update copyright info in Fsq.io LICENSE.	4 years ago
README.md	Remove pom-resolve code and references (#3927)	2 years ago
pants	Add Fsqio build tooling to Fsq.io (#8728)	11 months ago
pants-travis-ci.ini	Delete the custom JDK install for Fsq.io CI. (#744)	2 years ago
pants.ini	Add publishing config worked out in shading repo and docs (x2) (#15049)	6 months ago
upkeep	Support for WebpackSubsystem and add ivy.xml	7 months ago

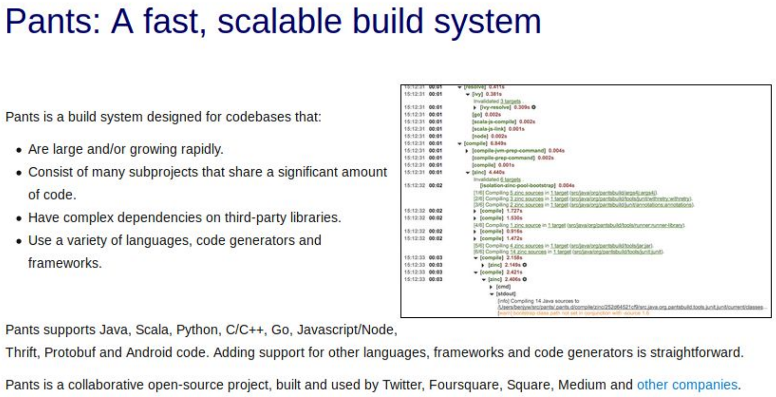


Stellar's public monorepo of go code https://stellar.org/developers

leighmcculloch services/ticker: set version for release v1.2.0 (#1962) ··· ✓ Latest commit 4c28b16 1 hour ago

.circleci	.circleci: remove go mod checks for go 1.12 (#1944)	5 days ago
.github	doc: lower the barrier-to-entry of writing great pr descriptions (#1889)	21 days ago
.vscode	cleanup	2 years ago
address	fixed assert in main_test.go	2 years ago
amount	AllowOverInt64 in amount.IntStringToAmount (#609)	last year
build	Remove unnecessary whitespace and fix typo in godoc comments (#1866)	last month
clients	Make horizonclient examples fully qualified (#1867)	29 days ago
crc16	Initial import from go-stellar-base	3 years ago
docs/reference	docs: fix link to stellartoml (#1743)	2 months ago
exp	exp/hubble: write entries to elasticsearch (#1932)	6 hours ago
handlers	Use go vet in CI (#611)	10 months ago
hash	Initial import from go-stellar-base	3 years ago
keypair	keypair: add MustRandom (#1907)	8 days ago
meta	hanlde bumpseq effects	last year
network	Run staticcheck in CI (#1453)	5 months ago
price	Ignore offers which result in overflow errors in path finding search (#…	2 months ago
protocols	services/horizon: Update "Action needed" tags (#1929)	6 days ago
services	services/ticker: set version for release v1.2.0 (#1962)	1 hour ago
strkey	Merge pull request #1579 from stellar/master	3 months ago
support	all: update all http.Servers to use a ReadTimeout	4 hours ago
tools	tools/horizon-cmp: Replay requests from ELB access log (#1789)	2 months ago

Monorepos tend to use a common build system



Pants: A fast, scalable build system

Pants is a build system designed for codebases that:

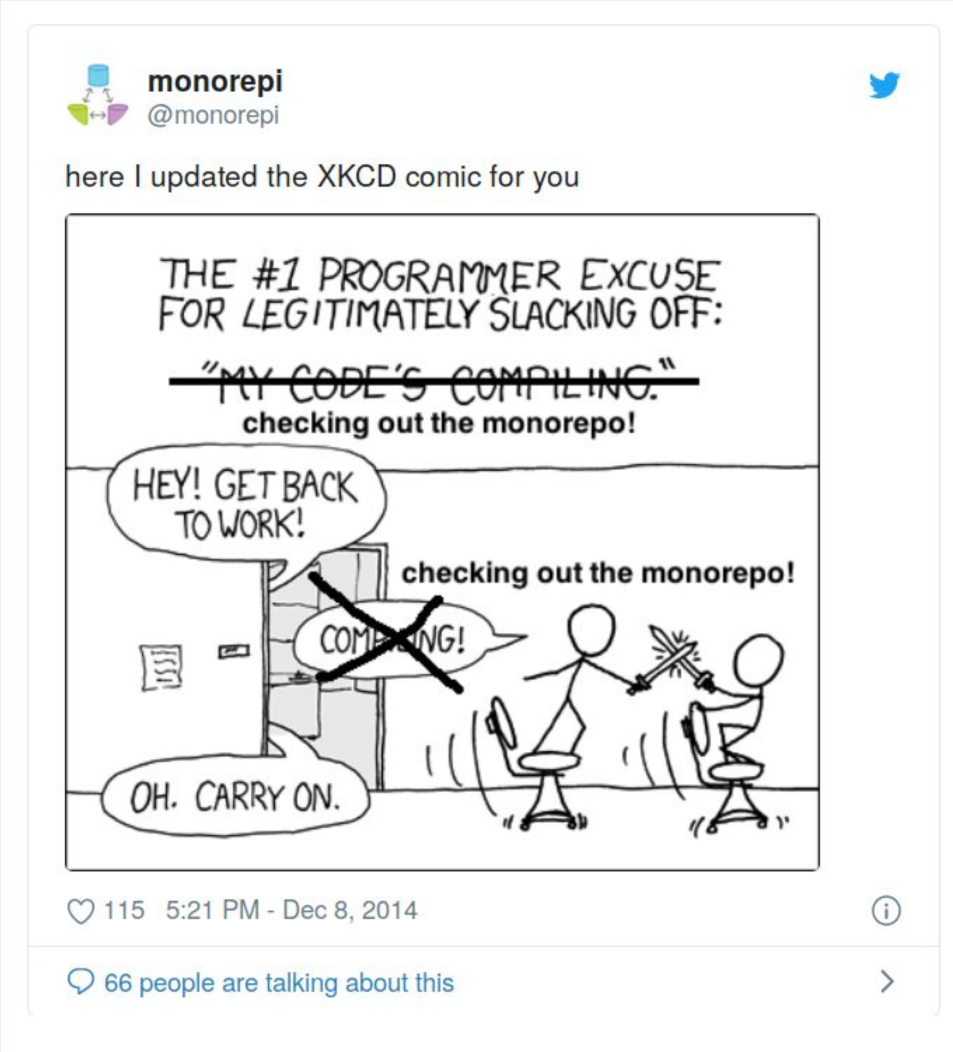
- Are large and/or growing rapidly.
- Consist of many subprojects that share a significant amount of code.
- Have complex dependencies on third-party libraries.
- Use a variety of languages, code generators and frameworks.

```
15:12:31 00:01    ▾ [resolve] 0.411s
15:12:31 00:01      ▾ [ivy] 0.381s
                       Invalidated 3 targets.
15:12:31 00:01         ▸ [ivy-resolve] 0.309s
15:12:31 00:01        [go] 0.002s
15:12:31 00:01        [scala-js-compile] 0.002s
15:12:31 00:01        [scala-js-link] 0.001s
15:12:31 00:01        [node] 0.002s
15:12:31 00:01    ▾ [compile] 6.849s
15:12:31 00:01      ▸ [compile-jvm-prep-command] 0.004s
15:12:31 00:01        [compile-prep-command] 0.002s
15:12:31 00:01        [compile] 0.001s
15:12:31 00:01      ▾ [zinc] 4.440s
                       Invalidated 6 targets.
15:12:32 00:02           [isolation-zinc-pool-bootstrap] 0.004s
                       [1/6] Compiling 5 zinc sources in 1 target (src/java/org/pantsbuild/args4j:args4j).
                       [2/6] Compiling 3 zinc sources in 1 target (src/java/org/pantsbuild/tools/junit/withretry:withretry).
                       [3/6] Compiling 2 zinc sources in 1 target (src/java/org/pantsbuild/junit/annotations:annotations).
15:12:32 00:02         ▸ [compile] 1.727s
15:12:32 00:02         ▸ [compile] 1.530s
                       [4/6] Compiling 1 zinc source in 1 target (src/java/org/pantsbuild/tools/runner:runner-library).
15:12:32 00:02         ▸ [compile] 0.916s
15:12:32 00:02         ▸ [compile] 1.472s
                       [5/6] Compiling 4 zinc sources in 1 target (src/java/org/pantsbuild/tools/jar:jar).
                       [6/6] Compiling 14 zinc sources in 1 target (src/java/org/pantsbuild/tools/junit:junit).
15:12:33 00:03         ▾ [compile] 2.158s
15:12:33 00:03           ▸ [zinc] 2.149s
15:12:33 00:03         ▾ [compile] 2.421s
15:12:33 00:03           ▾ [zinc] 2.406s
                             ▸ [cmd]
                             ▾ [stdout]
                                [info] Compiling 14 Java sources to
                                /Users/benjyw/src/pants/.pants.d/compile/zinc/252d64521cf9/src.java.org.pantsbuild.tools.junit.junit/current/classes...
                                [warn] bootstrap class path not set in conjunction with -source 1.6
```

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?



monorepi
@monorepi

here I updated the XKCD comic for you

THE #1 PROGRAMMER EXCUSE FOR LEGITIMATELY SLACKING OFF:
~~"MY CODE'S COMPILING."~~
checking out the monorepo!
HEY! GET BACK TO WORK!
checking out the monorepo!
~~COMPILING!~~
OH. CARRY ON.

115 5:21 PM - Dec 8, 2014

66 people are talking about this

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!

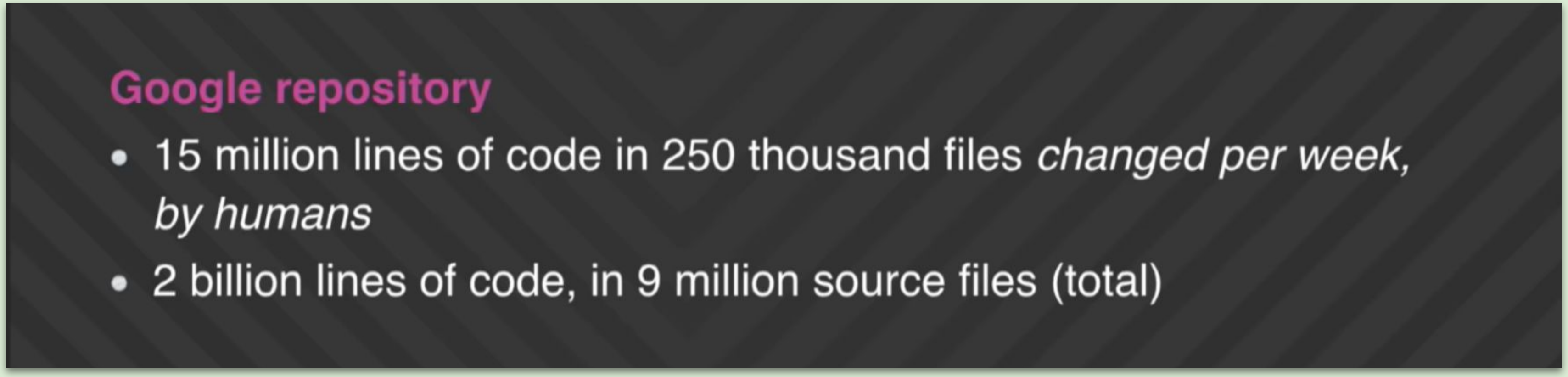
Drawbacks of Monorepos

- 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

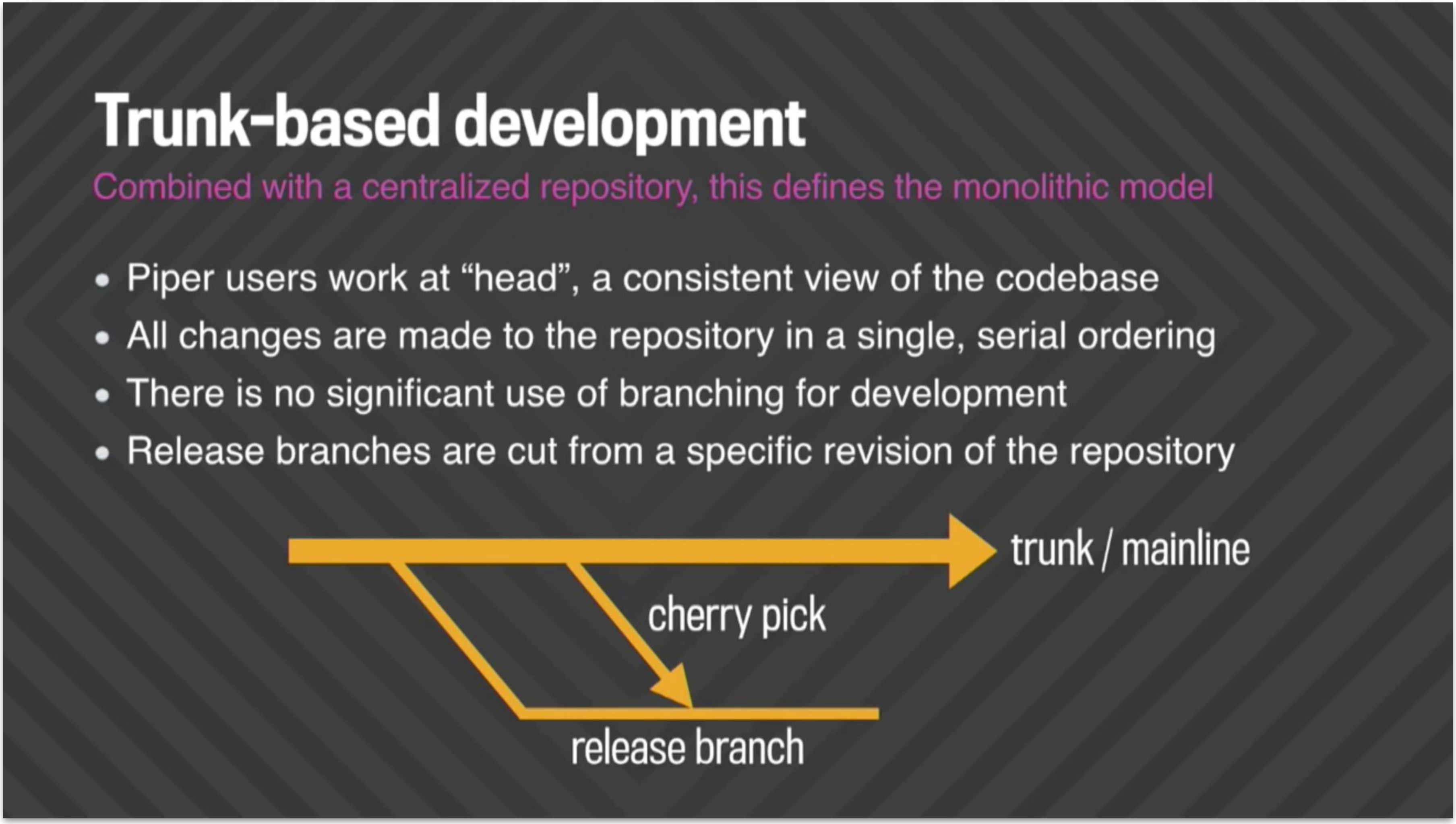
- 15 million lines of code in 250 thousand files *changed per week, by humans*
- 2 billion lines of code, in 9 million source files (total)

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

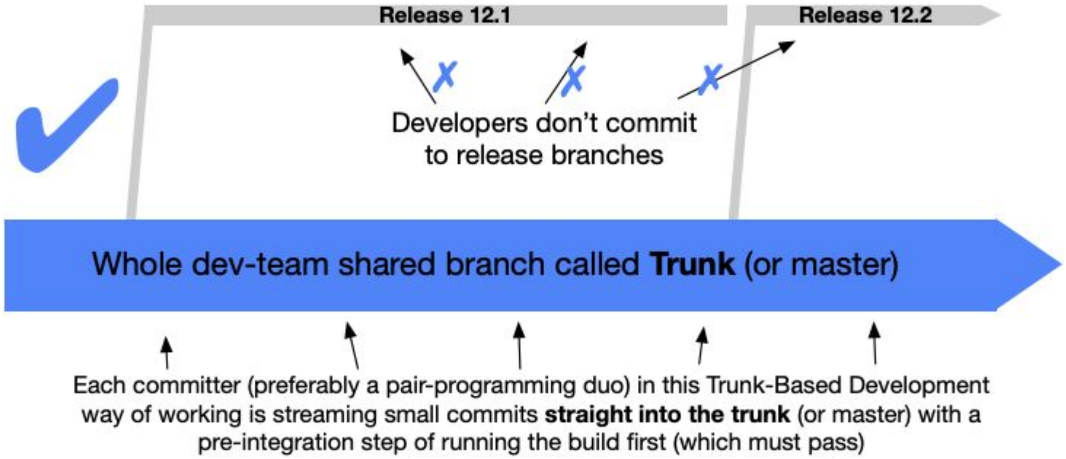


Trunk-based development

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



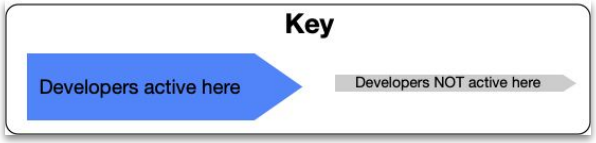
Release 12.1

Release 12.2

Developers don't commit to release branches

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

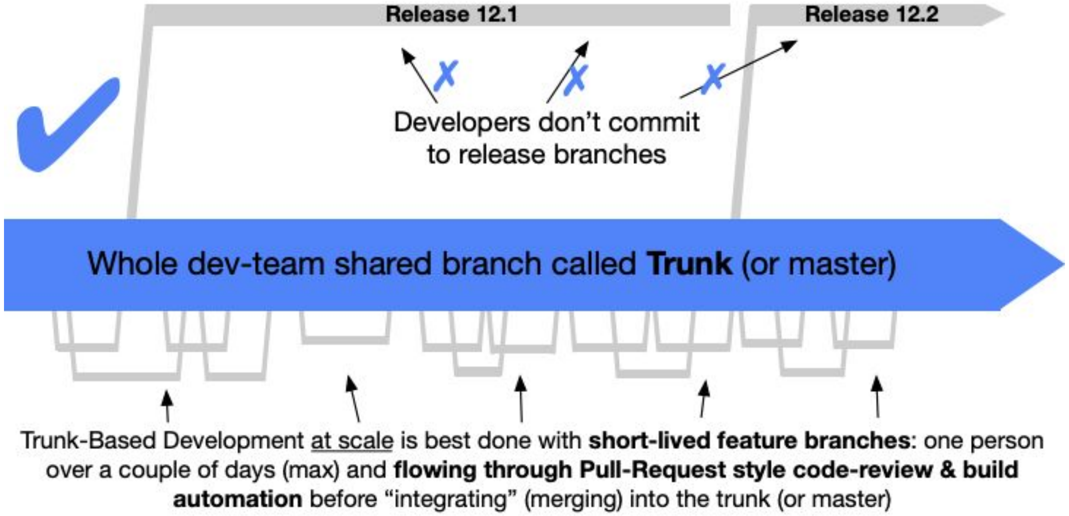
Each committer (preferably a pair-programming duo) in this Trunk-Based Development way of working is streaming small commits **straight into the trunk** (or master) with a pre-integration step of running the build first (which must pass)



Key

Developers active here

Developers NOT active here



Release 12.1

Release 12.2

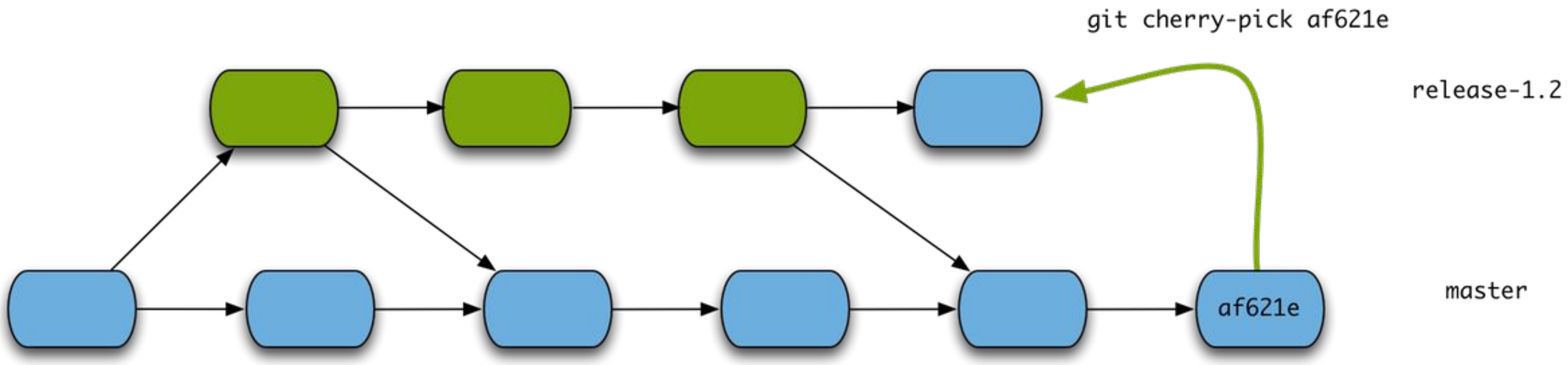
Developers don't commit to release branches

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)

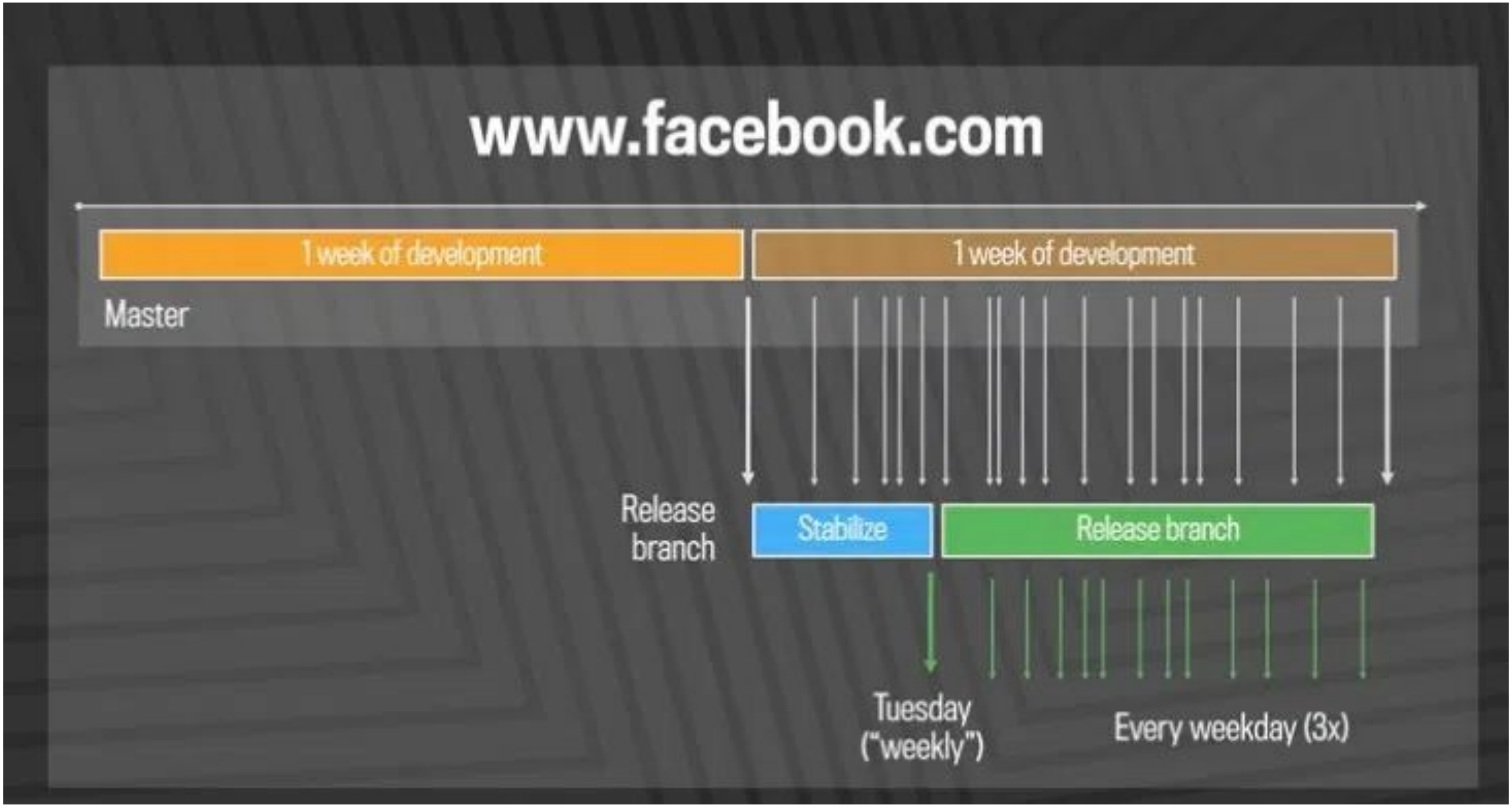
https://trunkbaseddevelopment.com

Cherrypicking



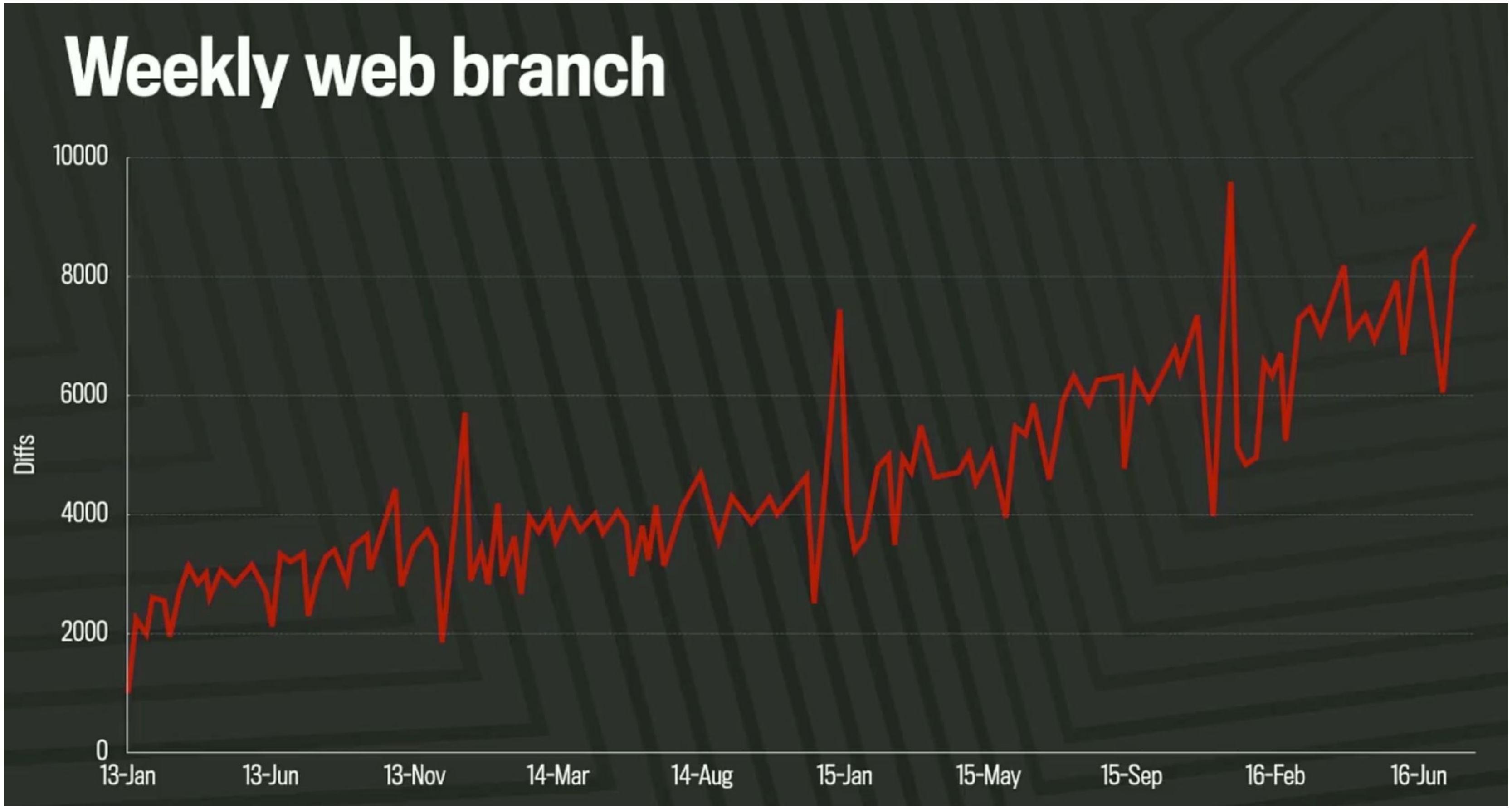
https://www.atlassian.com/blog/git/the-essence-of-branch-based-workflows

Fresh release branch every week

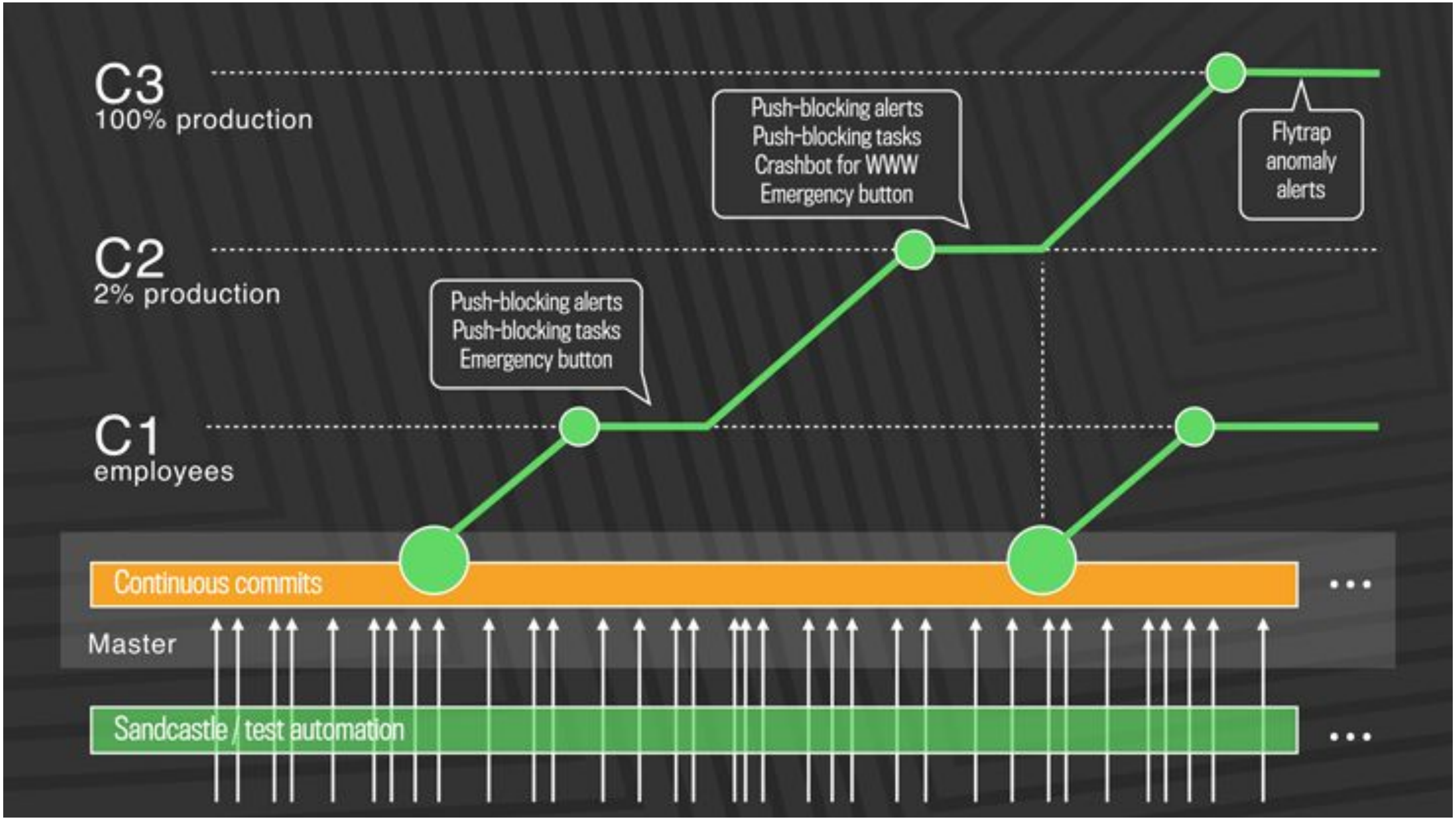


https://engineering.fb.com/web/rapid-release-at-massive-scale/

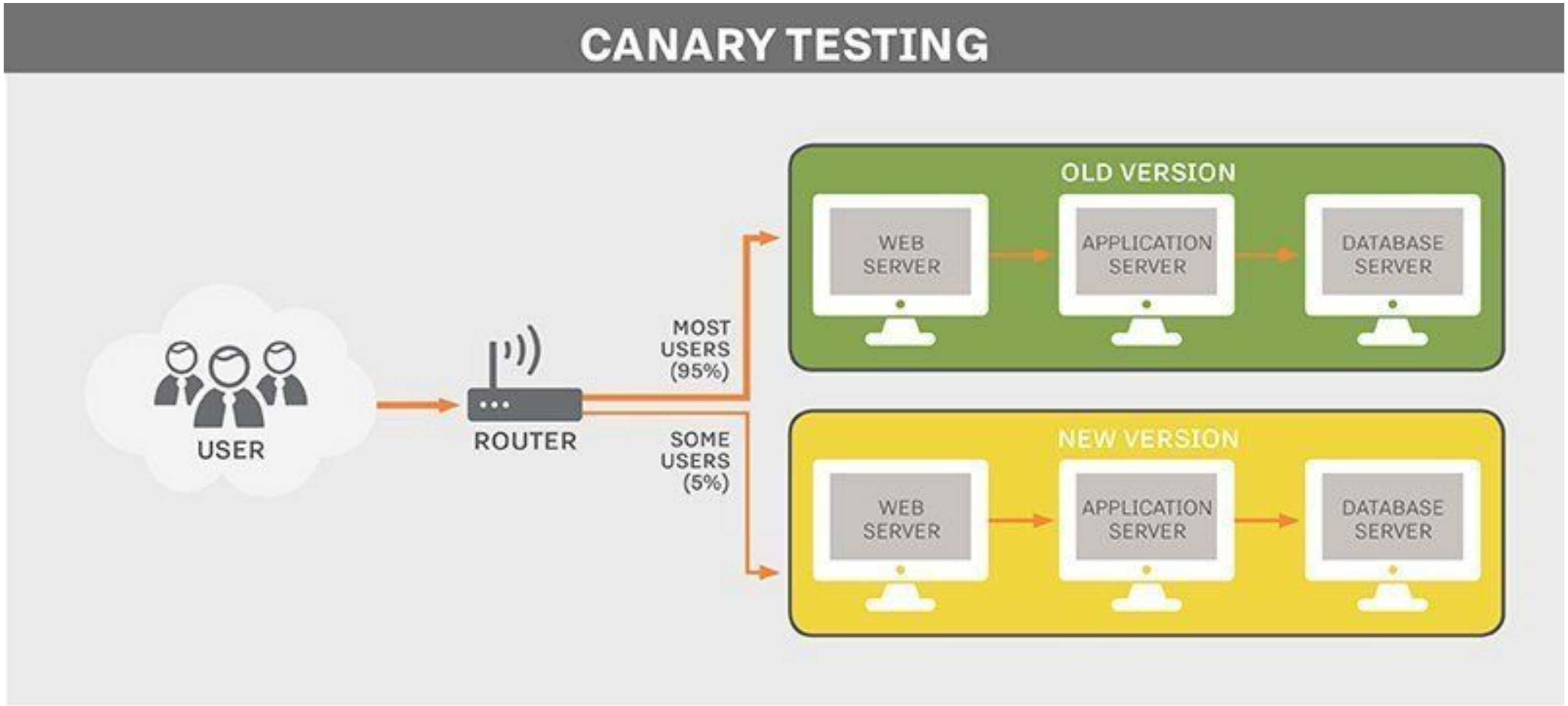
The number of commits in a branch cut became unsustainable



Quasi-continuous push from master (1,000+ devs, 1,000 diffs/day); 10 pushes/day

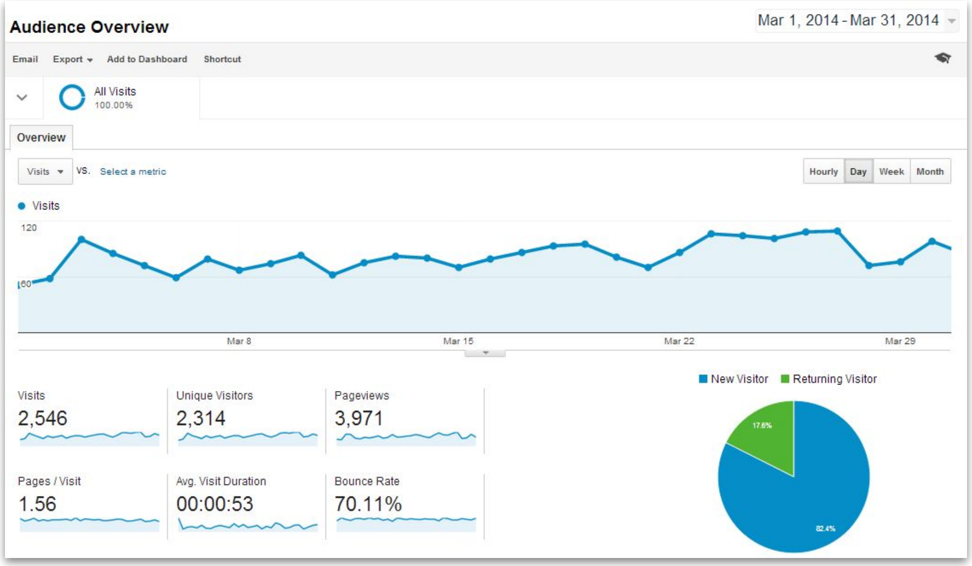


Principle: Every feature is an experiment


CANARY 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*



Audience Overview

Mar 1, 2014 - Mar 31, 2014

Email Export Add to Dashboard Shortcut

All Visits 100.00%

Overview

Visits vs. Select a metric

Hourly Day Week Month

Visits 2,546

Unique Visitors 2,314

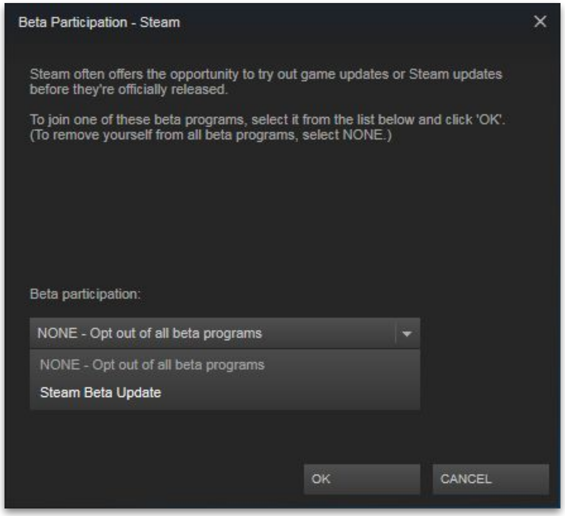
Pageviews 3,971

Pages / Visit 1.56

Avg. Visit Duration 00:00:53

Bounce Rate 70.11%

Aside: Opt-In Beta



Steam often offers the opportunity to try out game updates or Steam updates before they're officially released.

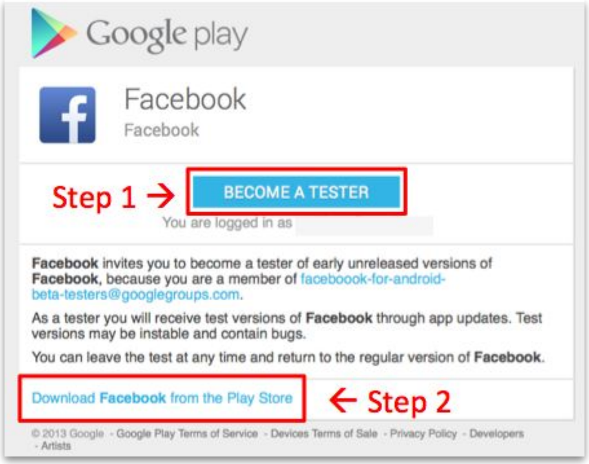
To join one of these beta programs, select it from the list below and click 'OK'. (To remove yourself from all beta programs, select NONE.)

Beta participation:

NONE - Opt out of all beta programs

NONE - Opt out of all beta programs

Steam Beta Update



Facebook

Facebook

Step 1 → BECOME A TESTER

You are logged in as [redacted]

Facebook invites you to become a tester of early unreleased versions of **Facebook**, because you are a member of facebooook-for-android-beta-testers@googlegroups.com.

As a tester you will receive test versions of **Facebook** through app updates. Test versions may be instable and contain bugs.

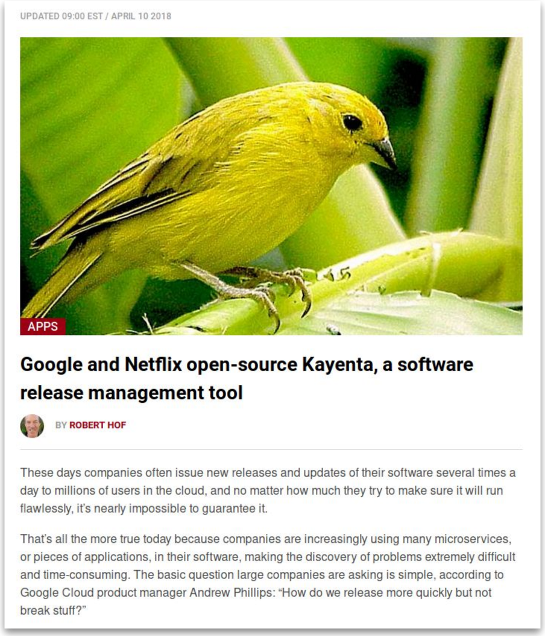
You can leave the test at any time and return to the regular version of **Facebook**.

Download **Facebook** from the Play Store ← Step 2

© 2013 Google - Google Play Terms of Service - Devices Terms of Sale - Privacy Policy - Developers - Artists

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.



UPDATED 09:00 EST / APRIL 10 2018

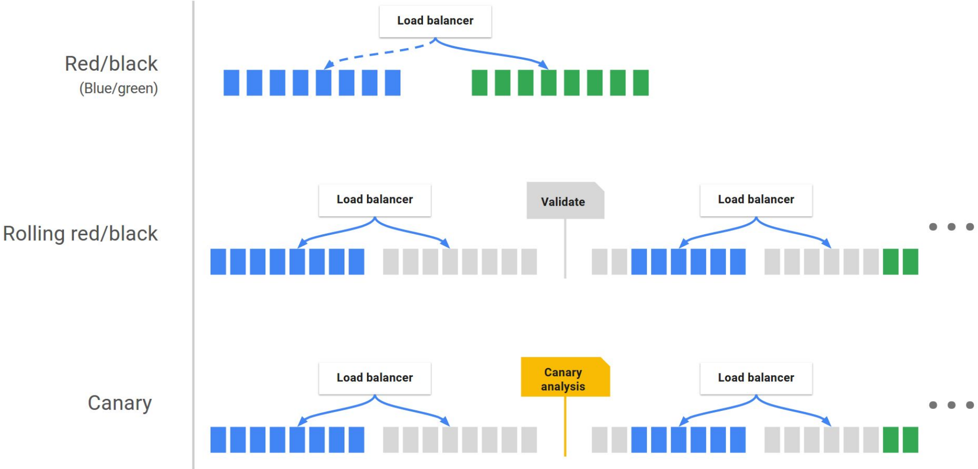
APPS

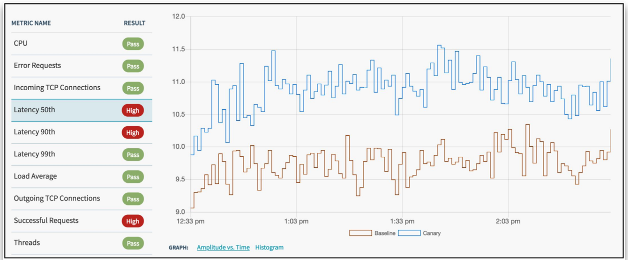
Google and Netflix open-source Kayenta, a software release management tool

BY ROBERT HOF

These days companies often issue new releases and updates of their software several times a day to millions of users in the cloud, and no matter how much they try to make sure it will run flawlessly, it's nearly impossible to guarantee it.

That's all the more true today because companies are increasingly using many microservices, or pieces of applications, in their software, making the discovery of problems extremely difficult and time-consuming. The basic question large companies are asking is simple, according to Google Cloud product manager Andrew Phillips: "How do we release more quickly but not break stuff?"

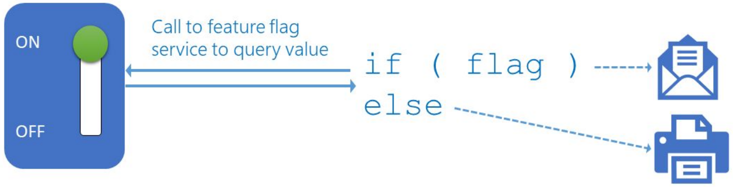


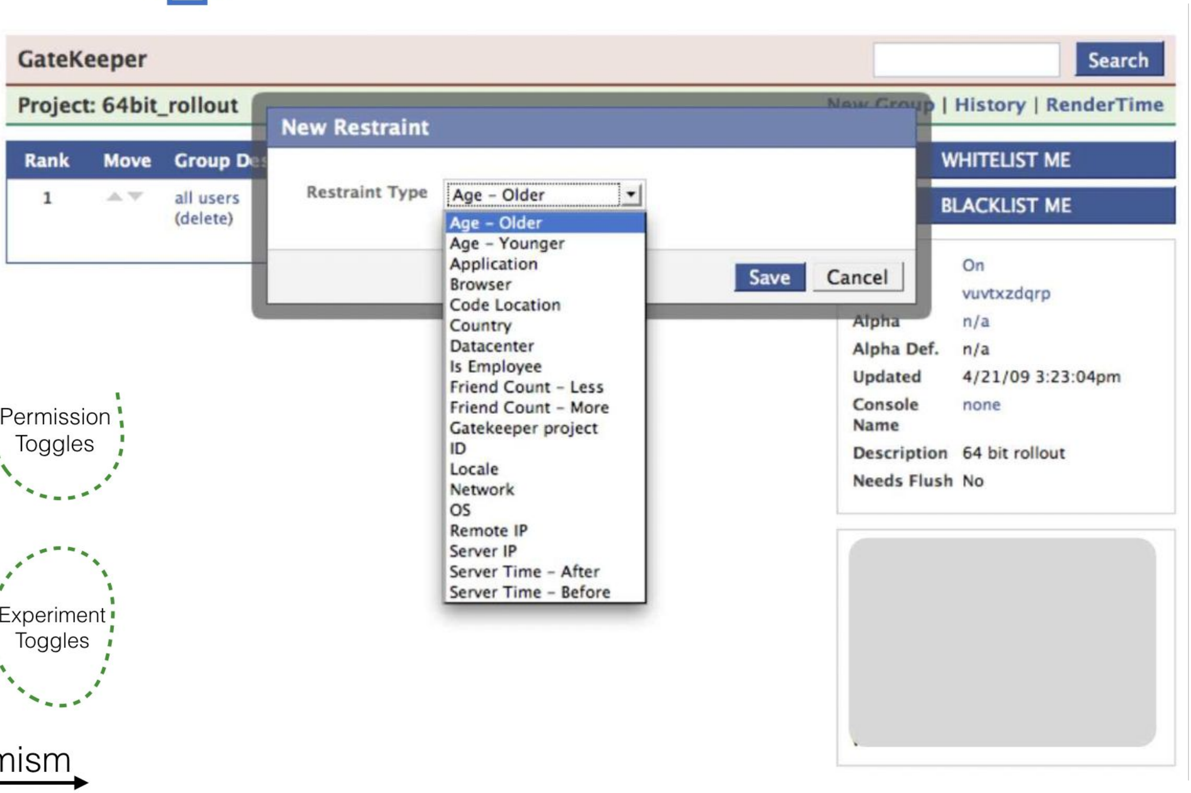


METRIC NAME	RESULT
CPU	Pass
Error Requests	Pass
Incoming TCP Connections	Pass
Latency 50th	High
Latency 90th	High
Latency 99th	Pass
Load Average	Pass
Outgoing TCP Connections	Pass
Successful Requests	High
Threads	Pass

https://medium.com/netflix-techblog/automated-canary-analysis-at-netflix-with-kayenta-3260bc7acc69
https://octopus.com/blog/blue-green-red-black
https://siliconangle.com/2018/04/10/google-netflix-open-source-kayenta-software-release-management-tool/

Control deployments at run-time using feature flags





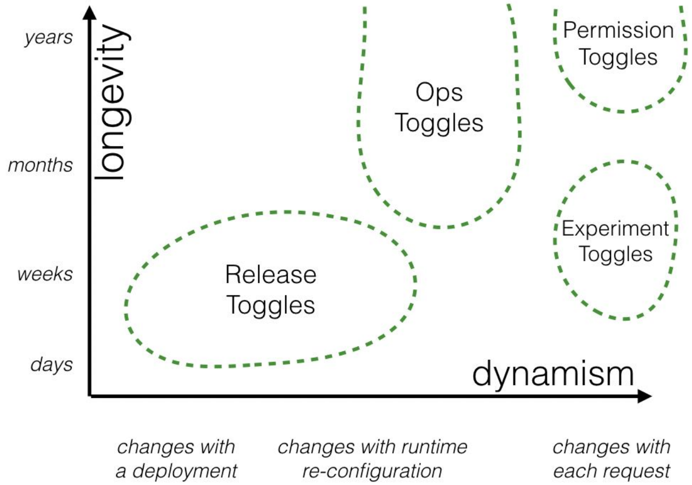
Project: 64bit_rollout

Rank	Move	Group Des
1	▲▼	all users (delete)

New Restraint

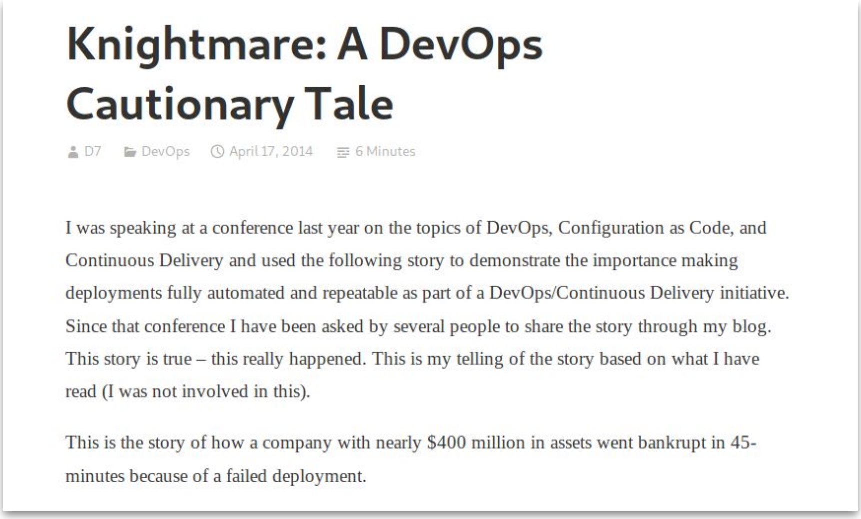
Restraint Type: Age – Older

	On
	vuvtxzdqrp
Alpha	n/a
Alpha Def.	n/a
Updated	4/21/09 3:23:04pm
Console Name	none
Description	64 bit rollout
Needs Flush	No



https://martinfowler.com/articles/feature-toggles.html
https://docs.microsoft.com/en-us/azure/devops/migrate/phase-features-with-feature-flags?view=azure-devops

Warning! Feature flags can be dangerous



Knightmare: A DevOps Cautionary Tale

D7 DevOps April 17, 2014 6 Minutes

I was speaking at a conference last year on the topics of DevOps, Configuration as Code, and Continuous Delivery and used the following story to demonstrate the importance making deployments fully automated and repeatable as part of a DevOps/Continuous Delivery initiative. Since that conference I have been asked by several people to share the story through my blog. This story is true – this really happened. This is my telling of the story based on what I have read (I was not involved in this).

This is the story of how a company with nearly $400 million in assets went bankrupt in 45-minutes because of a failed deployment.

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.**

https://dougseven.com/2014/04/17/knightmare-a-devops-cautionary-tale/

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