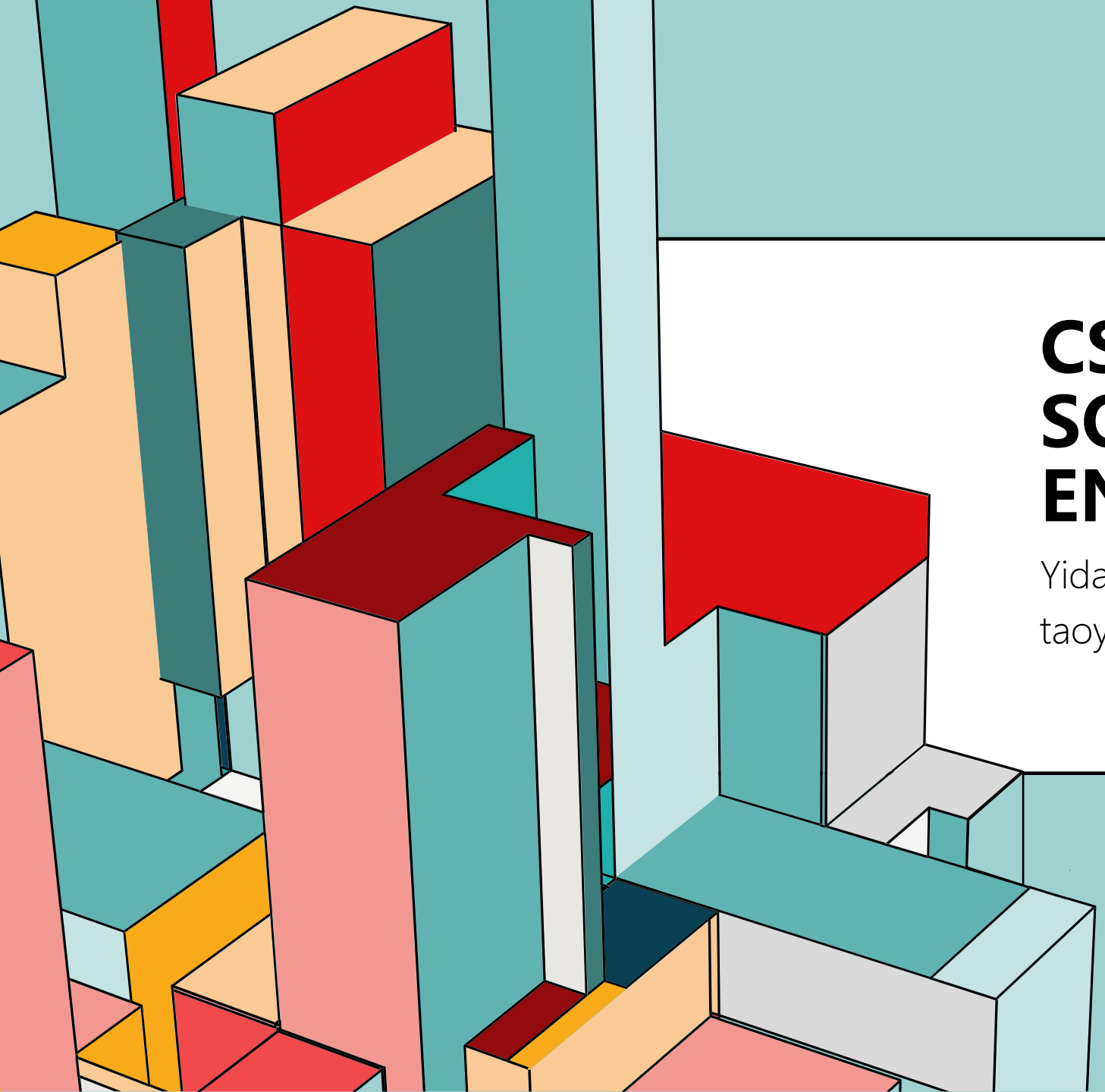
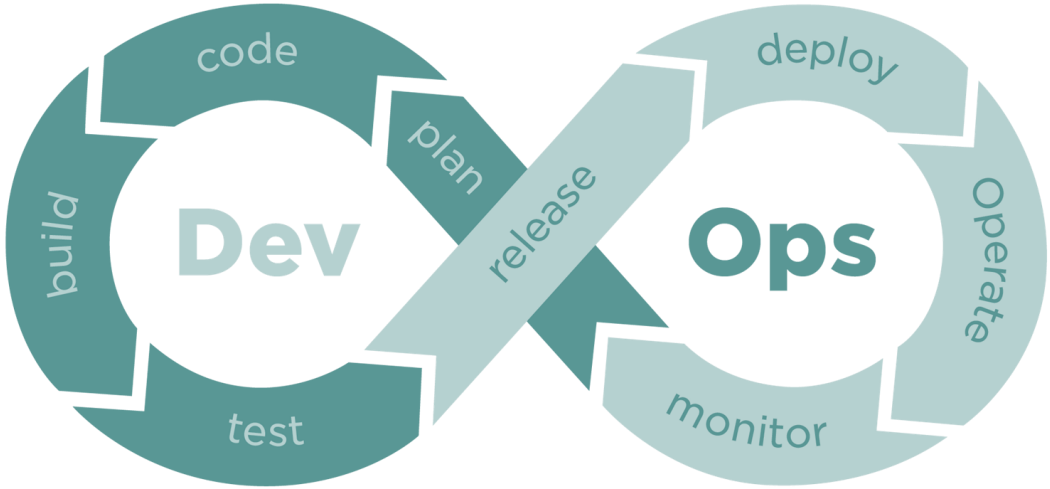


CS304 SOFTWARE ENGINEERING

Yida Tao

taoyd@sustech.edu.cn

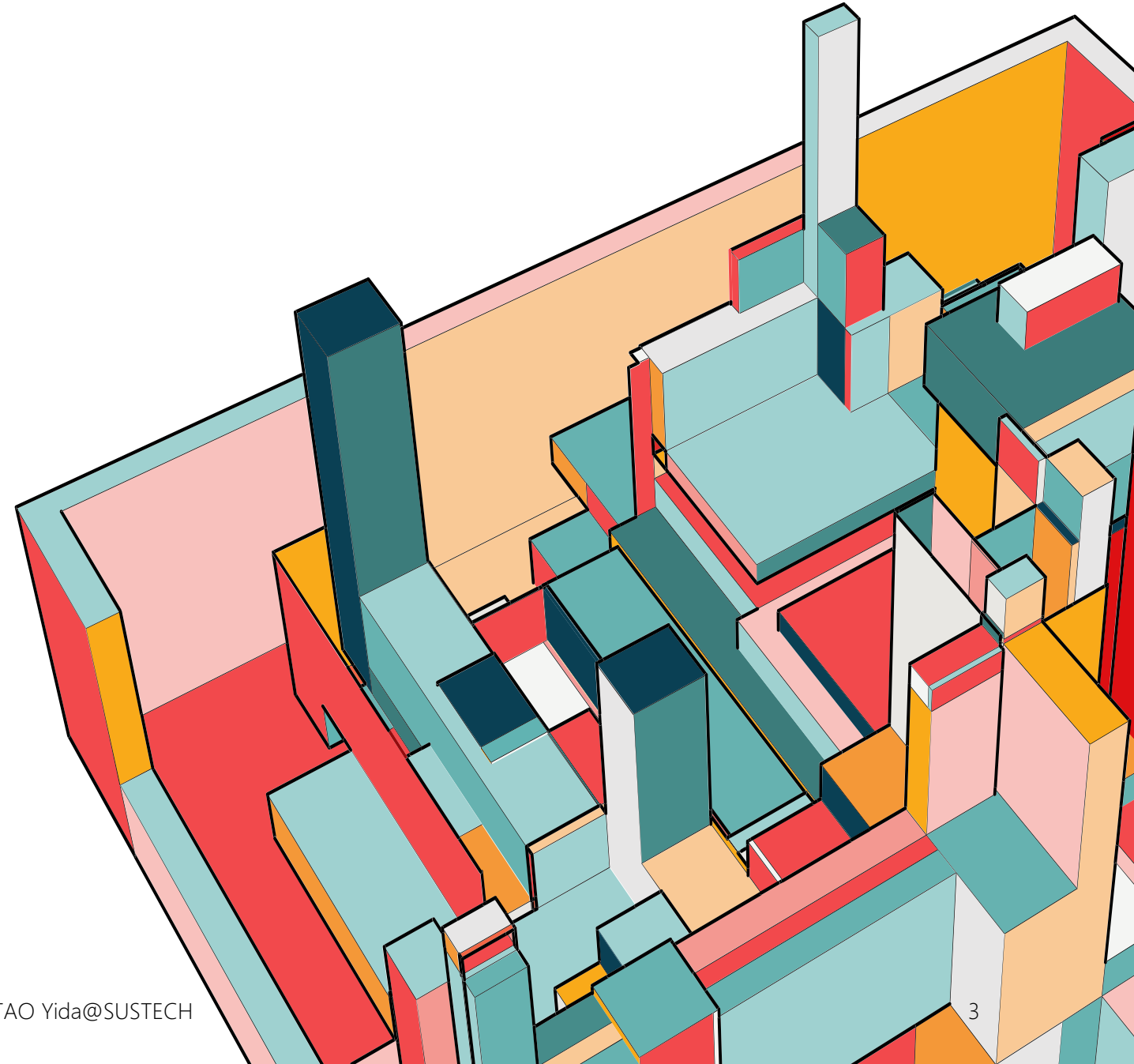
WHERE ARE WE NOW?



We want to automate the commit + build + test process, so that every new update can be pushed to release **timely** and **confidently**

LECTURE 11

- Continuous Integration
- CI/CD Pipeline
- Deployment Strategy
- Branching Strategy
- Case Study



HISTORY

- In 1980s, Microsoft's products become too complex to be handled by individuals
- Microsoft **continually synchronize** what people are doing and **periodically stabilize** the product in increments as a project proceeds, rather than once at the end of a project.
- Microsoft people refer to their techniques as the "daily build", "nightly build", "sync-and-stabilize", or "zero-defect" process: the practice of **completing a software build of the latest version of a program on a daily basis**.

https://dl.acm.org/doi/pdf/10.1145/255656.255698

HISTORY

- In 1990s, the Chrysler Comprehensive Compensation System (C3 Project) was compensation system capable of supporting 90, 000 employees. However, the project encountered significant difficulties and delays.

- In 1996, Kent Beck and his team took over the project, where they discovered that integrating and running the project code was a challenging and time-consuming task, often taking one to two weeks.

- The team gradually increased integration frequencies and found that this practice significantly reduced the time for each integration. Moreover, software issues could be detected earlier and resolved more easily due to this improved integration practice.

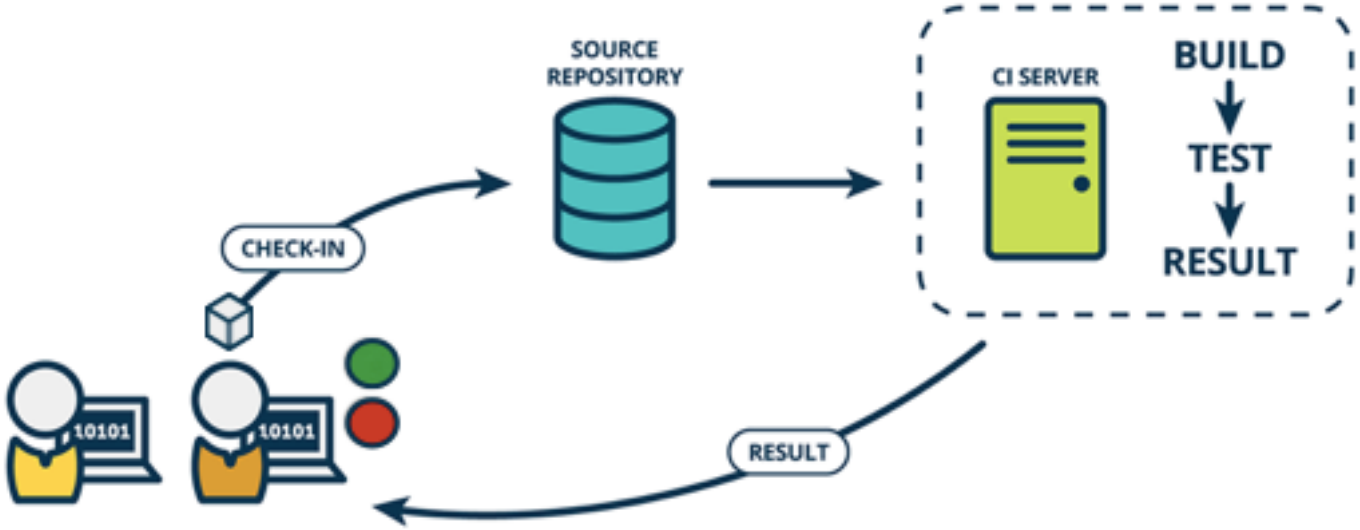
CONTINUOUS INTEGRATION (CI)

- Developers continually commit changes in small increments (at least daily, or even several times a day)
- Each change is automatically built and tested by the CI server before it is merged into the product.
- If any issues are discovered during this phase, the CI server blocks the code from merging and alerts the team so they can quickly fix any errors.

The fundamental goal of CI is to automatically catch problematic changes as early as possible.

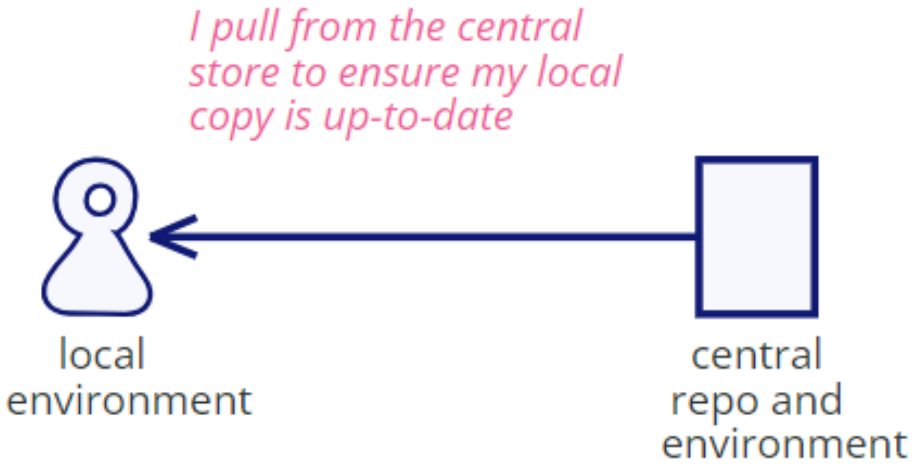
CI PROCESS - PREPARATIONS

- Version control
  - git and GitHub
- Automated build
  - Maven, buildfile as code
- Automated quality control
  - Linters, JUnit tests
- Automated doc generation
  - JavaDoc
- Team consensus
  - Frequency of build
  - Quality standards



https://techbeacon.com/app-dev-testing/are-you-really-doing-continuous-integration-heres-how-tell

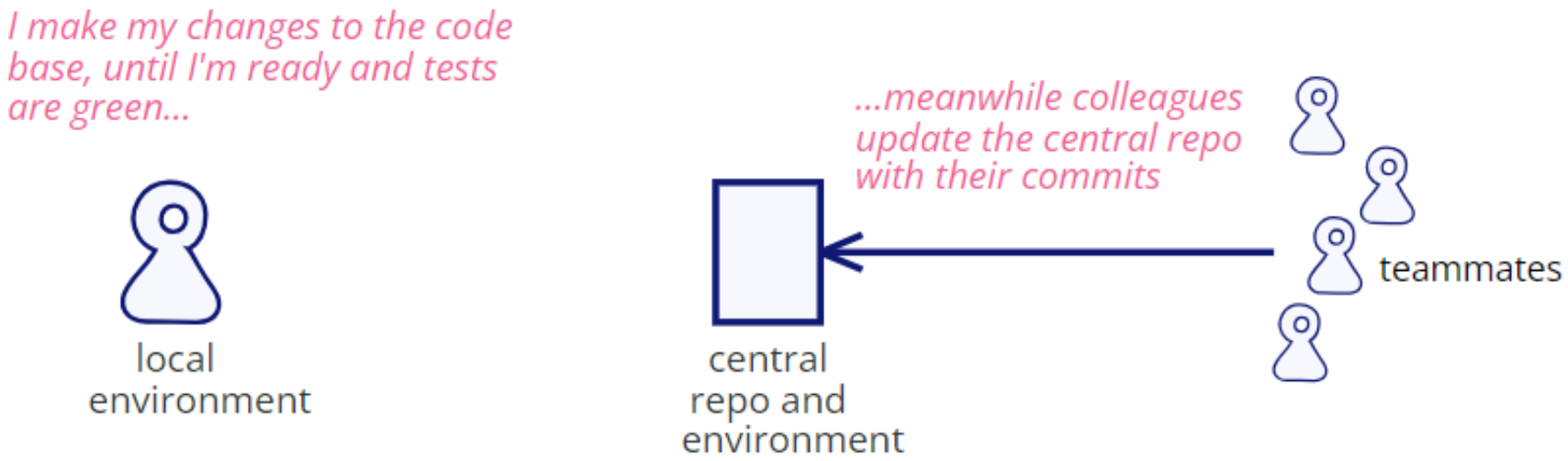
CI PROCESS - COMMIT PHASE



STEP 1

https://martinfowler.com/articles/continuousIntegration.html

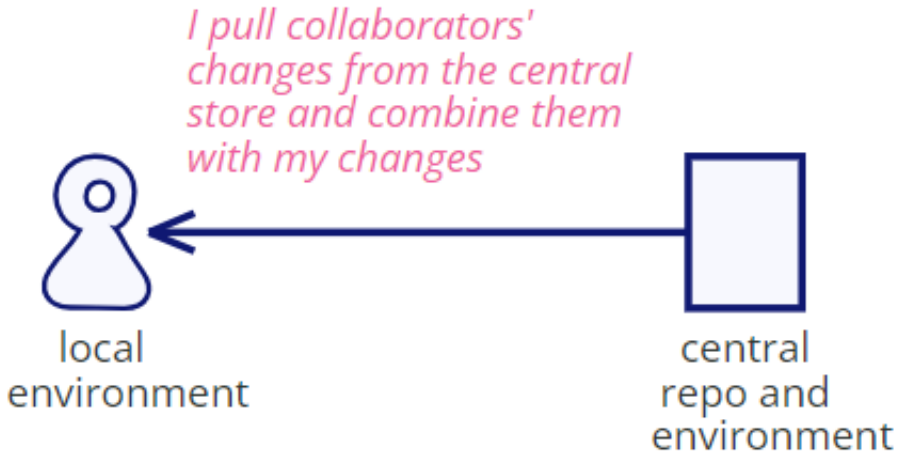
CI PROCESS - COMMIT PHASE



STEP 2

https://martinfowler.com/articles/continuousIntegration.html

CI PROCESS - COMMIT PHASE



STEP 3

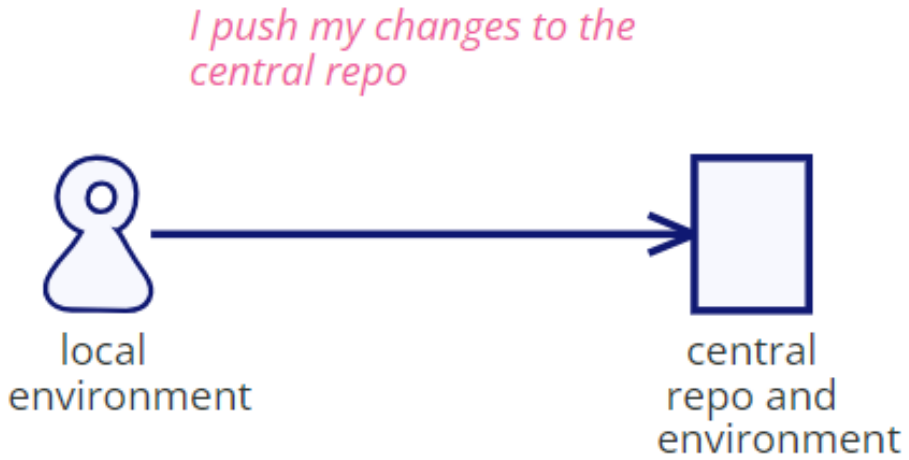
https://martinfowler.com/articles/continuousIntegration.html

CI PROCESS - COMMIT PHASE

STEP 4

https://martinfowler.com/articles/continuousIntegration.html

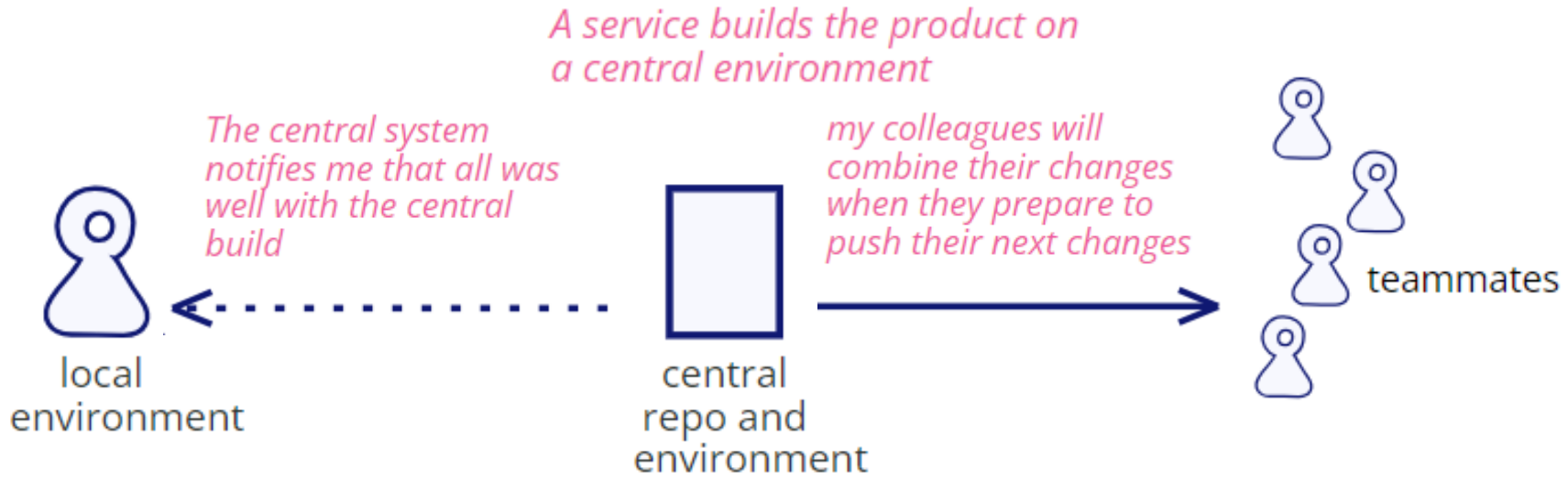
CI PROCESS - COMMIT PHASE



STEP 5

https://martinfowler.com/articles/continuousIntegration.html

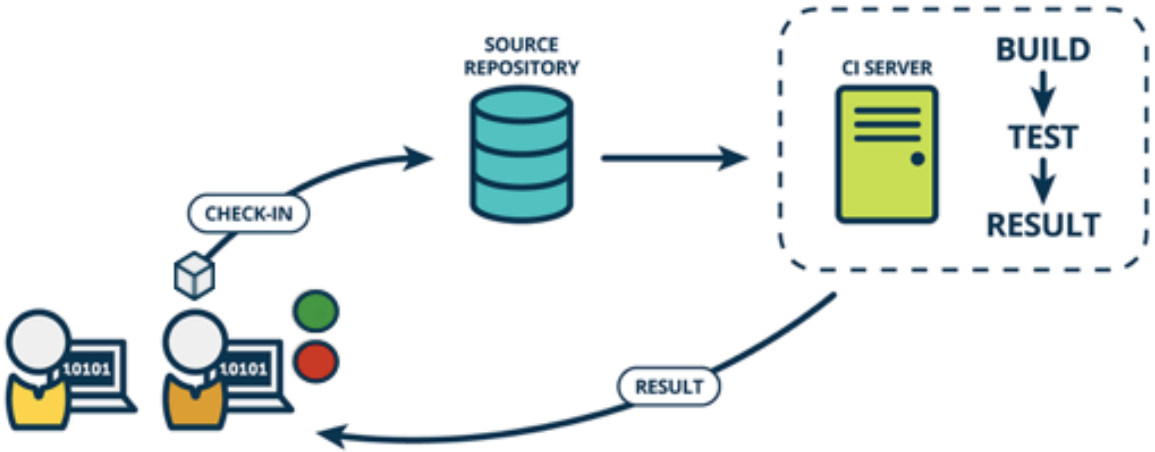
CI PROCESS - COMMIT PHASE



STEP 6

https://martinfowler.com/articles/continuousIntegration.html

CI PROCESS - CI SERVERS

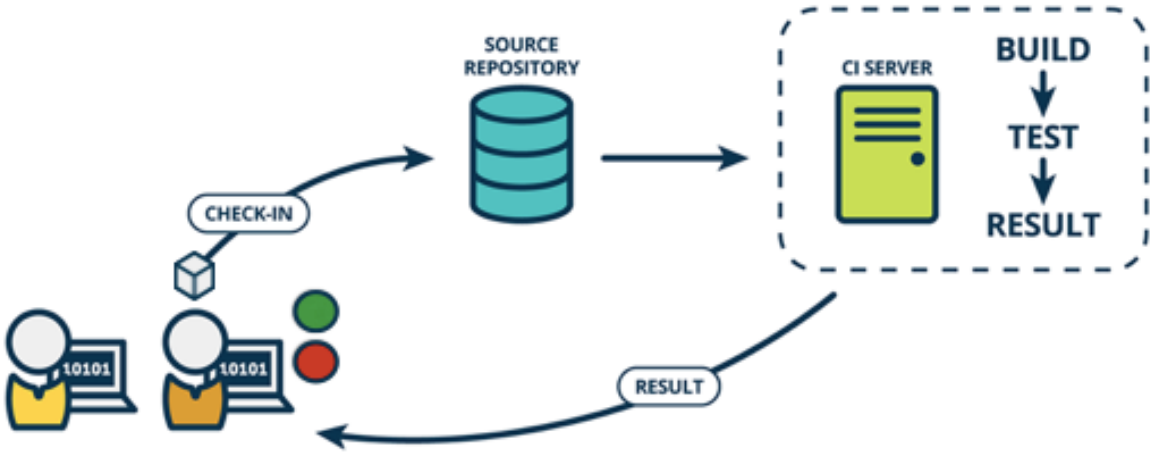


https://techbeacon.com/app-dev-testing/are-you-really-doing-continuous-integration-heres-how-tell

Typical tasks executed in a CI server

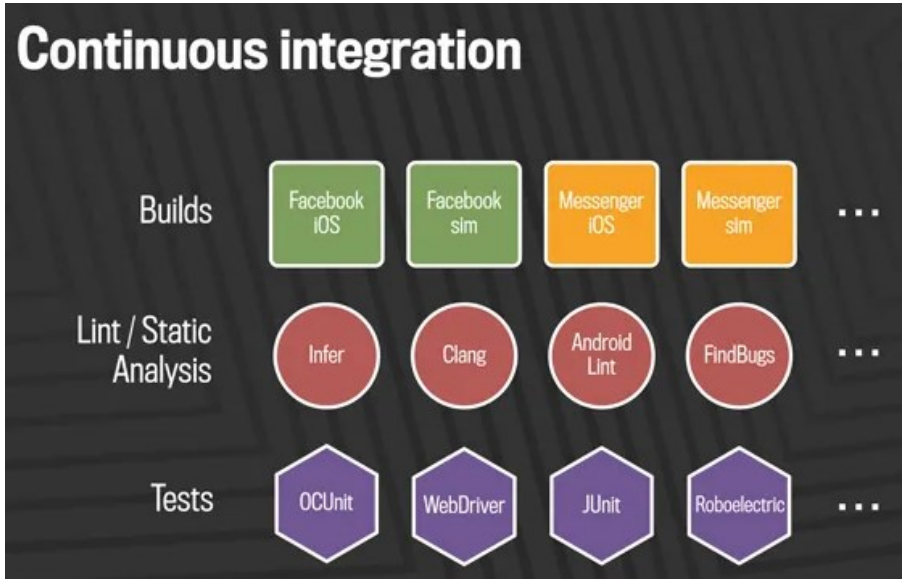
- Compilation
- Linters & code analysis
- Unit testing and integration testing
- Test coverage analysis
- Artifact generation
- Security scans
- Notification and reporting
- Deployment to testing/production environment (later)

CI PROCESS - CI SERVERS



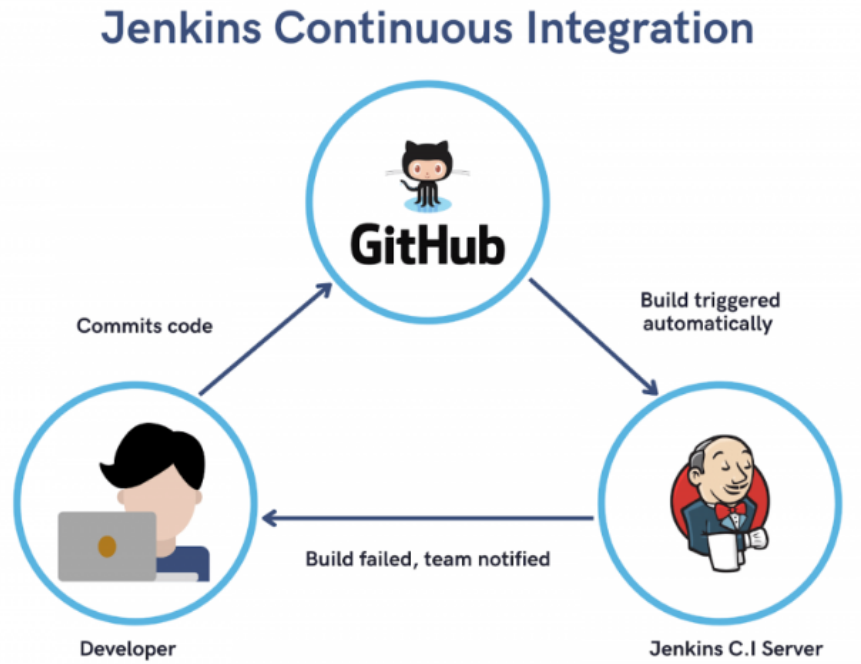
https://techbeacon.com/app-dev-testing/are-you-really-doing-continuous-integration-heres-how-tell

Engineering at Meta



https://engineering.fb.com/2017/08/31/web/rapid-release-at-massive-scale/

CI SERVERS



Jenkins TeamCity circleci Bamboo GitLab

- Jenkins is an open-source CI server.
- It helps automate building, testing, and deploying process
- It facilitates continuous integration and continuous delivery and is widely used in DevOps environments.

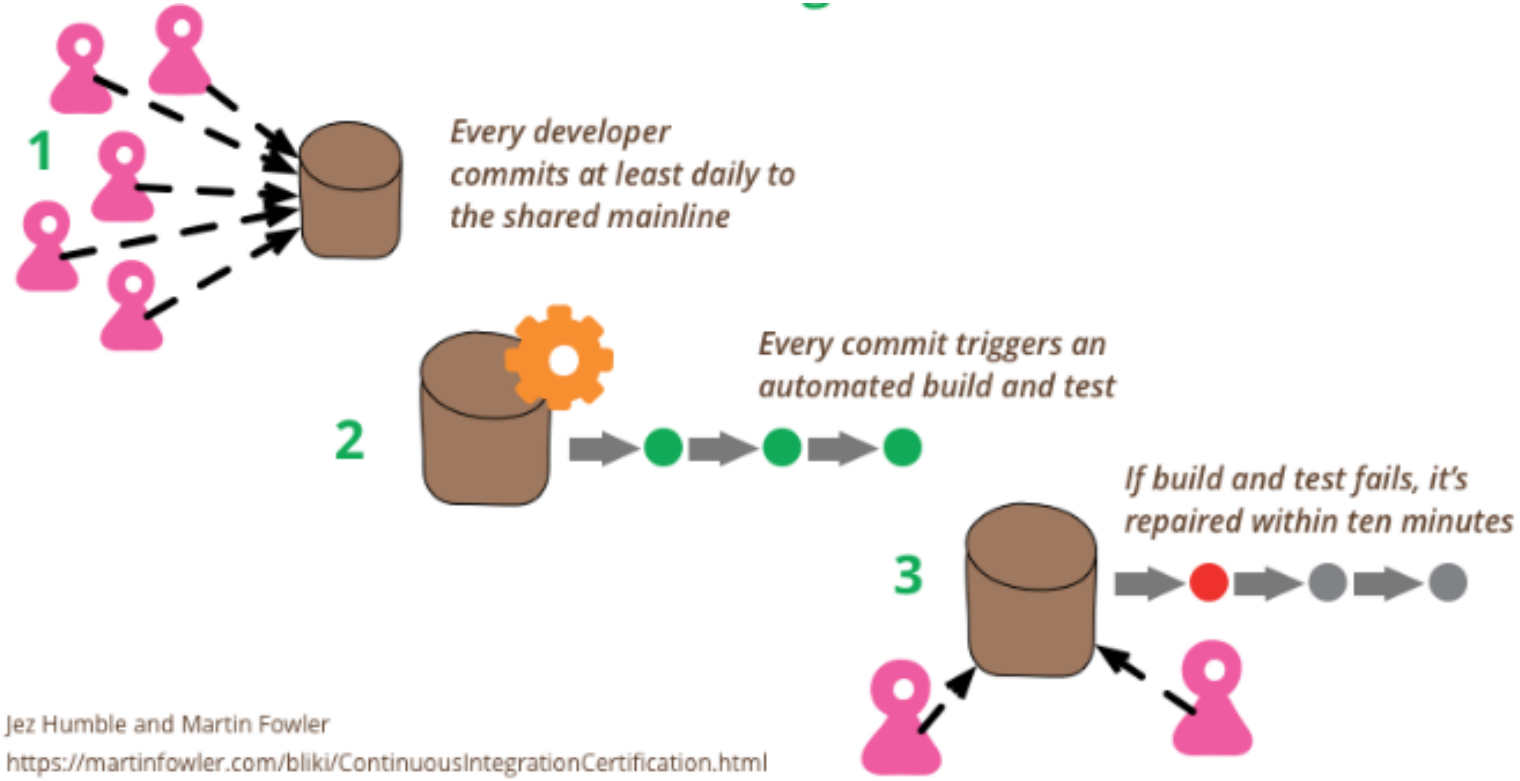
CI FOUNDATION: BUILDFILE AS CODE

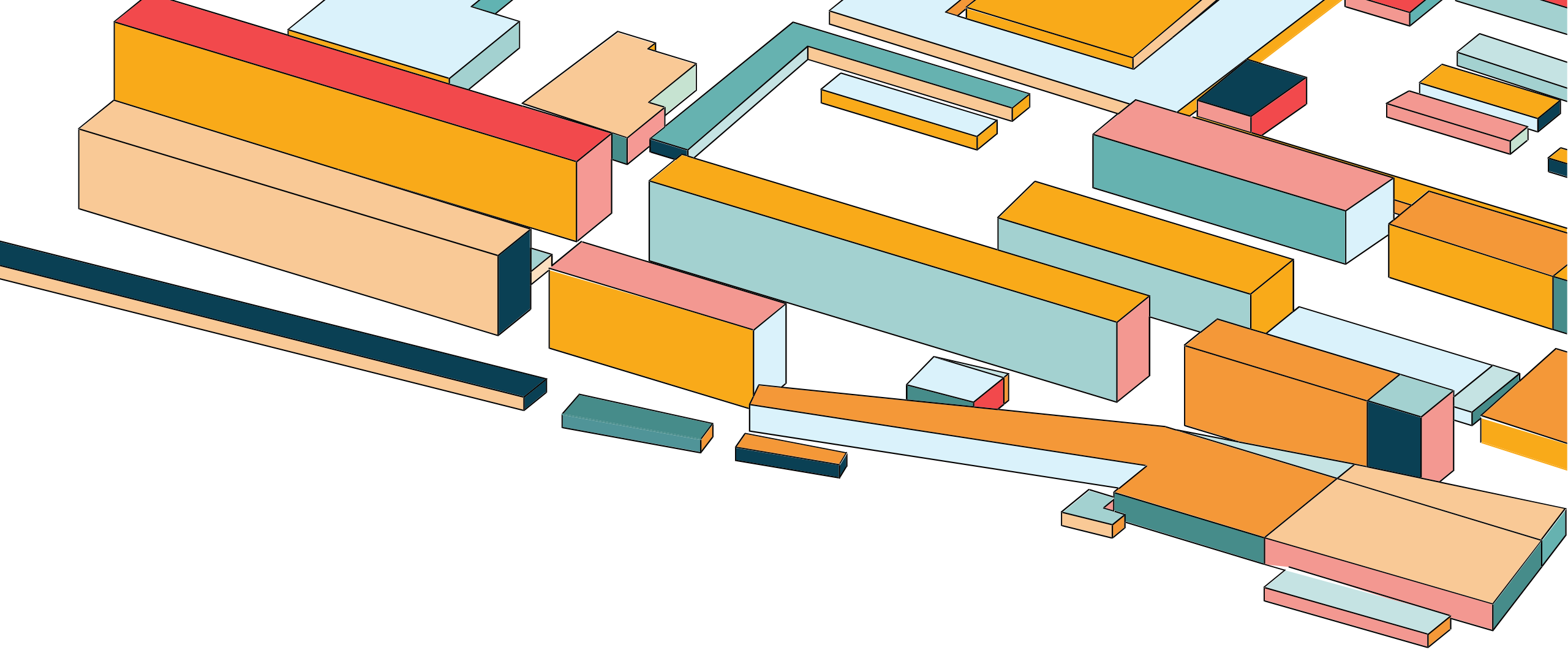
```
diff --git a/plc4j/pom.xml b/plc4j/pom.xml

index a18fc21..770ce9f 100644 (file)

--- a/plc4j/pom.xml
+++ b/plc4j/pom.xml
@@ -24,7 +24,7 @@
    <parent>
      <groupId>org.apache.plc4x</groupId>
      <artifactId>plc4x-parent</artifactId>
-     <version>0.3.0</version>
+     <version>0.3.1-SNAPSHOT</version>
    </parent>
```

CI BEST PRACTICE

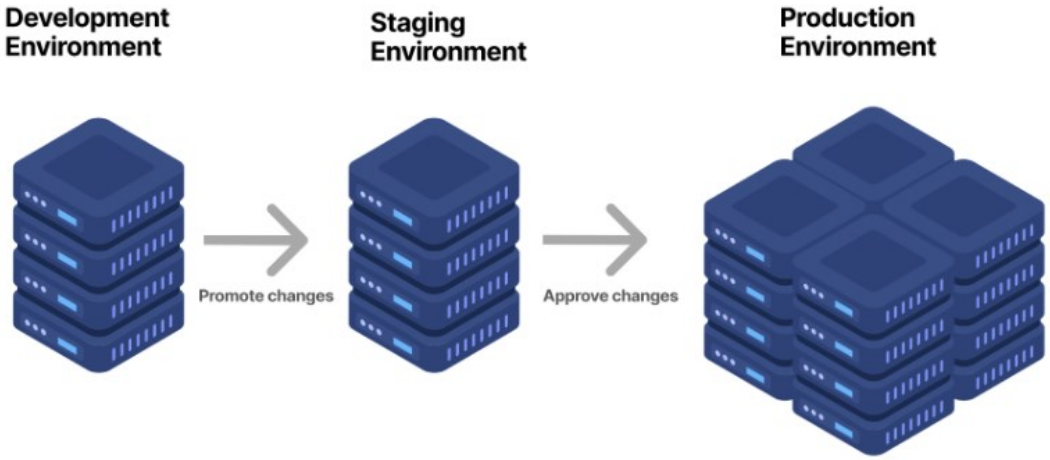




CONTINUOUS DELIVERY AND DEPLOYMENT

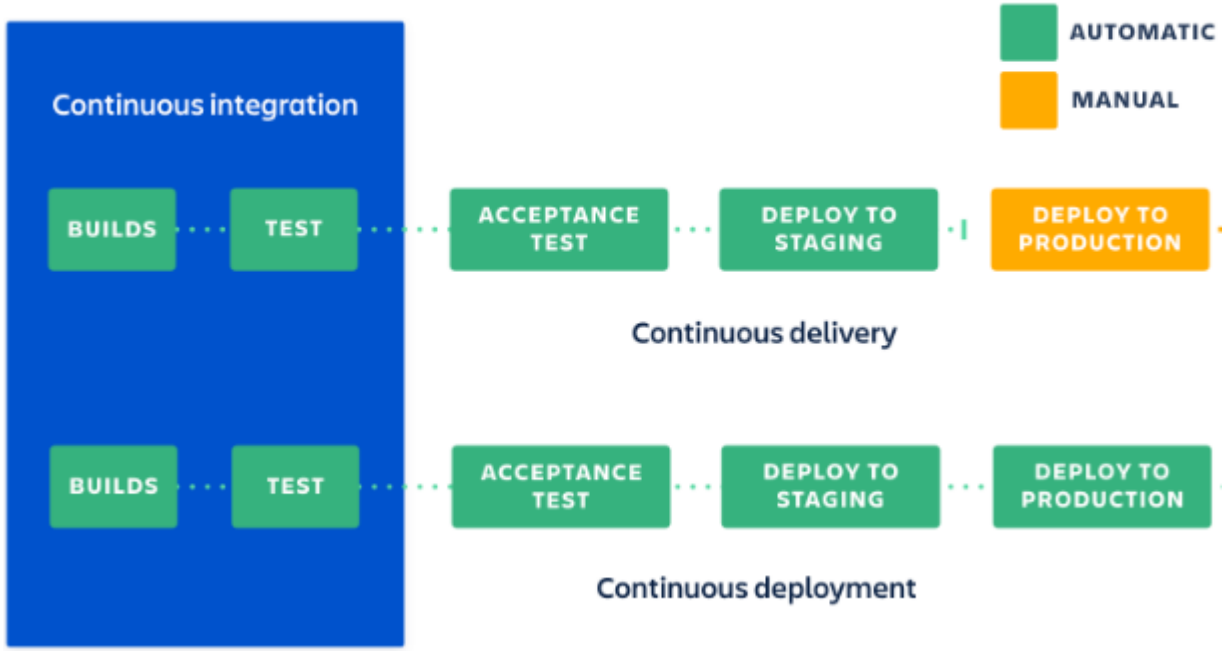
ENVIRONMENTS

- Development Environment (开发环境): where developers write, test, and debug code in an isolated setting before it is integrated with other codebases.
- Staging Environment (类生产环境): closely resembles the production environment and is used to validate changes, perform integration testing, and simulate real-world conditions before deploying updates to production.
- Production Environment (生产环境): the live environment where the final, tested code is deployed and accessed by end-users, serving as the backbone for delivering products or services to customers.



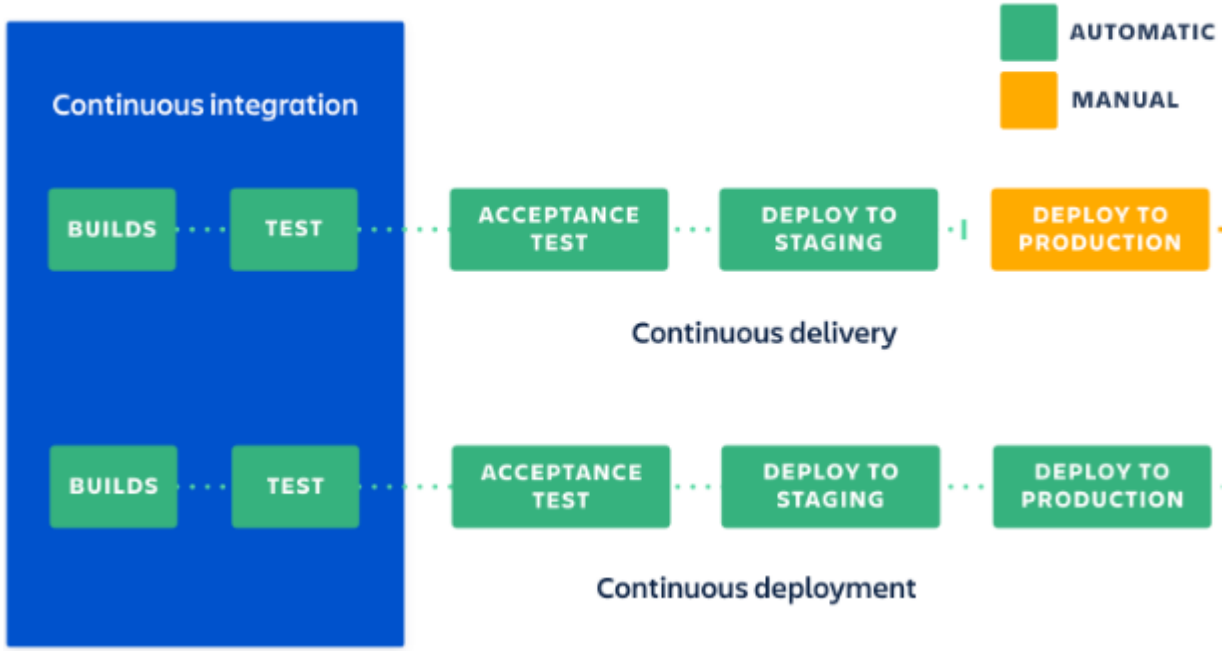
CONTINUOUS DELIVERY VS. DEPLOYMENT

- **Continuous delivery** is an extension of continuous integration since it automatically deploys all code changes to a staging and/or production environment after the build stage.
- With **continuous delivery**, you can decide to release daily, weekly, fortnightly, or whatever suits your business requirements.



CONTINUOUS DELIVERY VS. DEPLOYMENT

- **Continuous deployment** goes one step further than continuous delivery. With this practice, every change that passes all stages of your production pipeline is released to your customers.
- There's **no human intervention**, and only a failed test will prevent a new change to be deployed to production.



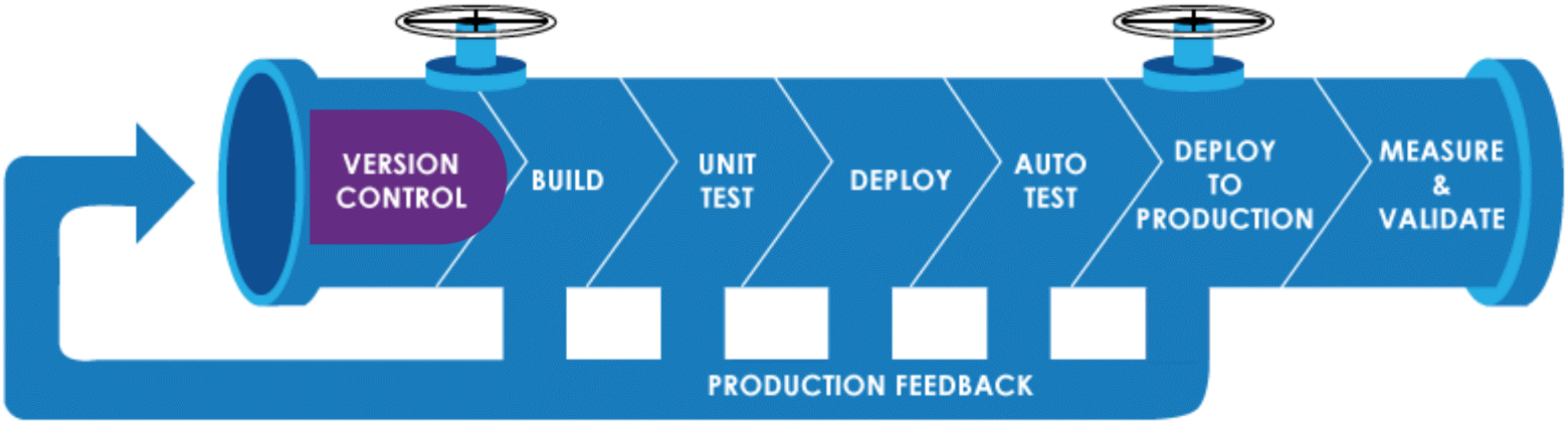
With continuous deployment, developers can focus on building software, and they see their work go live minutes after they've finished working on it.

CI/CD PIPELINE

- A software deployment pipeline is a set of automated processes that an application goes through from the development to final production

- A deployment pipeline includes various stages such as building, testing, and deploying, and different environments like staging, testing, and production.

CI/CD PIPELINE



https://medium.com/jaanvi/basics-of-ci-cd-pipeline-5762e0eca44e

CI/CD PIPELINE - ARCHITECTURE

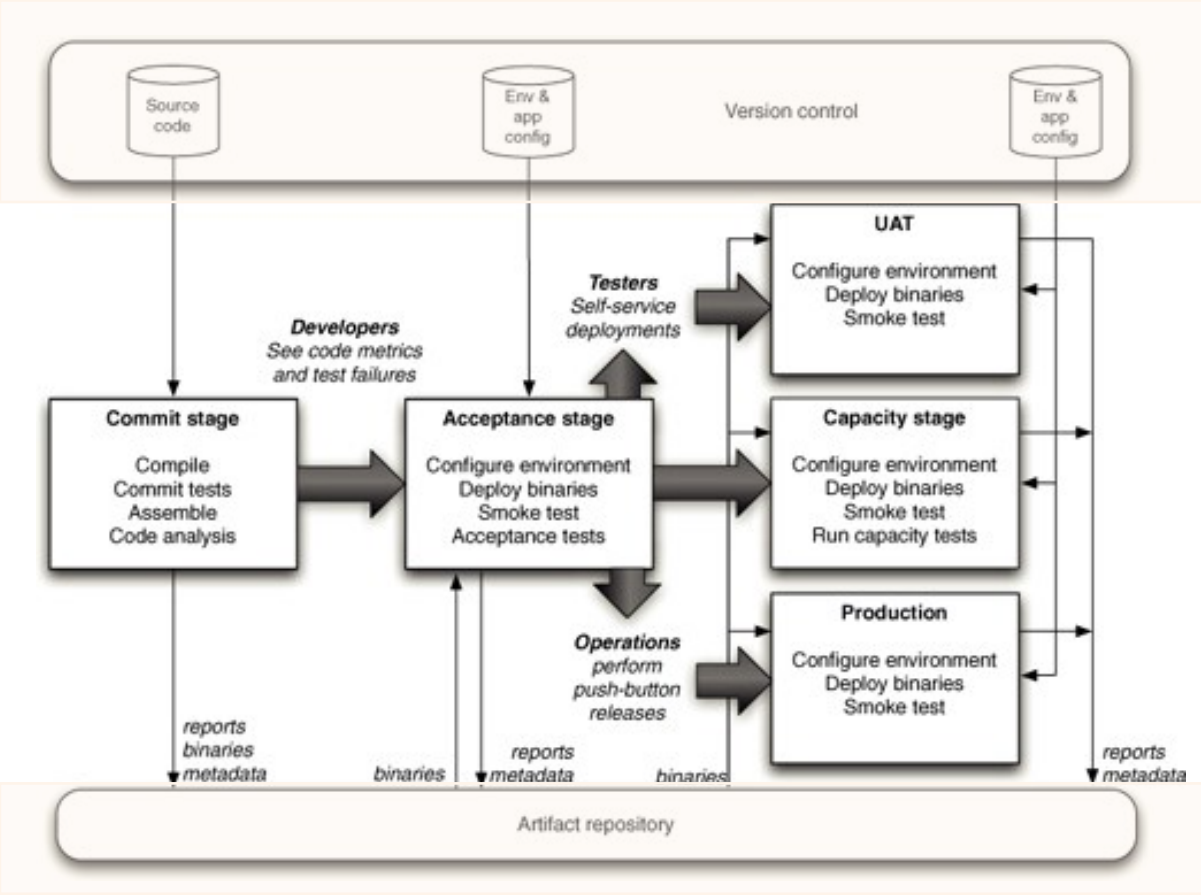
Common components of a deployment pipeline

- Version control
- Artifact repositories
- Build automation tools
- Automated testing tools
- Continuous integration (CI) servers
- Deployment tools

PIPELINE BREAKDOWN

Required for CI/CD pipelines

- Version control
  - Source code
  - Buildfile (as code)
  - Documentation (as code)
  - Infrastructure (as code)
- Artifact repositories
  - Libraries, plugins, etc.

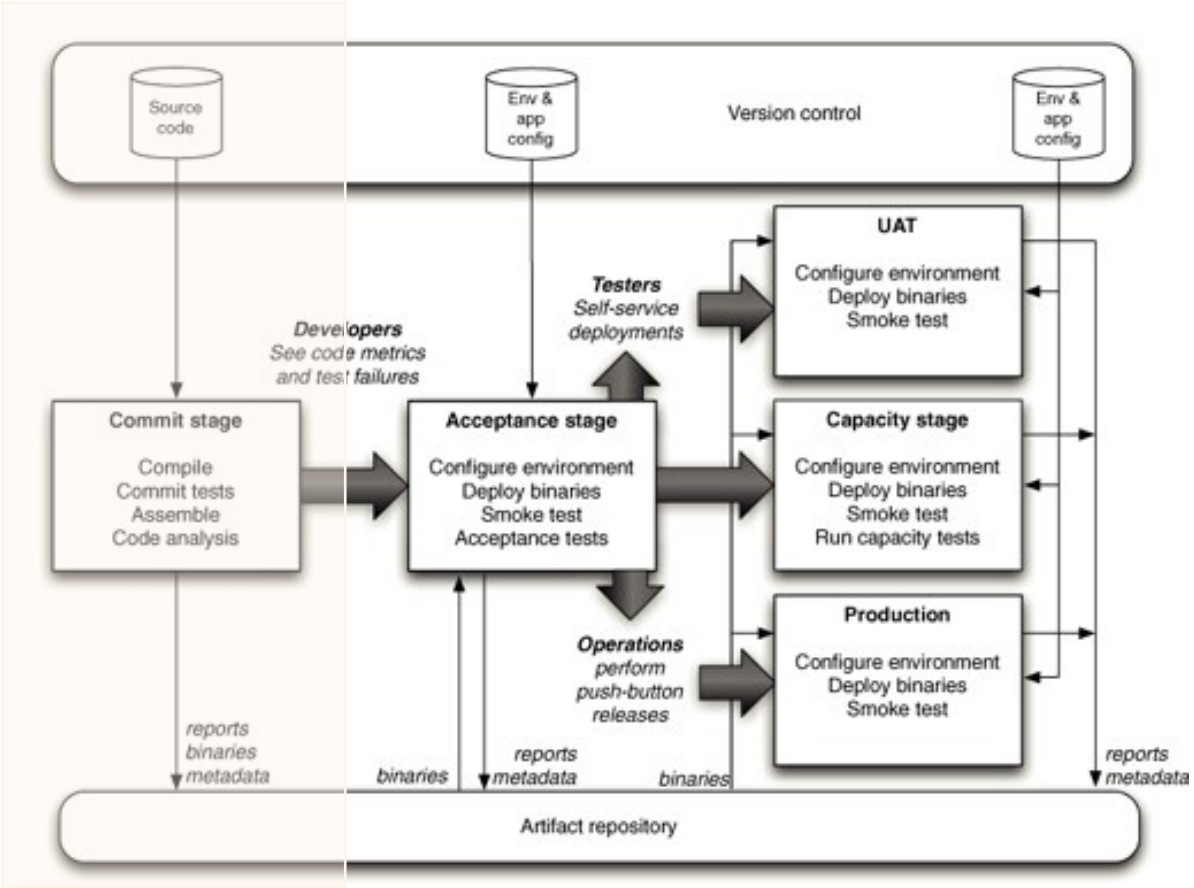


Continuous Delivery Reliable Software Release through Build, Test, and Deployment Automation. Jez Humble and David Farley.

PIPELINE BREAKDOWN

Stage 1

1. Commit to version control
2. Trigger the build
   - Compile
   - Unit Test
   - Doc
   - Binary
3. Generated binaries stored in the artifact repository

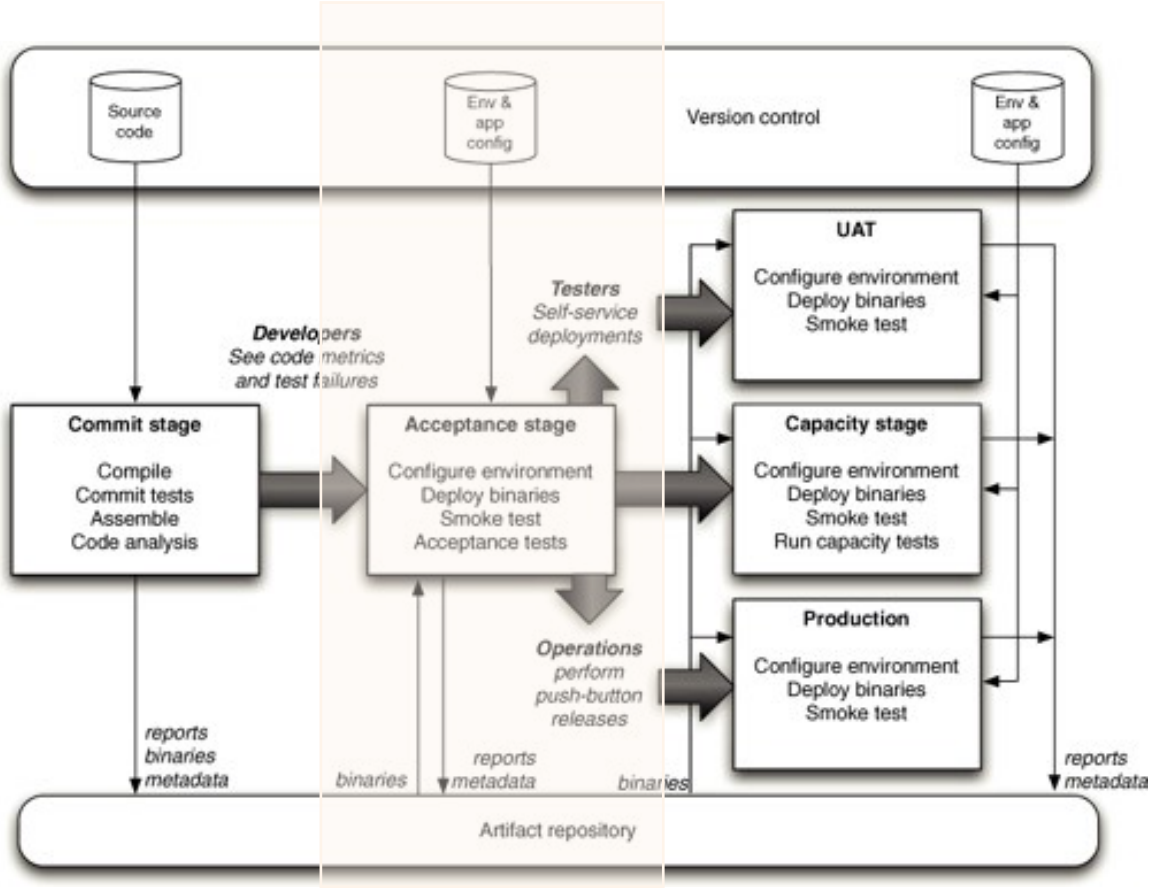


Continuous Delivery Reliable Software Release through Build, Test, and Deployment Automation. Jez Humble and David Farley.

PIPELINE BREAKDOWN

Stage 2

1. Automatically triggered by stage 1
2. Run automated acceptance tests that take a long time to execute (e.g., integration testing, system/E2E testing)
3. Can be executed on the staging environment



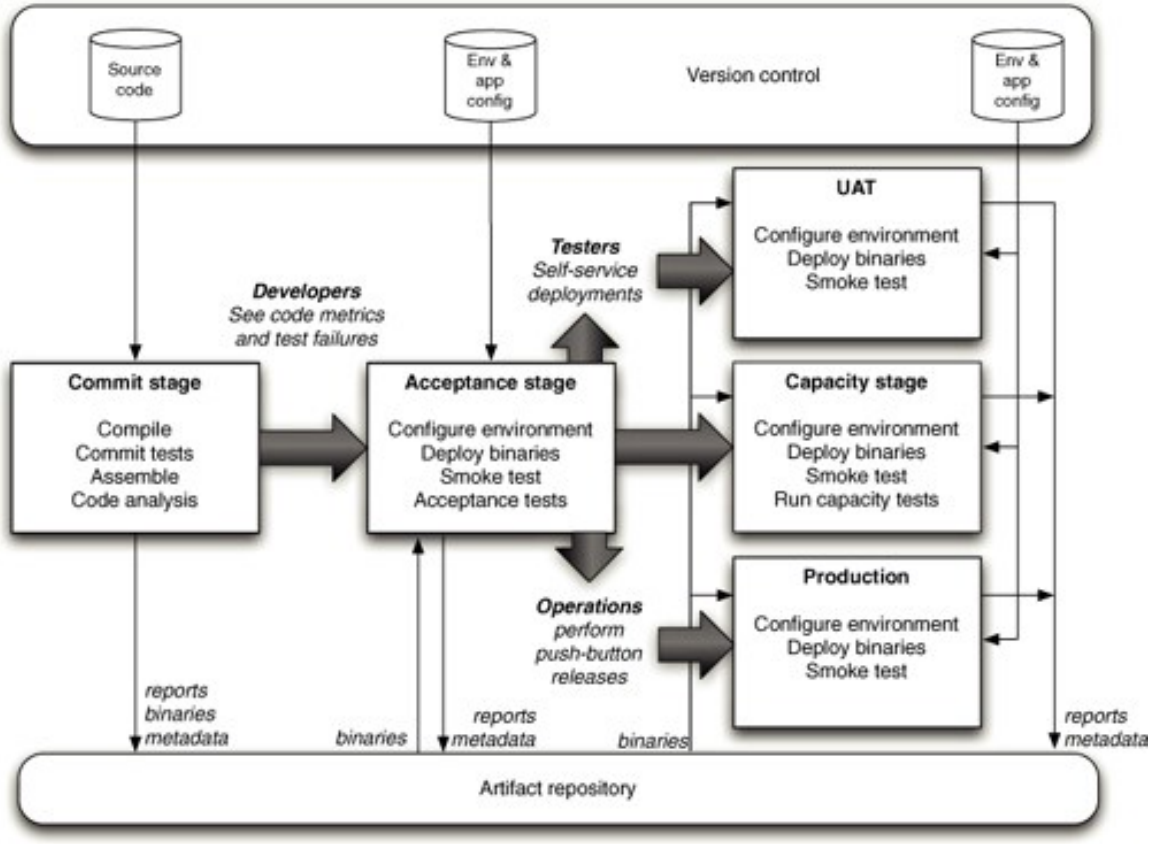
Continuous Delivery Reliable Software Release through Build, Test, and Deployment Automation. Jez Humble and David Farley.

PIPELINE BREAKDOWN

Stage 2 – notes

1. Configurations for the environment are version controlled just like source code (infrastructure as code)

2. Smoke test: It consists of a minimal set of tests run on each build to test major functionalities. Smoke testing is a confirmation for QA team to proceed with further software testing.



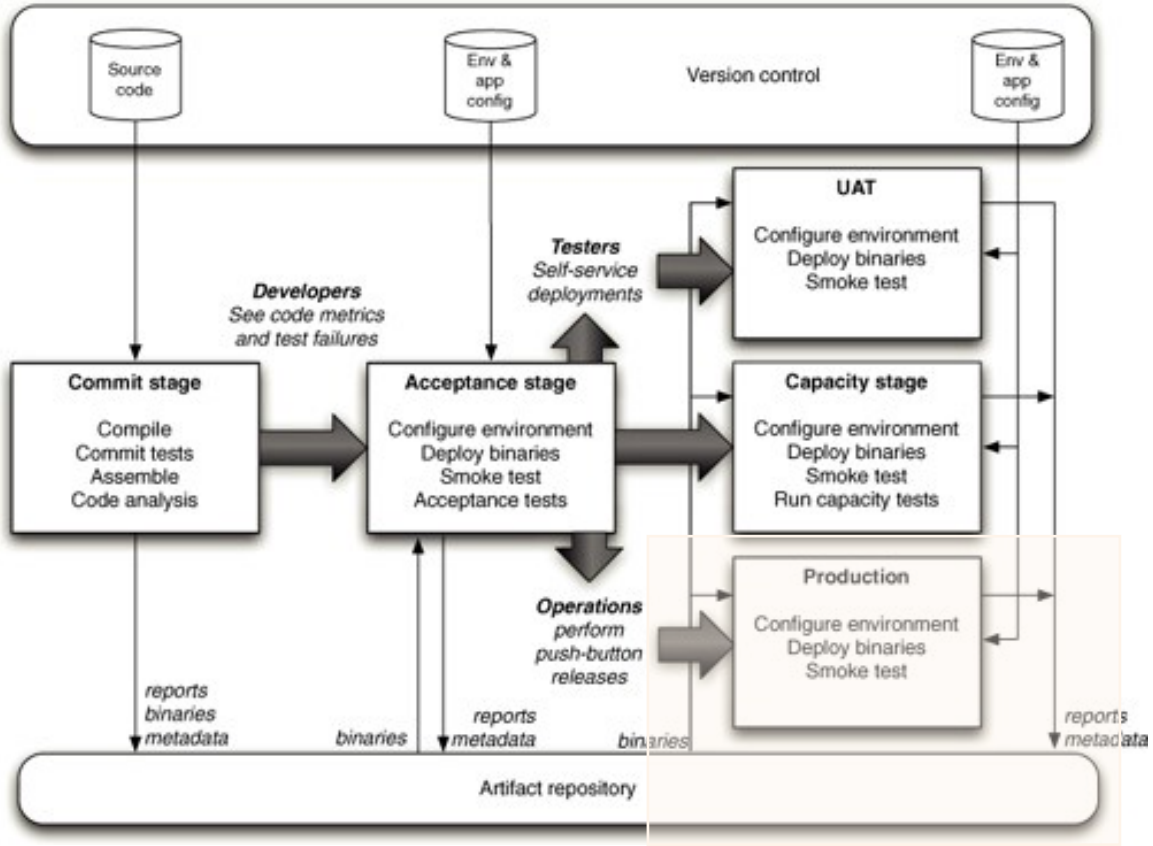
Continuous Delivery Reliable Software Release through Build, Test, and Deployment Automation. Jez Humble and David Farley.

PIPELINE BREAKDOWN

Stage 3

1. There might be different options for different companies

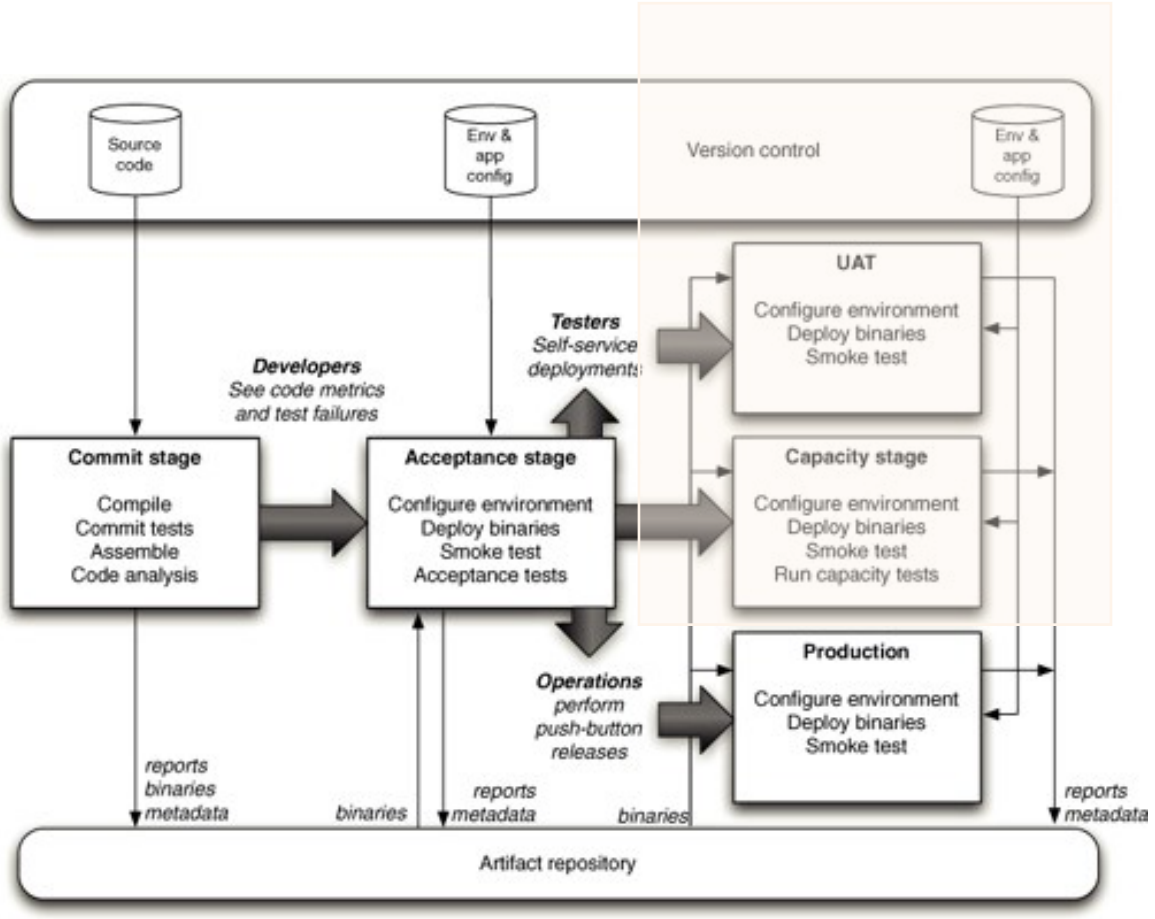
2. The most basic one: the Ops team confirms and deploys the verified binary version to the production environment with one button click.



PIPELINE BREAKDOWN

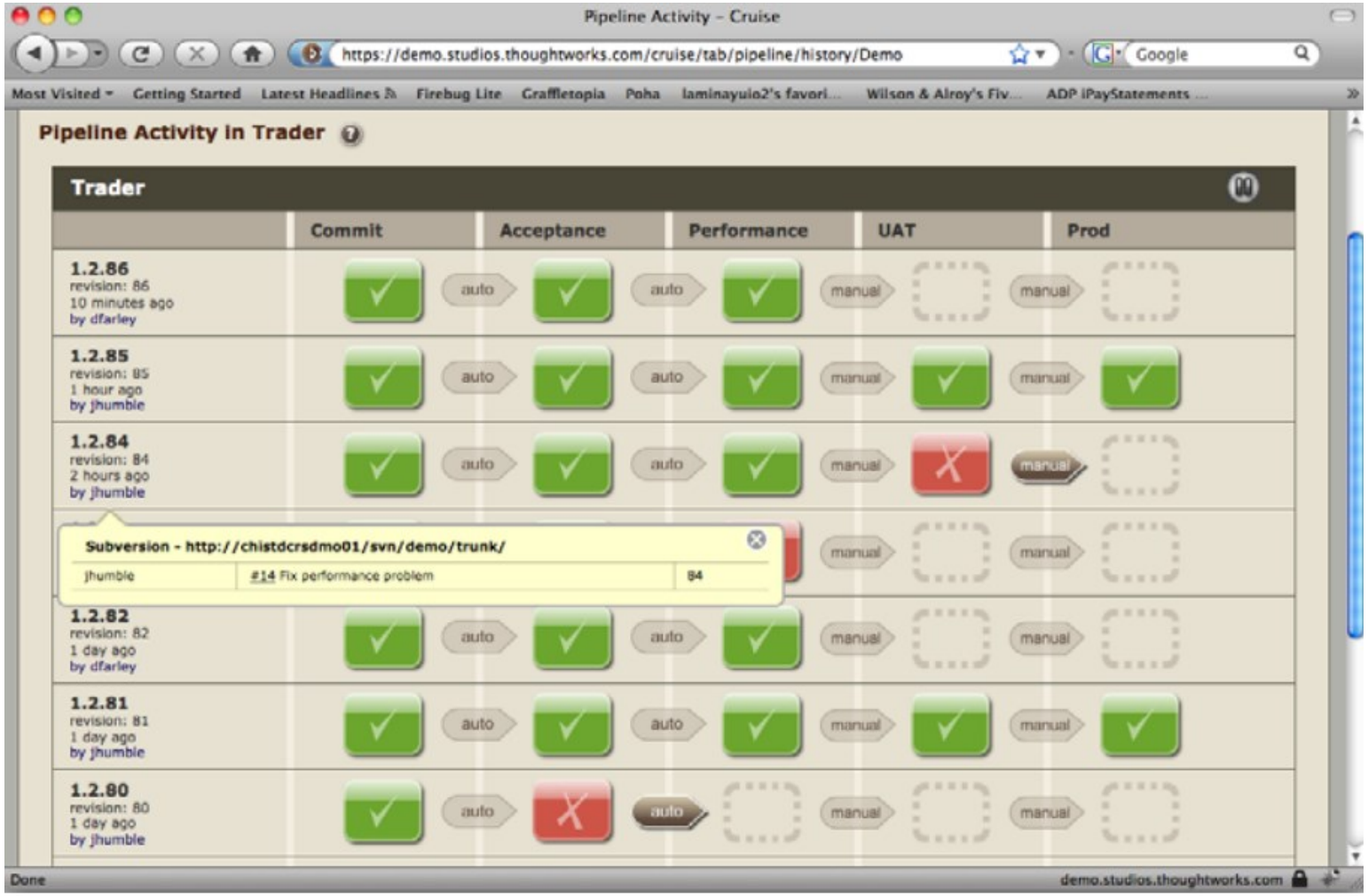
Stage 3 - Optional

1. The binary from stage 2 is automatically deployed to a staging environment for tool-aided non-functional tests or manual UAT by testers.
2. Testers could select the proper version to be released to the production environment.



Continuous Delivery Reliable Software Release through Build, Test, and Deployment Automation. Jez Humble and David Farley.

A CI/CD PIPELINE RUNNING INSTANCE

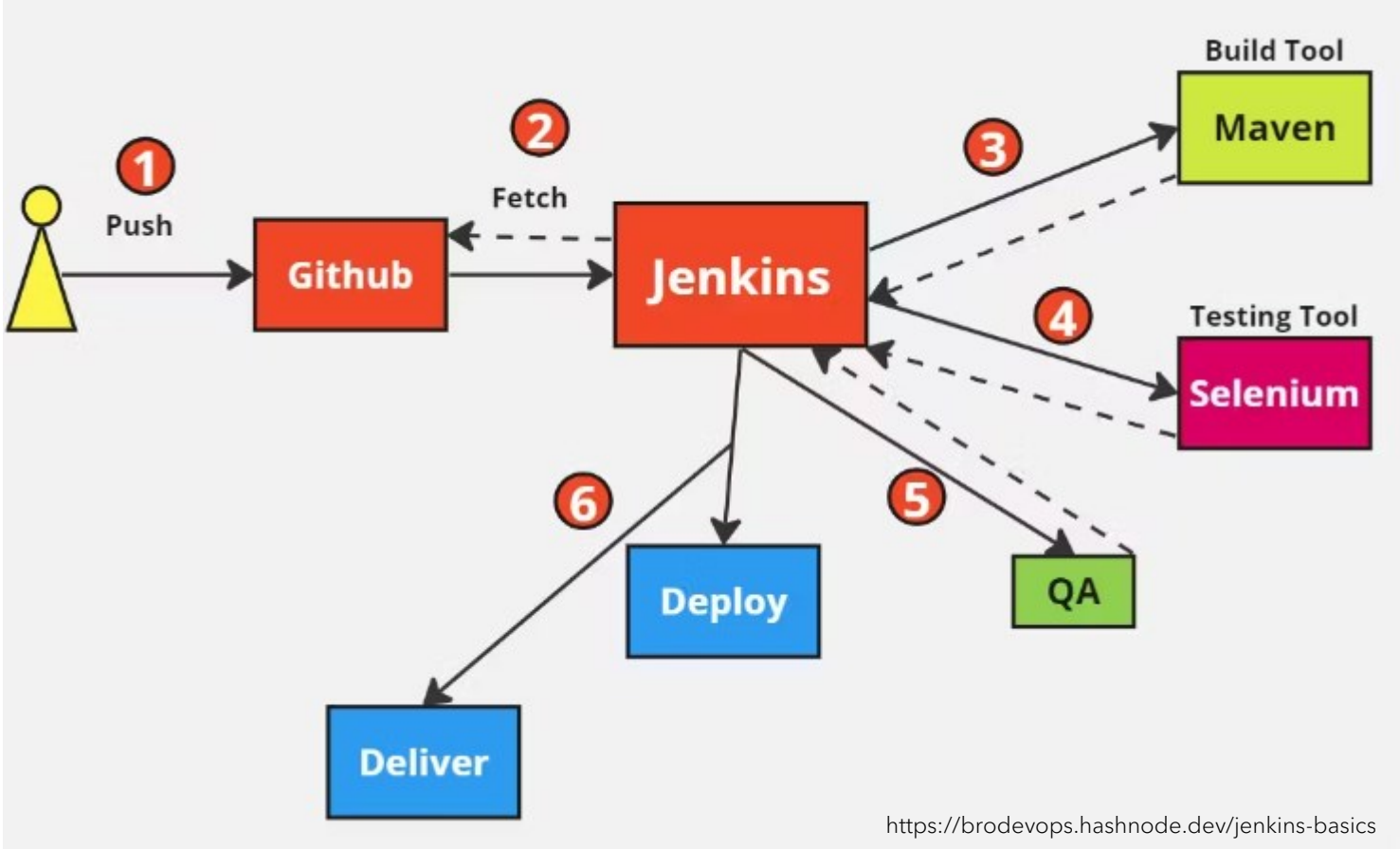


Continuous Delivery Reliable Software Release through Build, Test, and Deployment Automation. Jez Humble and David Farley.

CI/CD PIPELINE

- Ensures that the application is delivered reliably, quickly, and with high quality.
- Enables the team to deliver software changes more frequently and with greater confidence. Minimizes the risk of human error.
- Provides visibility into the progress of the application
- Enables faster feedback to developers in case of any issues or failures.

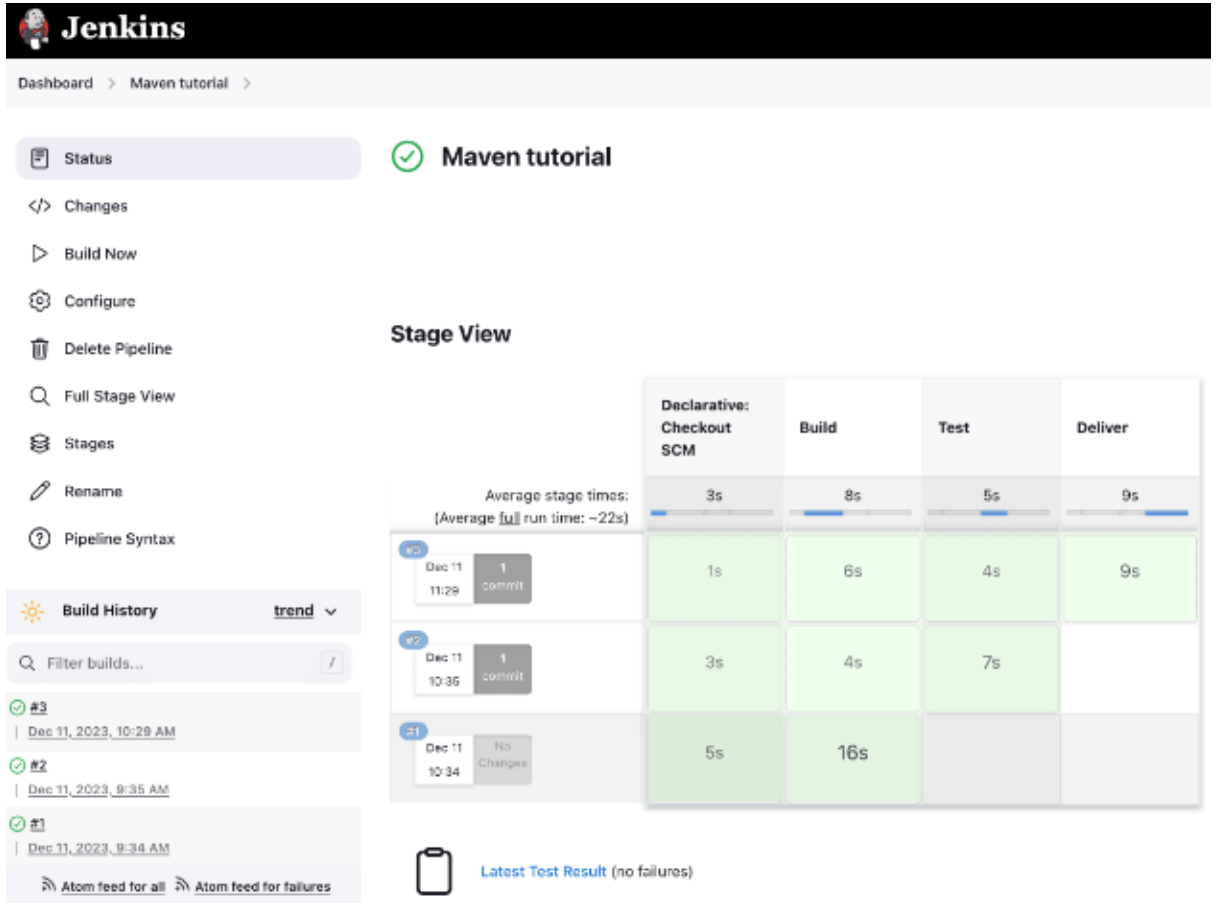
JENKINS PIPELINE



https://brodevops.hashnode.dev/jenkins-basics

JENKINS PIPELINE

```
pipeline {
    agent any
    options {
        skipStagesAfterUnstable()
    }
    stages {
        stage('Build') {
            steps {
                sh 'mvn -B -DskipTests clean package'
            }
        }
        stage('Test') {
            steps {
                sh 'mvn test'
            }
            post {
                always {
                    junit 'target/surefire-reports/*.xml'
                }
            }
        }
        stage('Deliver') { (1)
            steps {
                sh './jenkins/scripts/deliver.sh' (2)
            }
        }
    }
}
```



https://www.jenkins.io/doc/tutorials/build-a-java-app-with-maven/

CI/CD IN INDUSTRY

A P2P e-commerce company Etsy: before and after continuous deployment

	Before 2010	After 2010
Deployment time	Server shutdown for 6-14 hours	<15 mins
Deployment effort	A dedicated team	1 person
Deployment frequency	Highest priority, but very low frequency	50 / day

Source: Continuous Delivery: The Dirty Details. Mike Brittain. 2012

CI/CD IN INDUSTRY

Company	Deploy Frequency	Deploy Lead Time	Reliability	Customer Responsiveness
Amazon	23,000 / day	minutes	high	high
Google	5,500 / day	minutes	high	high
Netflix	500 / day	minutes	high	high
Facebook	1 / day	hours	high	high
Twitter[2]	3 / week	hours	high	high
typical enterprise	once every 9 months	months or quarters	low/medium	low/medium

Source: Continuous Delivery. Jez Humble, David Farley. 2011

CI/CD VS. RAPID RELEASE

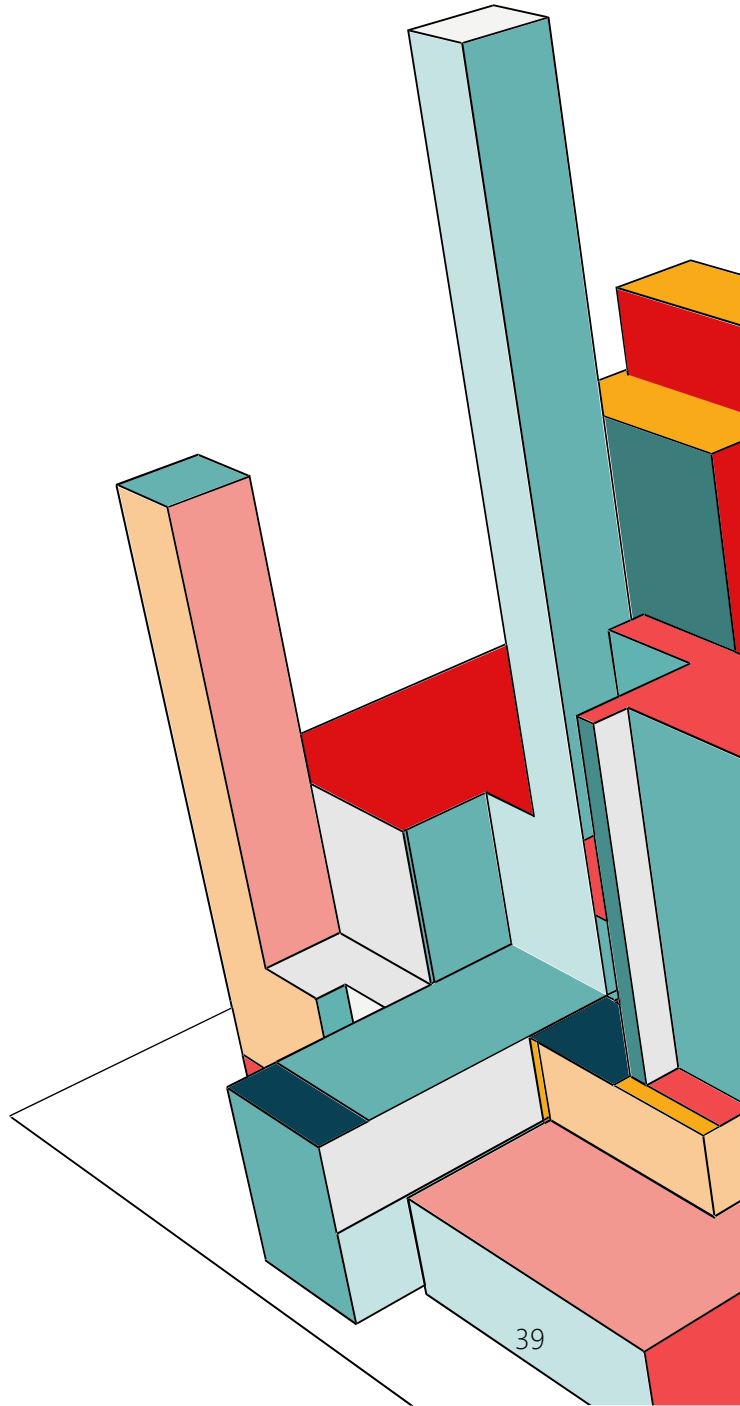
- Deployment: making the software available for use, in staging or production environment.
- Release: making the software accessible by end users in the production environment.

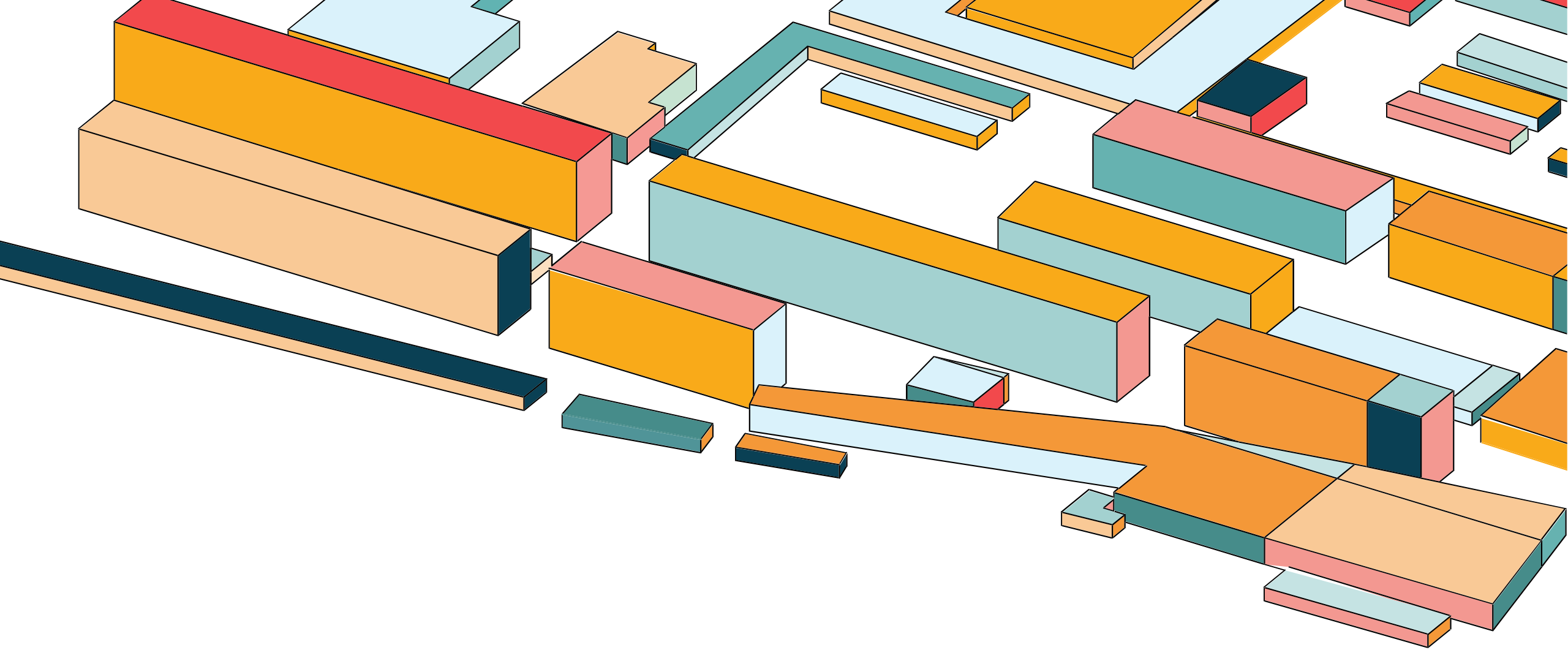
Sometimes people use “deployment” and “release” for the same thing: some companies will release at the same time as deployment to production is taking place.

So, CI/CD is sometimes also referred to as “rapid release” (高频发布).

RISKS OF CI/CD OR RAPID RELEASE

- New production version (release) has problems and needed to be rolled back to previous versions
- Service is temporarily down during upgrading or rollback. Every second spent in downtime loses money in the operation.
- Companies should have deployment/release strategies to handle the risks.



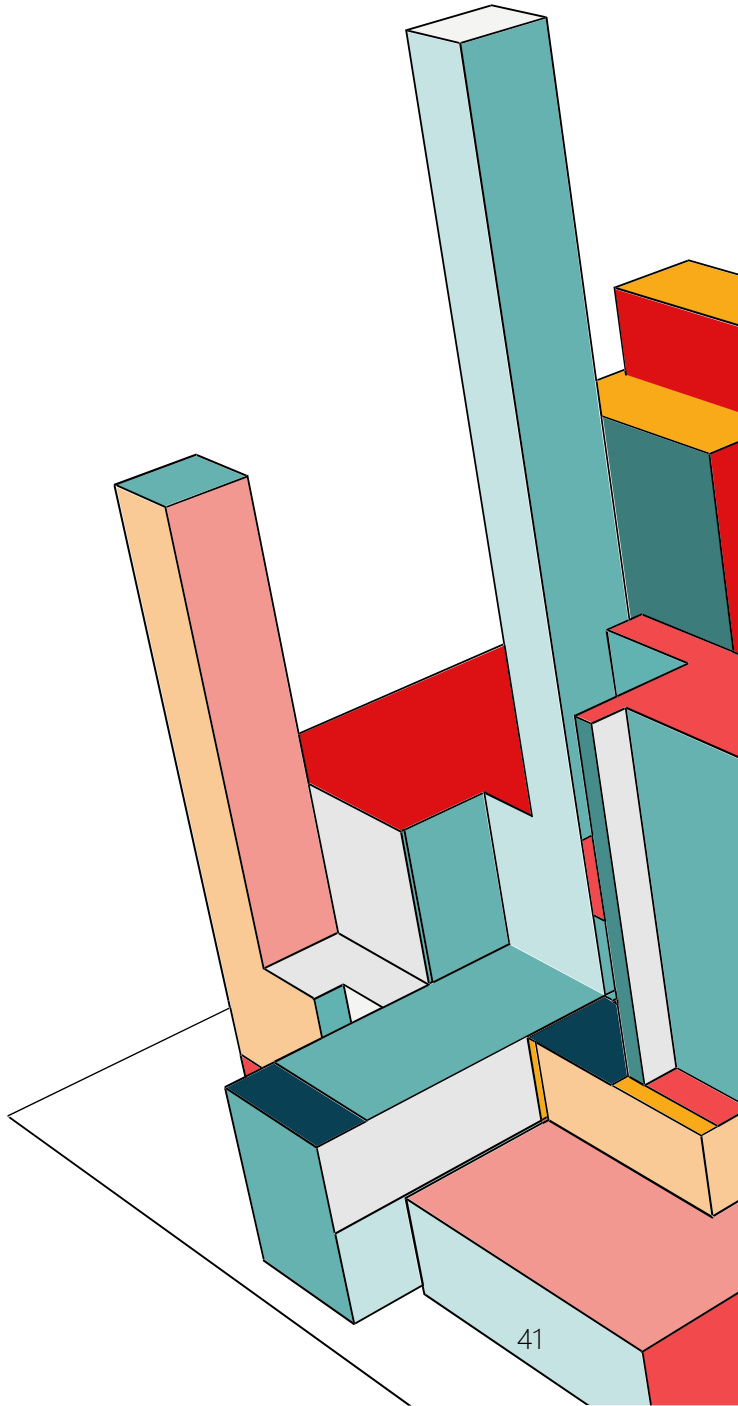


DEPLOYMENT STRATEGY

EXPECTATIONS FOR CONTINUOUS DEPLOYMENT

- Zero Downtime
- Quick to market, rapid release
- Quick customer feedback on new release
- Easy to roll back in case something goes wrong

Low-risk release (低风险发布)



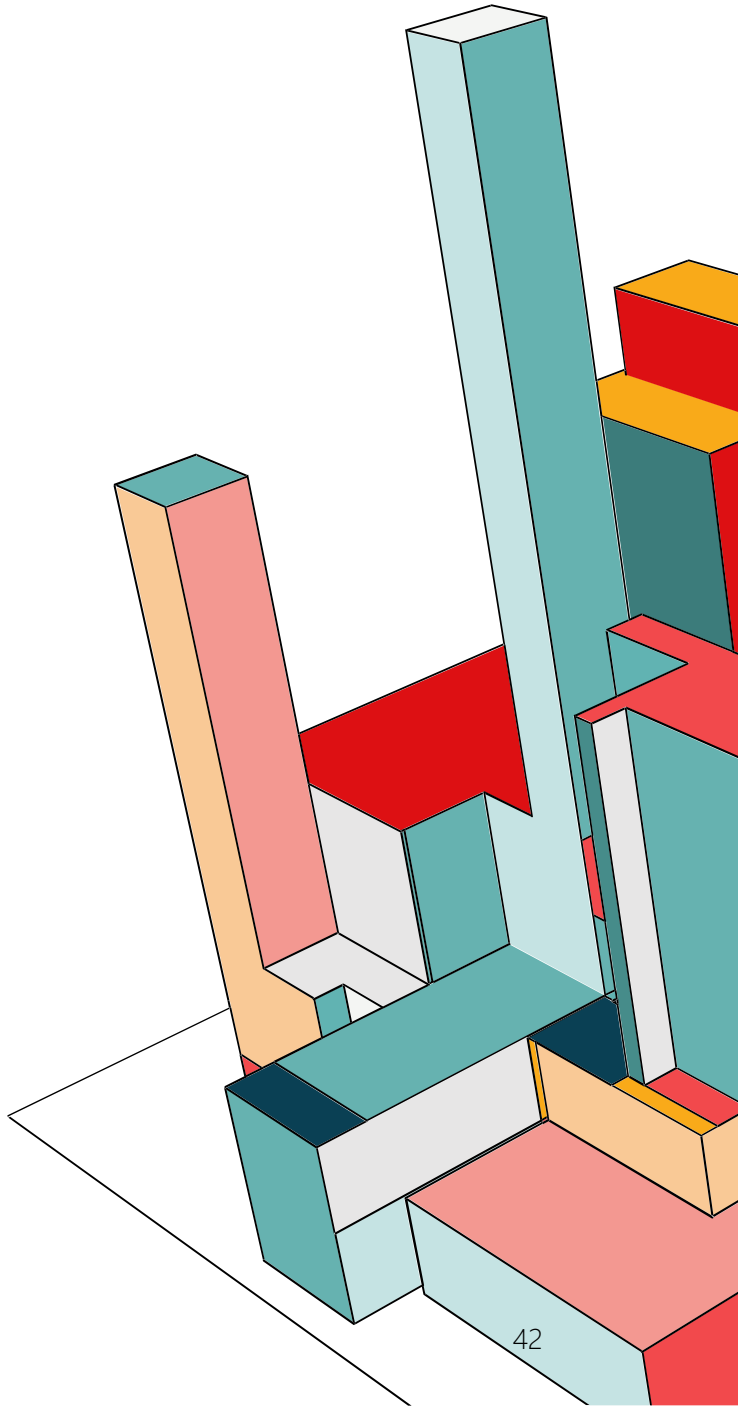
DEPLOYMENT STRATEGIES

Goal: minimize the impact of new deployment on end users

- Blue-Green Deployment (蓝绿部署)
- Rolling Deployment (滚动部署)
- Canary/Greyscale Release (金丝雀发布/灰度发布)

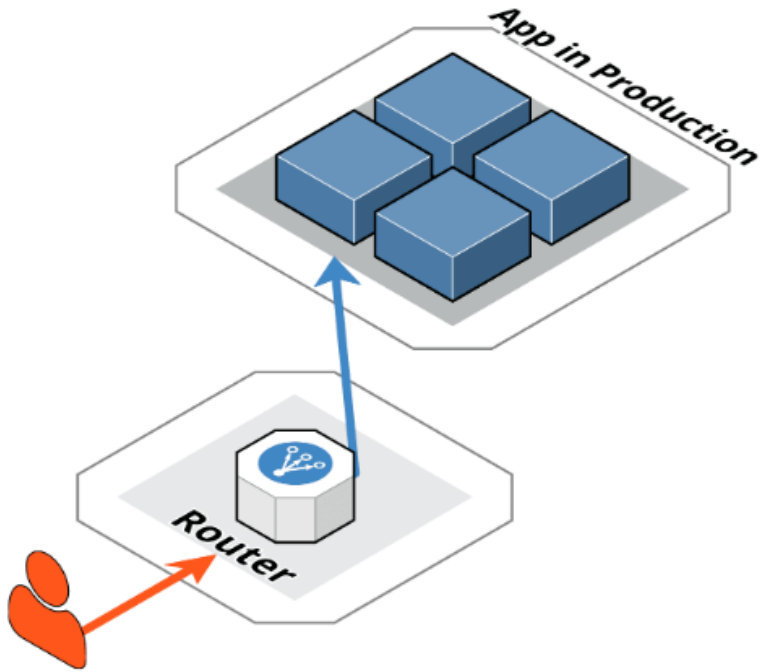
Supporting technology

- Feature toggle (功能开关)



BLUE-GREEN DEPLOYMENT

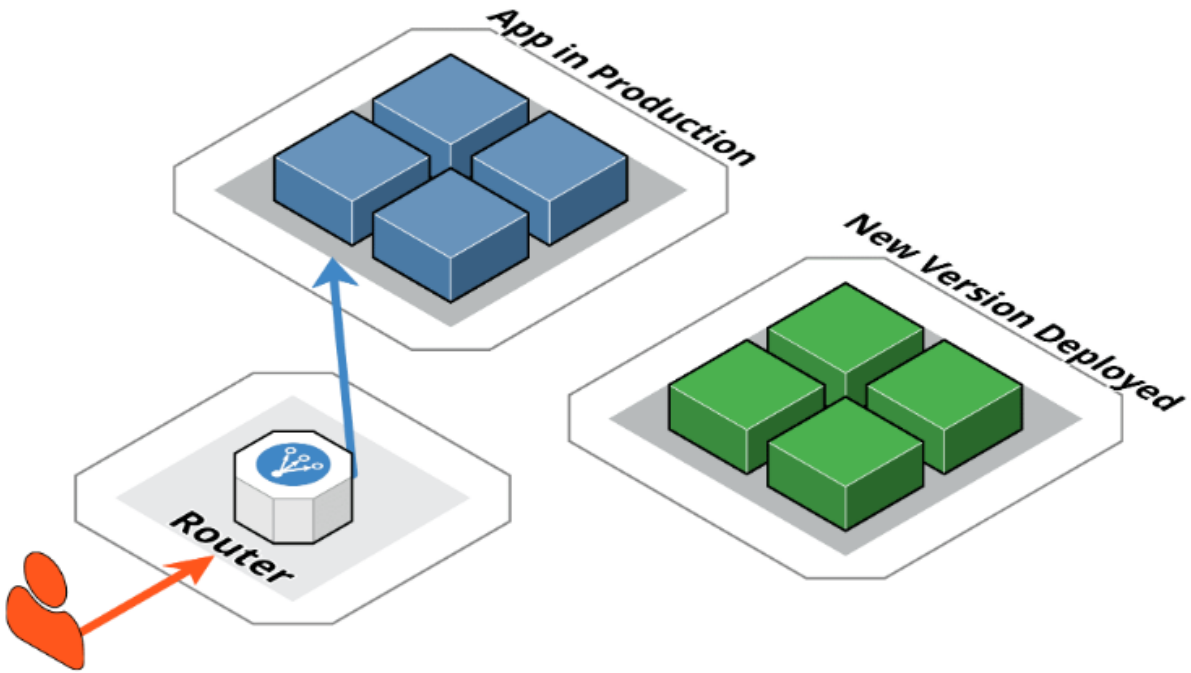
- A deployment strategy in which you create two separate, but identical environments.
- Blue environment is running the current/old application version



https://candost.blog/the-blue-green-deployment-strategy/

BLUE-GREEN DEPLOYMENT

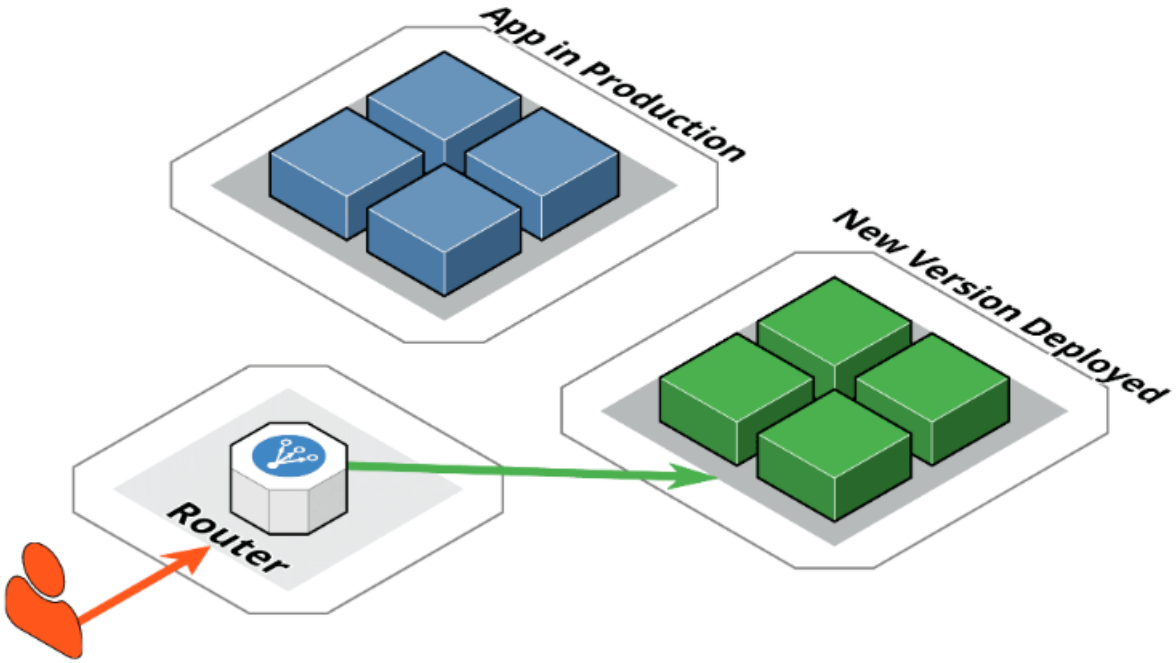
- A deployment strategy in which you create two separate, but identical environments.
- Blue environment is running the current/old application version
- When we want to deploy a new version, a.k.a. green, we create a new green environment close to production and deploy our new version into it.



https://candost.blog/the-blue-green-deployment-strategy/

BLUE-GREEN DEPLOYMENT

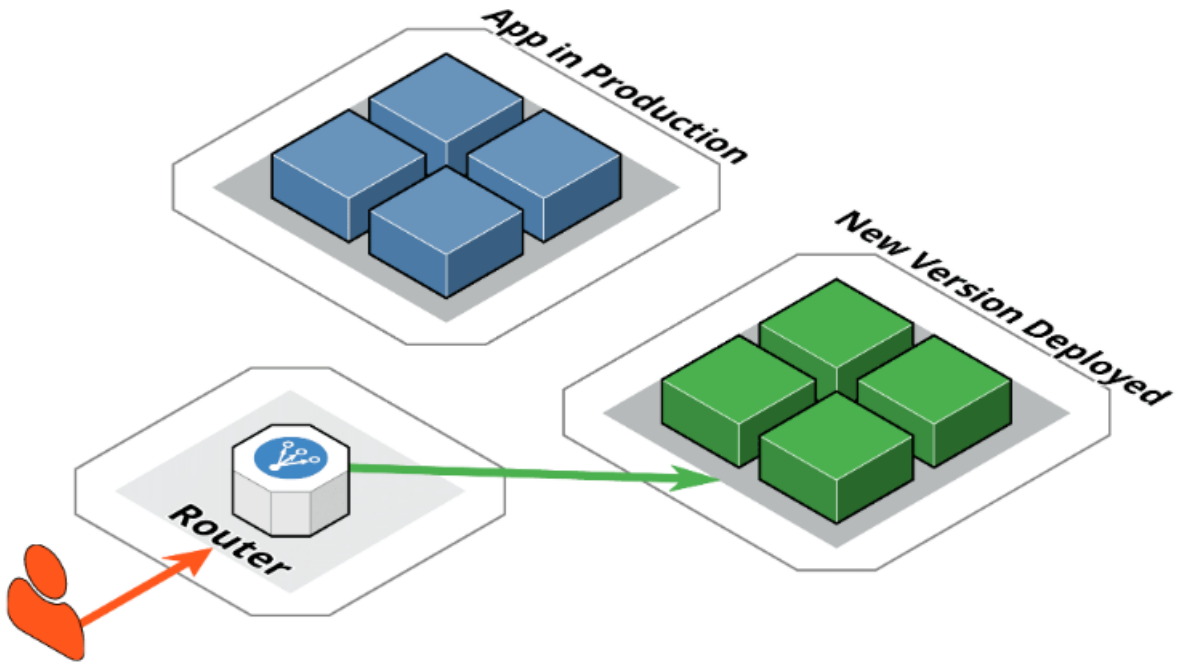
- After testing and making sure the green is working correctly, we route all traffic to the new green version.
- When we fully transfer the traffic to the new version, we can destroy or recycle the old blue instances.
- When we discover the green version is broken, it's easy to roll back to the blue version.



https://candost.blog/the-blue-green-deployment-strategy/

BLUE-GREEN DEPLOYMENT

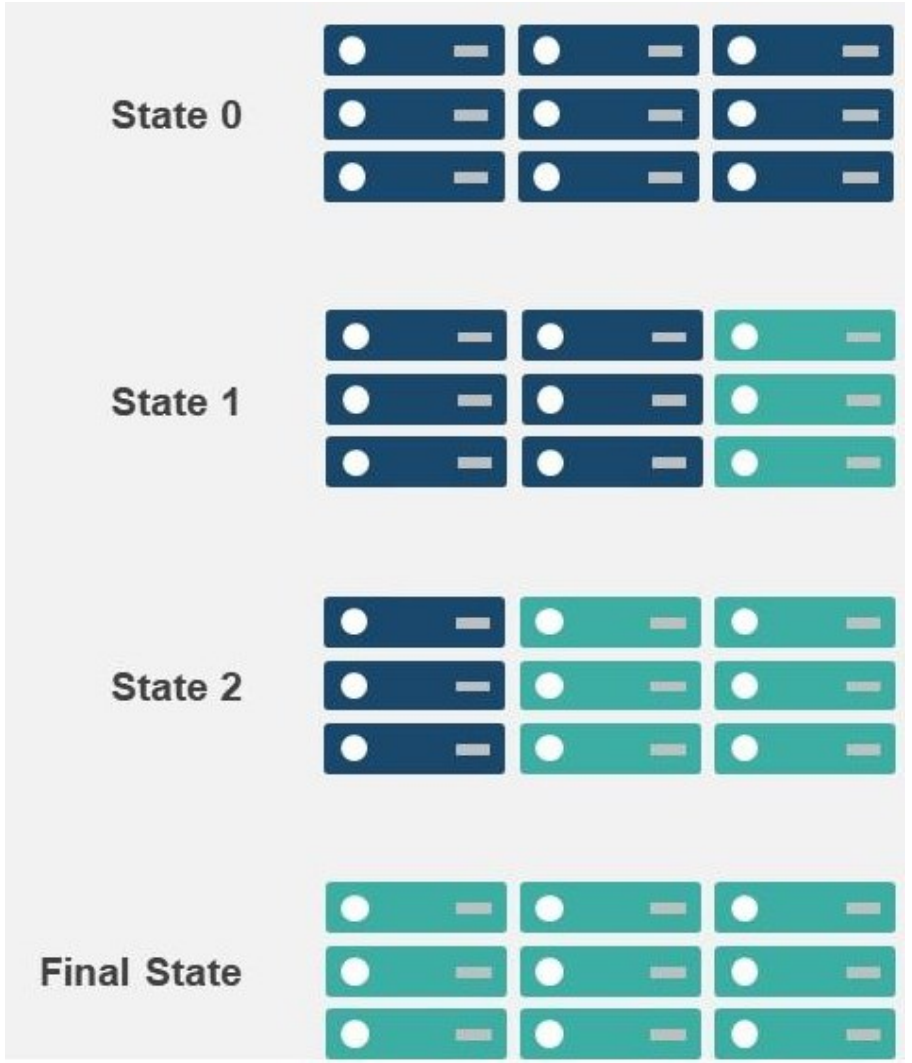
- Benefits
  - Minimal downtime
  - Rollback is easy and fast
- Drawbacks
  - Expensive: a redundant infrastructure
  - Data consistency and integrity problems



https://candost.blog/the-blue-green-deployment-strategy/

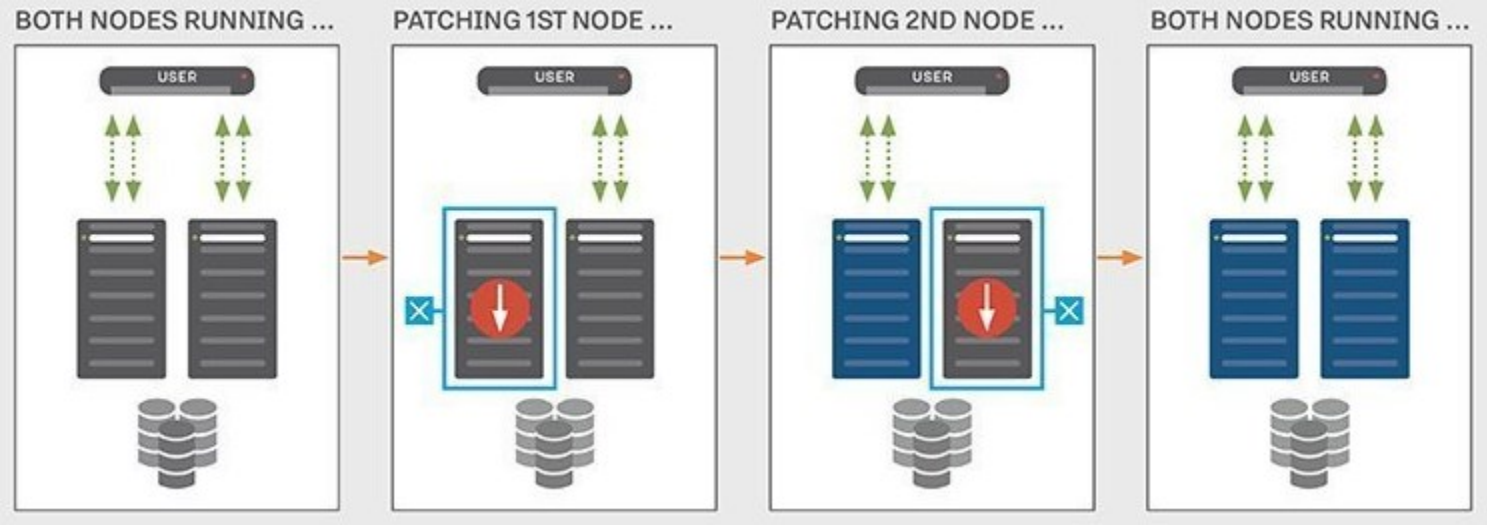
ROLLING DEPLOYMENT

- A rolling deployment is a software release strategy that staggers deployment across multiple phases, which usually includes one or more servers performing functions within a **server cluster**.
- Rather than updating all servers simultaneously, the organization installs the updated software package on one server or subset of servers at a time.



ROLLING DEPLOYMENT

- In a rolling deployment, the organization can take one of the nodes **offline**, with the load balancer configured to direct traffic to the remaining servers still running the current software version.
- The idle servers receive updates and testing; the remaining online servers support user traffic.

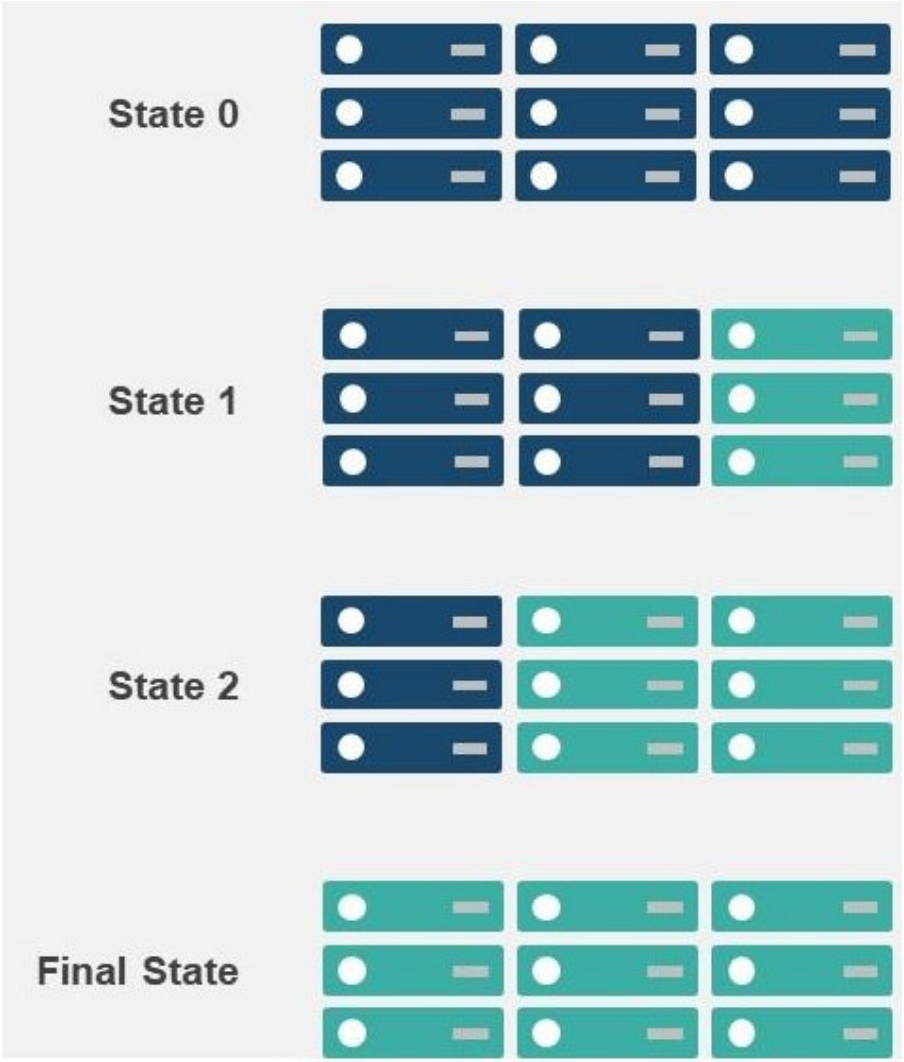


https://www.techtarget.com/searchitoperations/definition/rolling-deployment

ROLLING DEPLOYMENT

- Benefits:
  - Less infrastructure cost and maintenance efforts
  - Early detection of problems

- Drawbacks:
  - A significant latency between the moment you start deploying the new version and the moment it is all live.
  - If problems slip past initial checks, they'll propagate as more servers are updated, making rollback difficult
  - Can't control which users get the new version



Canary release (greyscale release)

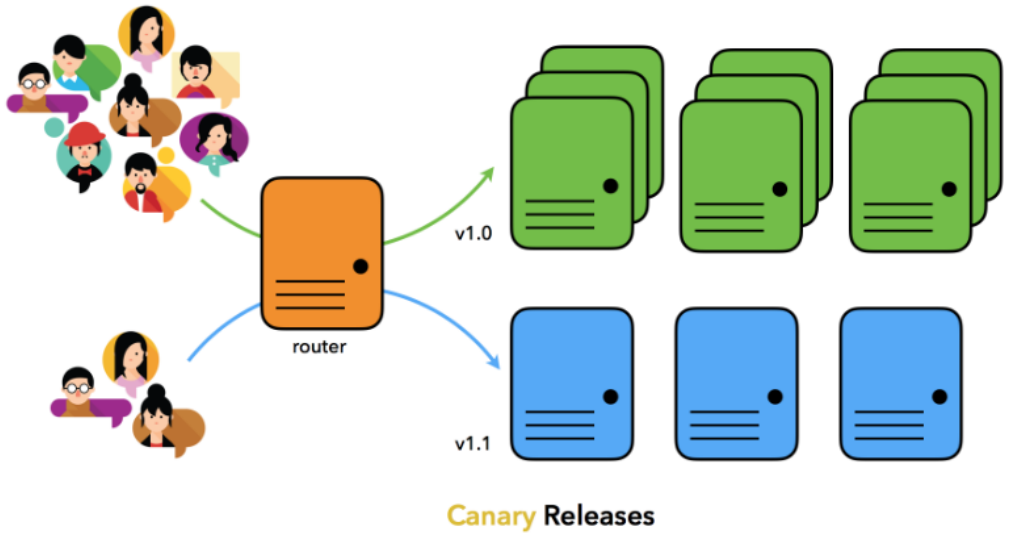
- “Canary releases” get their name from an old coal mining tactic. Miners would release canaries into coal mines in an attempt to gauge the amount of toxic gases present. If the canary survived, things were safe.

- When it comes to software development and new releases, a canary deployment substitutes a bird for software users (although hopefully in a much safer way), wherein the **pre-selected group** of users helps you identify bugs, breaks, and hazards of a new feature.

- With the information collected from the selected user group (canaries), teams can then strengthen the feature so it’s ready for a wider audience.

CANARY RELEASE (GREYSCALE RELEASE)

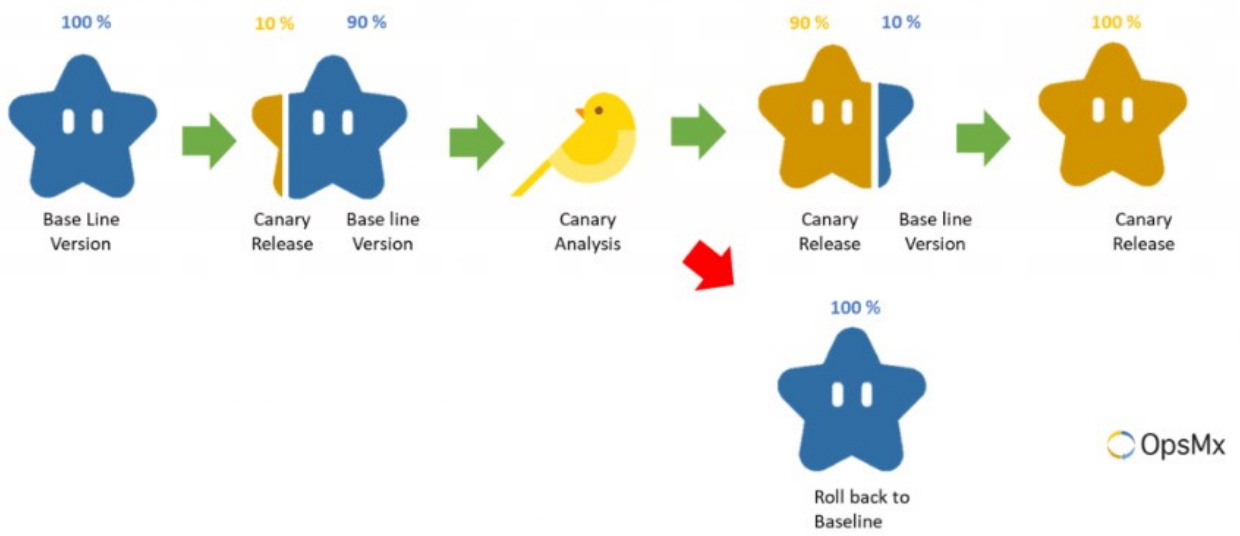
- A canary release is when you make new software features available to **a limited selection of users** ahead of a wider release.
- Sometimes the users selected to participate in canary releases are targeted for specific reasons, such as location or device type, but other times it can be completely random.



https://gowrishankar.info/blog/istio-service-mesh-canary-release-routing-strategies-for-ml-deployments-in-a-kubernetes-cluster/

Canary release (greyscale release)

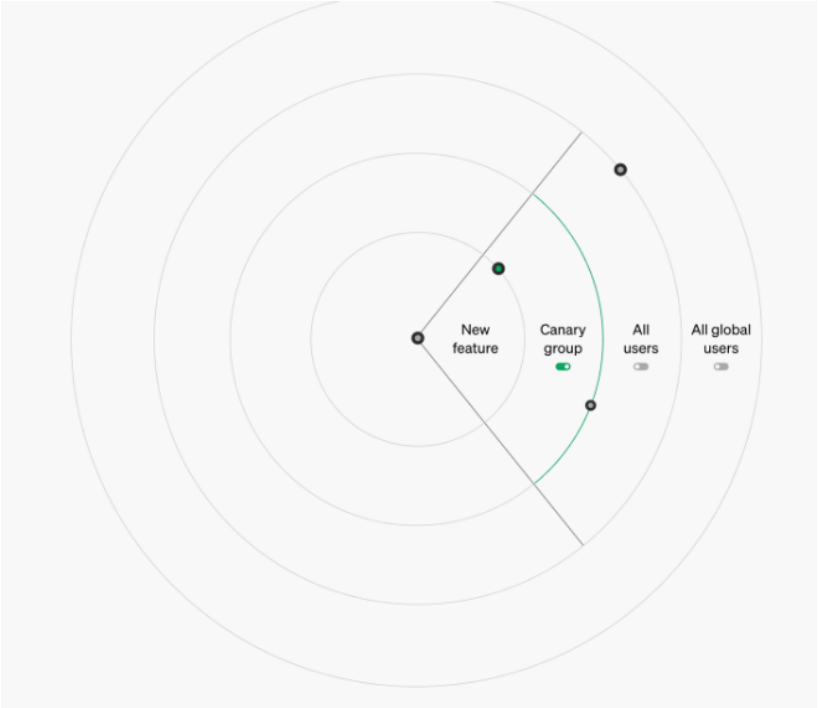
- A small percentage uses the canary version, while the rest of the users continue using the old version.
- Based on feedback and performance, all users can be gradually migrated to either canary version or rolled back to the old version.
- If problems occur, the majority of users won't be affected.



https://www.opsmx.com/blog/what-is-canary-deployment/

Canary release (greyscale release)

- A small percentage uses the canary version, while the rest of the users continue using the old version.
- Based on feedback and performance, all users can be gradually migrated to either canary version or rolled back to the old version.

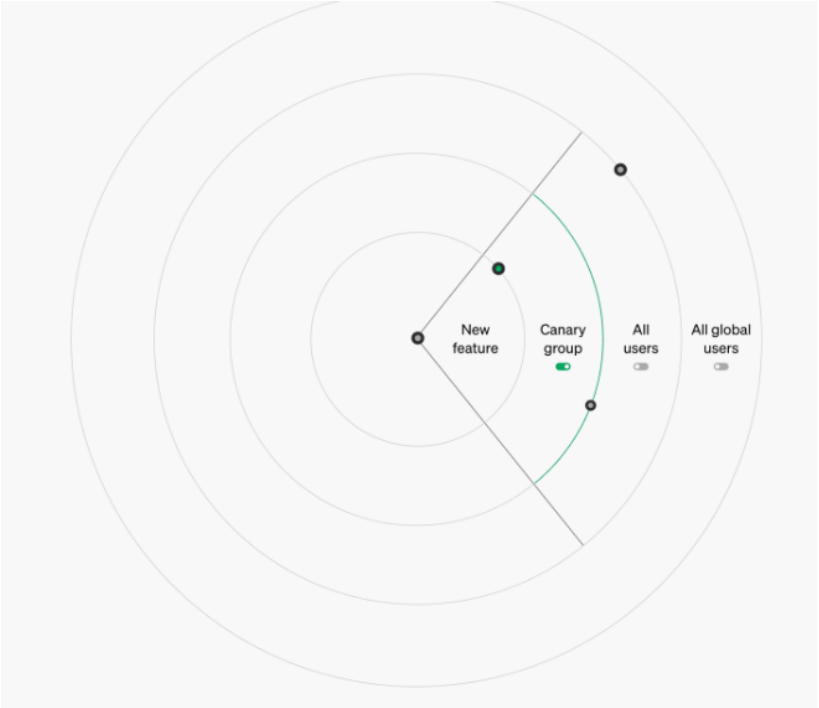


https://www.opsmx.com/blog/what-is-canary-deployment/

Canary release (greyscale release)

- Benefits
  - Test the new version with real users
  - Identify bugs or performance issues before releasing to a wider audience. In case of failure, only a small number of users is affected
  - Rollback is simple and quick

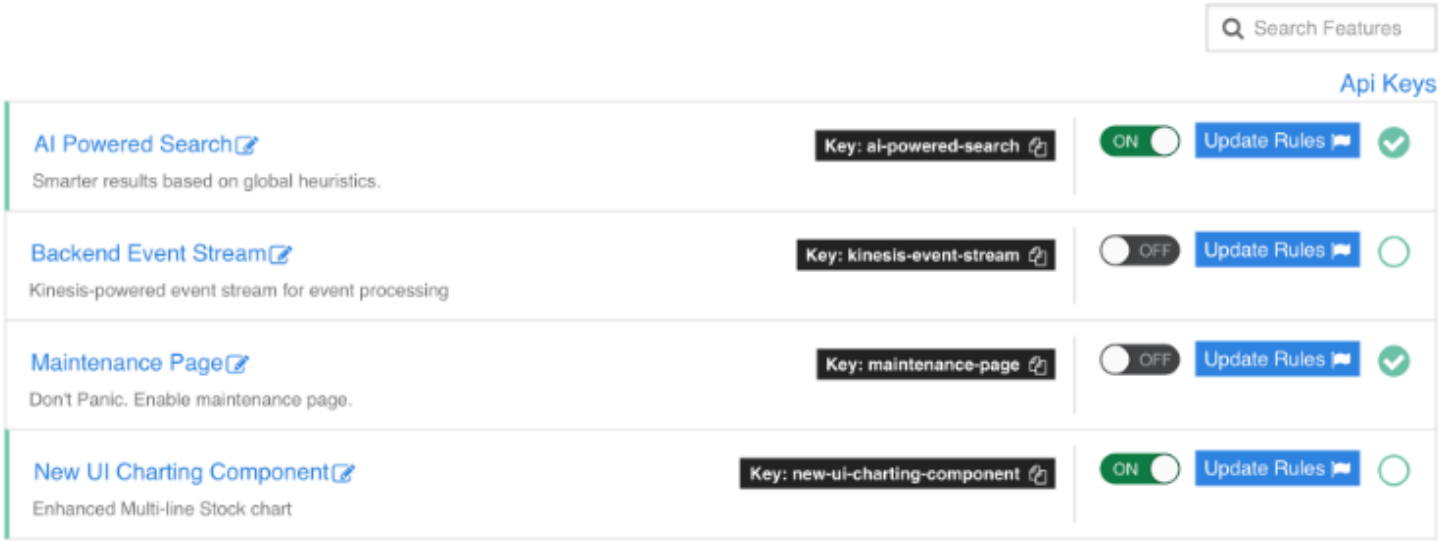
- Drawbacks
  - Complex implementation & infrastructure mechanism (e.g., API compatibility, DB integrity, user verification, etc.)



https://www.opsmx.com/blog/what-is-canary-deployment/

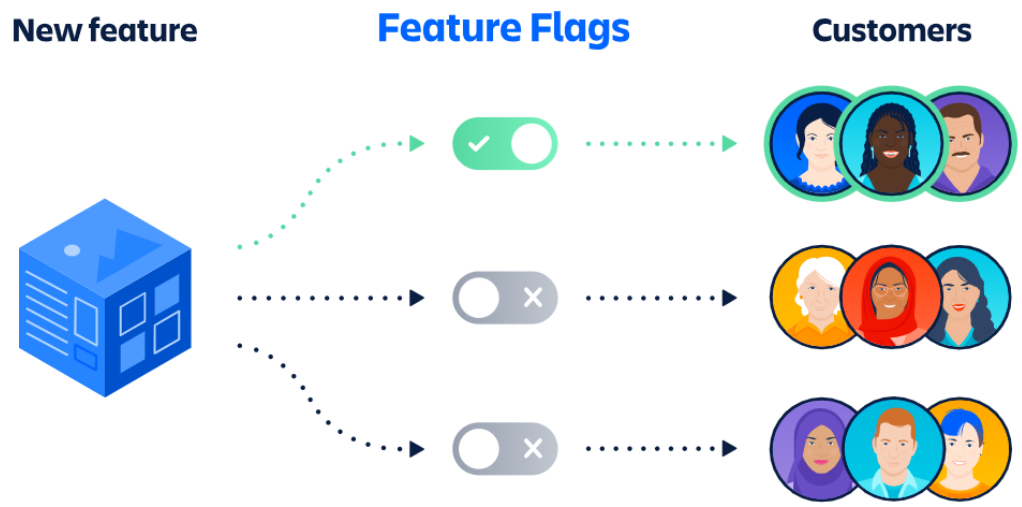
FEATURE TOGGLE

Feature Toggles (Feature Flags) are a powerful technique, allowing teams to modify system behavior without changing code.



FEATURE TOGGLE

Feature Toggles could be used for different user groups and different environments.

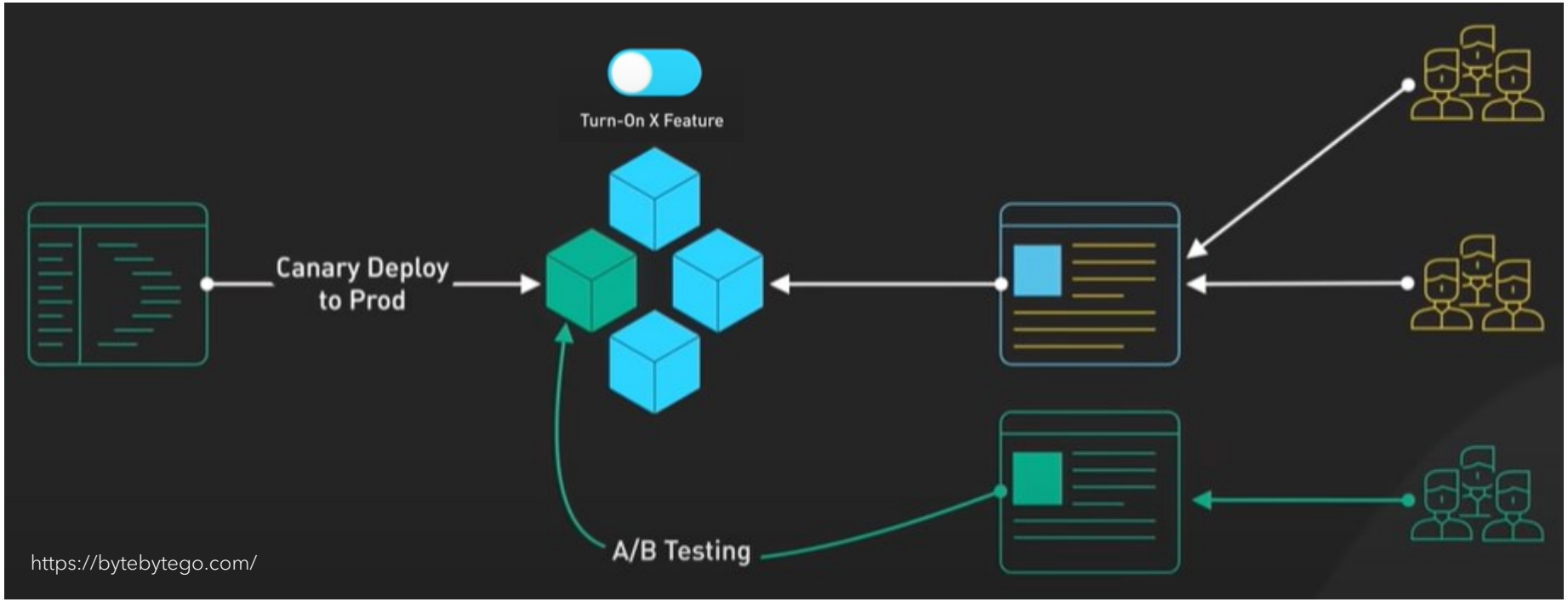


https://www.atlassian.com/solutions/devops/integrations/feature-flags

FEATURE KEY	DEV	PROD
covid-banner	on	on
experiment-implementation	on	off
feature-focus	on	off
feature-usage-code	on	on
papercups-enabled	on	off
show-experiment-groups	on	off
show-experiment-targeting	on	off

FEATURE TOGGLE

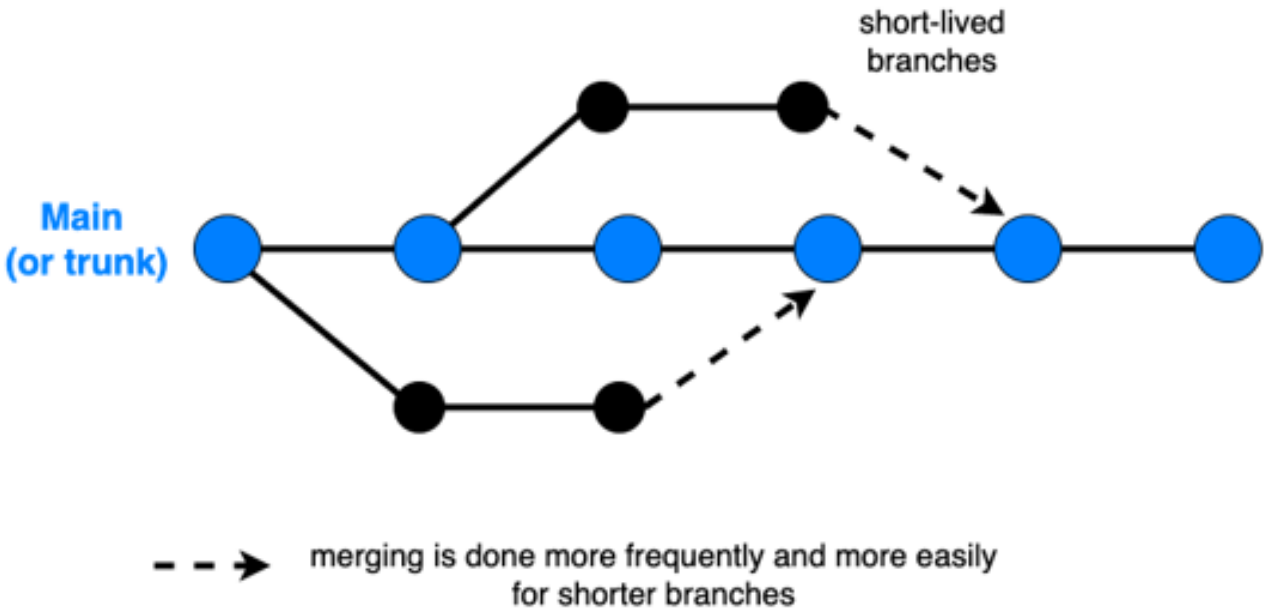
Feature Toggles can be used in combination with any deployment strategies.



BRANCHING STRATEGY

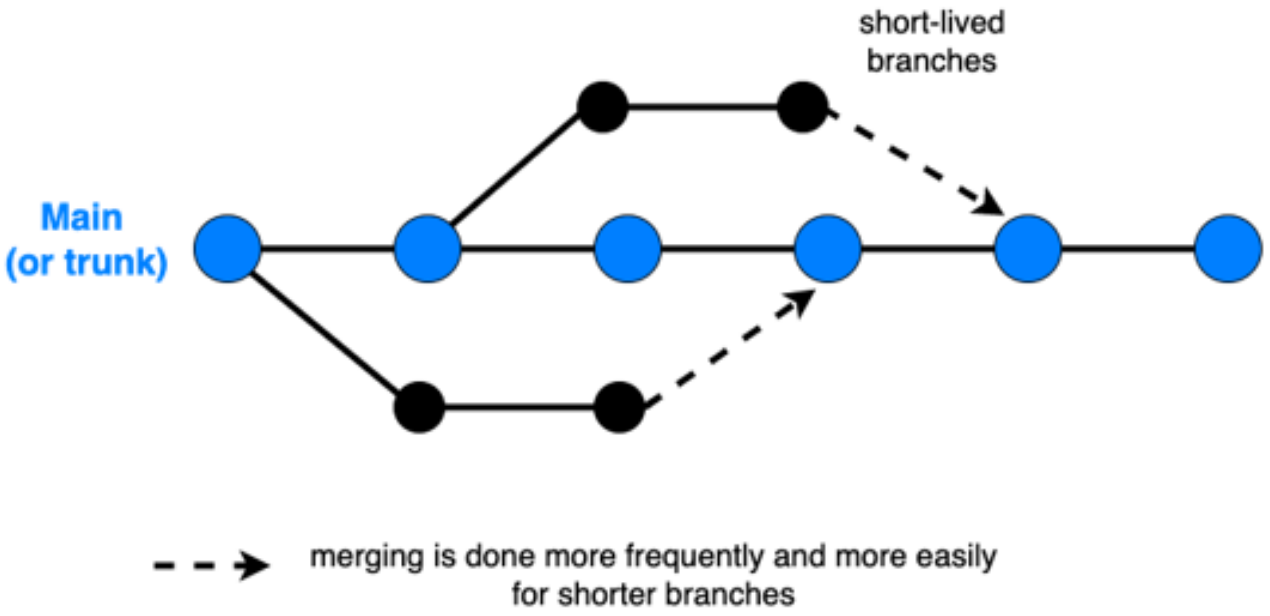
- Trunk-based dev & release（主干开发，主干发布）
- Trunk-based dev & branch-based release（主干开发，分支发布）
- Branch-based dev & trunk-based release（分支开发，主干发布）

TRUNK-BASED DEV & RELEASE



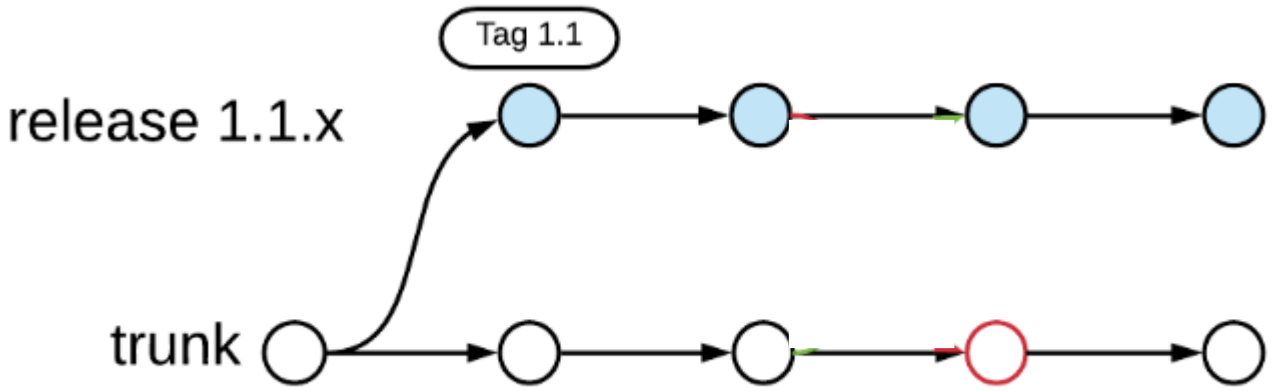
- Trunk Based Development has each developer work locally and independently on their project, and then merge their changes back into the main branch
- Main brunch is directly used for deployment as release

TRUNK-BASED DEV & RELEASE



- Benefits
  - Less management efforts for branches
- Drawbacks
  - Unfinished work affects release
  - Hard to merge branches back in case of high-frequency deployment & rapid release

Trunk-based dev & branch-based release

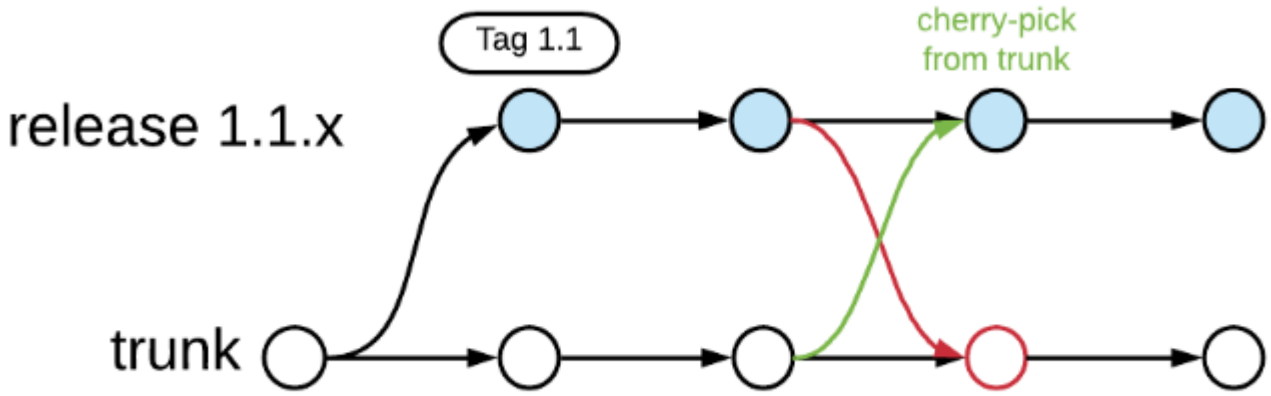


Meanwhile, all dev work still happens on the trunk

- The decision is made to release a state of the trunk
- A release branch is created, starting from the commit representing that state of the trunk
- Potentially, some very limited work happens in the release branch in order to fully make it release-ready
- The code is released and the released commit is tagged

https://convincedcoder.com/2019/02/16/Trunk-based-development/

TRUNK-BASED DEV & BRANCH-BASED RELEASE



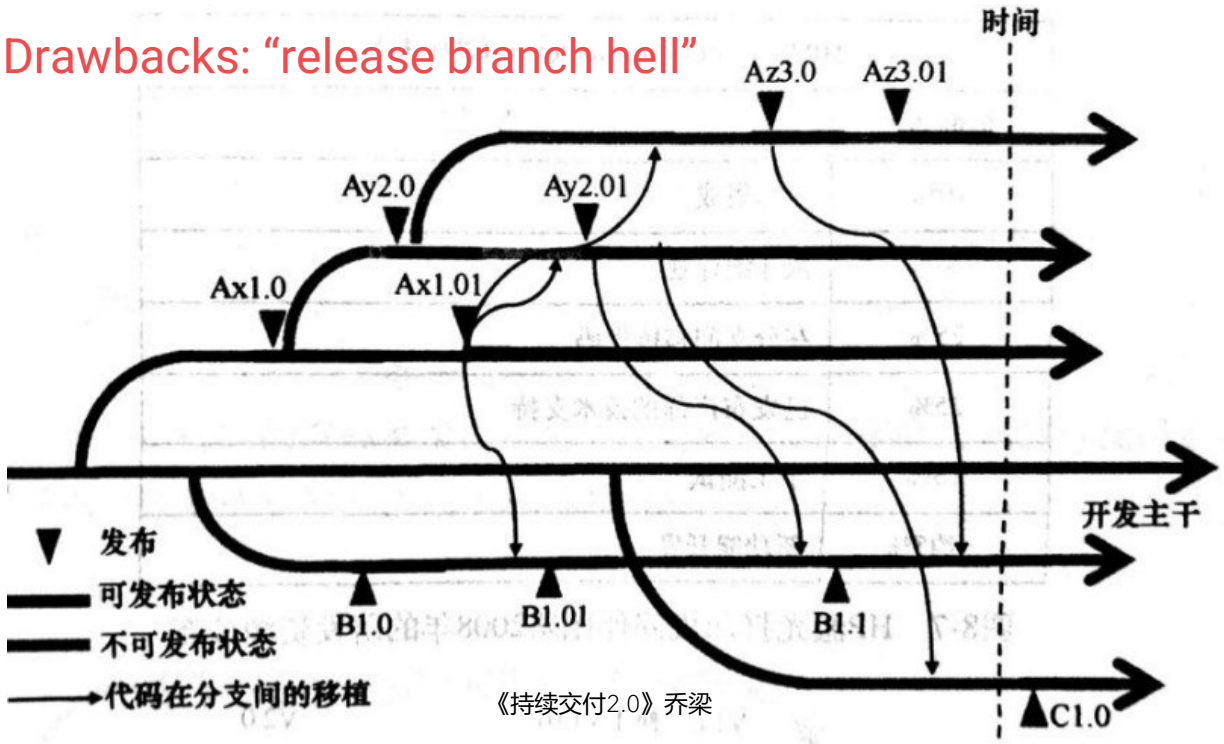
Benefits: devs stay on trunk and are less affected by release; release won't be affected by unfinished work on trunk

- If the release contains bugs, fix the bugs on the trunk and cherry-pick them into the release branch
- Not the other way around: This helps prevent regression bugs caused by applying a fix in a release branch but forgetting to apply the fix on the trunk as well. Exceptions are only allowed if it is really not possible to reproduce the bug on the trunk.
- The code is released and the released commit is tagged

https://convincedcoder.com/2019/02/16/Trunk-based-development/

TRUNK-BASED DEV & BRANCH-BASED RELEASE

Drawbacks: “release branch hell”

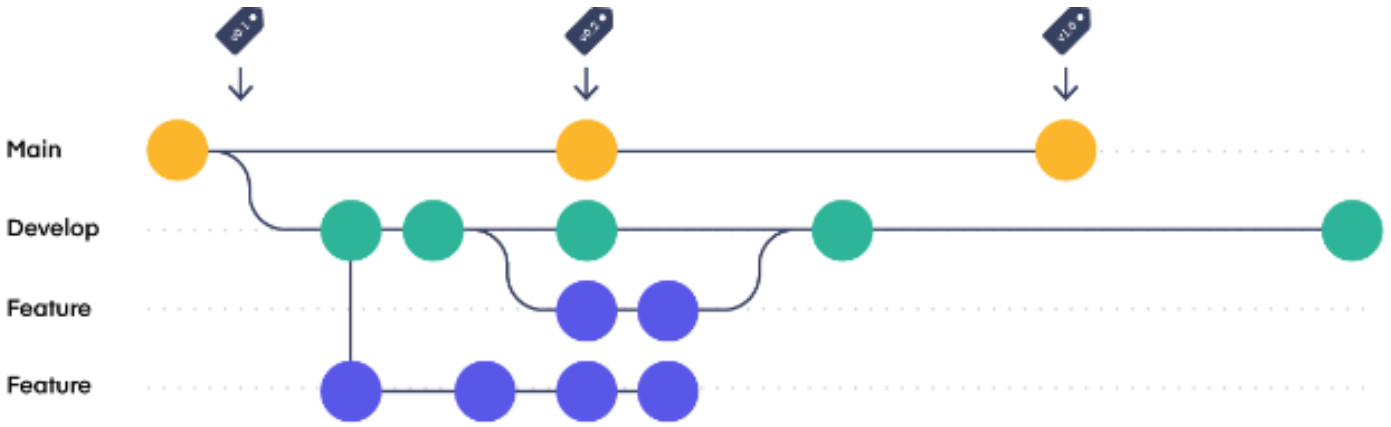


《持续交付2.0》乔梁

Time	Work
10%	Code integration
20%	Plan
25%	Port code between branches
25%	Tech support for released product
15%	Manual testing
5%	New feature development

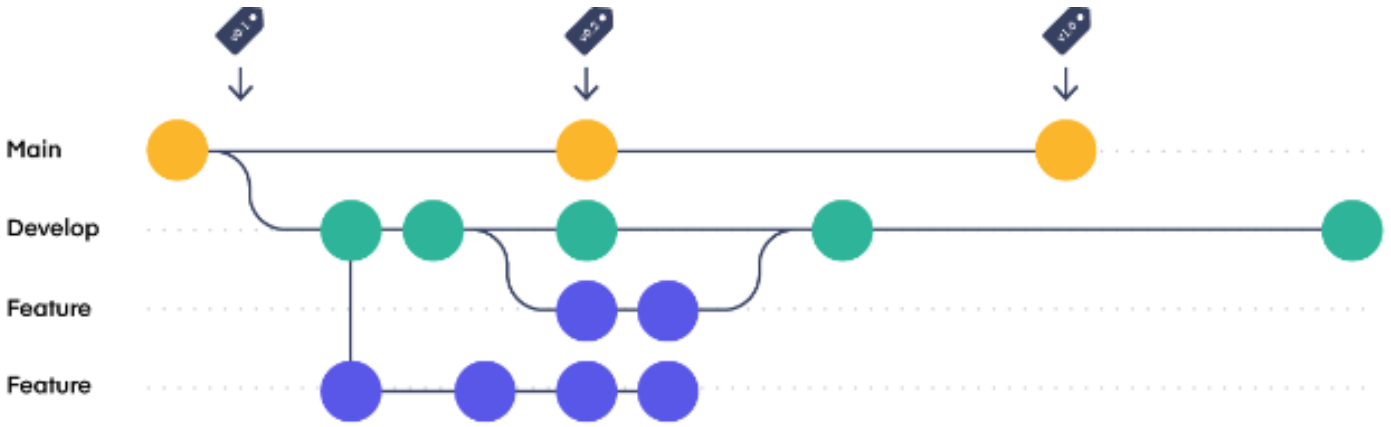
HP LaserJet Firmware team (2008)

BRANCH-BASED DEV & TRUNK-BASED RELEASE



- In the Feature Branch workflow, main represents the official project history.
- Instead of committing directly on their local main branch, developers create a new branch every time they start work on a new feature.
- Feature branch is merged back to main branch once finished and tested.
- Release is done on the main branch.

BRANCH-BASED DEV & TRUNK-BASED RELEASE



Benefits

- Independent development on branches
- Teams can choose which features to be released.
- If bugs occur, fixes can be done directly on the main branch or on a new hotfix branch, without affecting other feature branches.

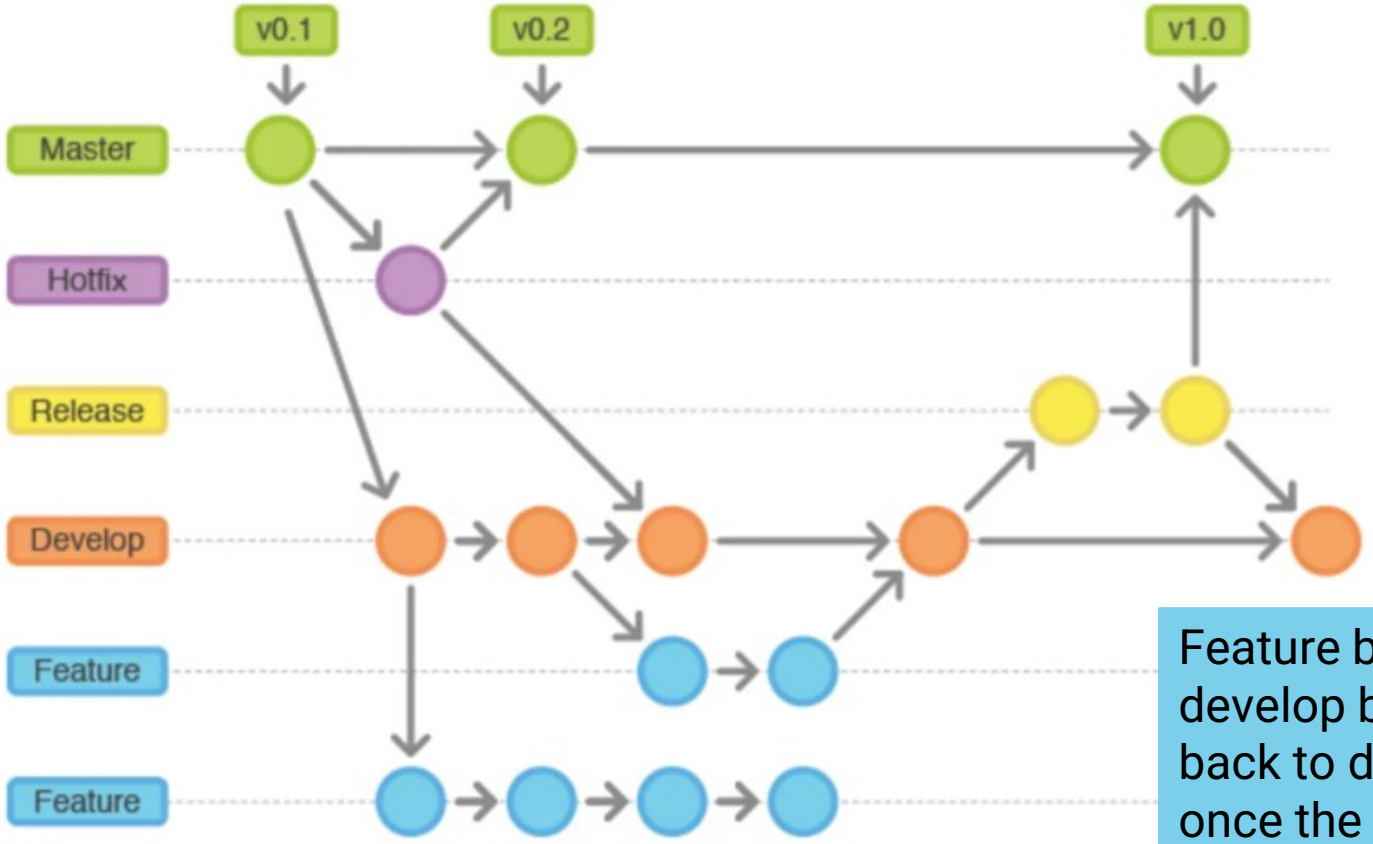
Drawbacks

- Merge efforts on many branches

GITFLOW BRANCHING STRATEGY

Master: branch for official releases

Develop: branch for feature integration

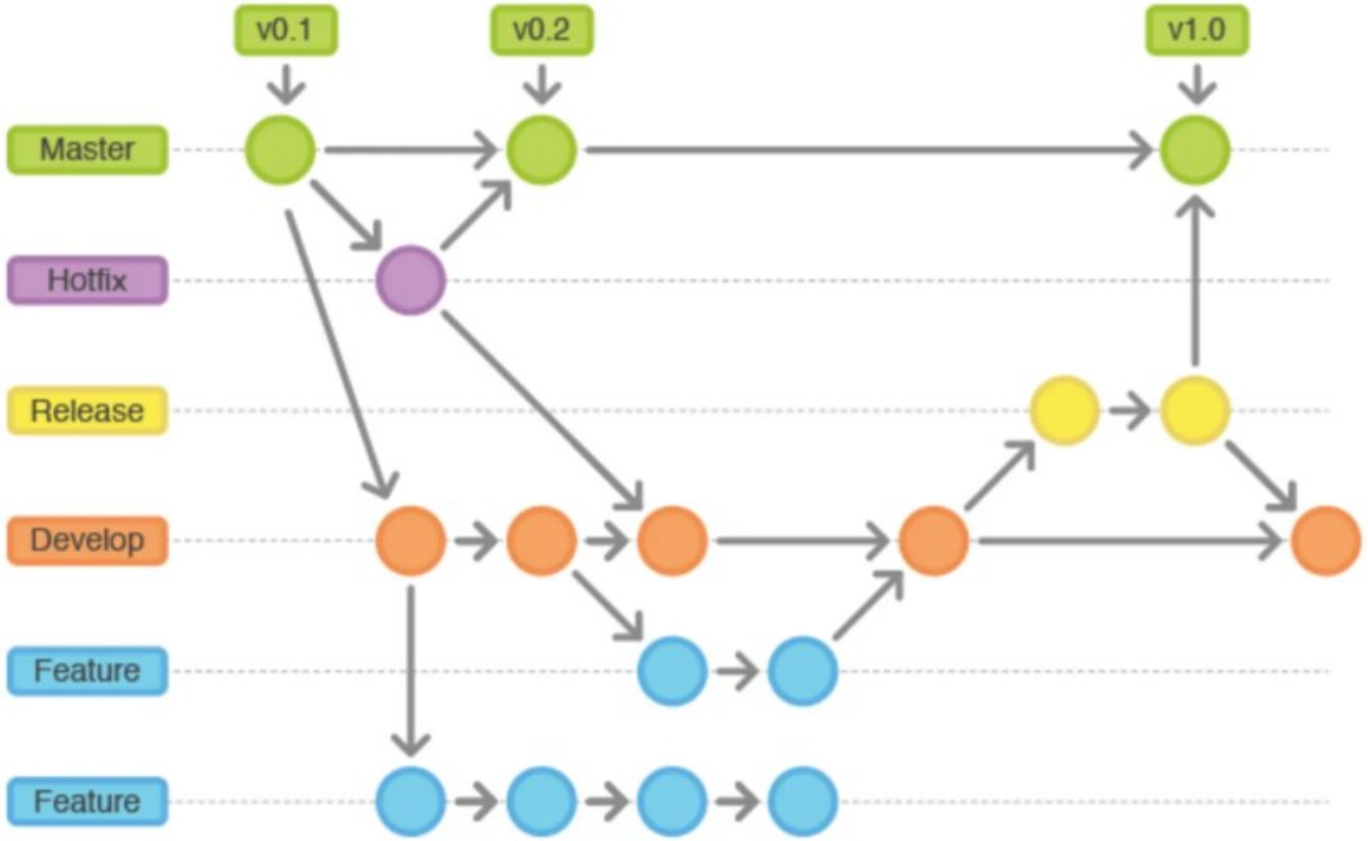


Feature branch: pull from develop branch; merge back to develop branch once the feature is done.

GITFLOW BRANCHING STRATEGY

Release: branch for pre-release

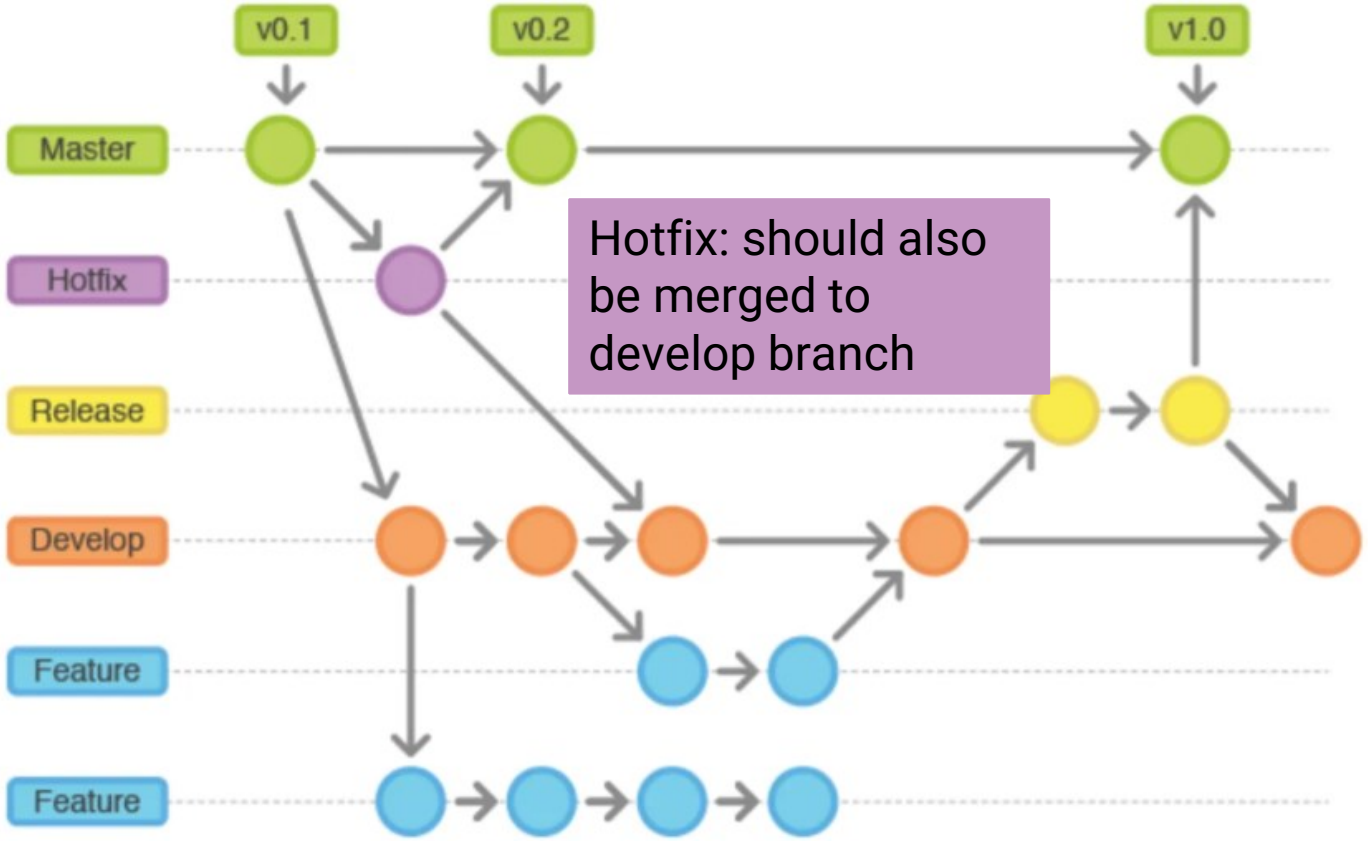
- only bug fixes, documentation, etc.
- Not used for new feature development.



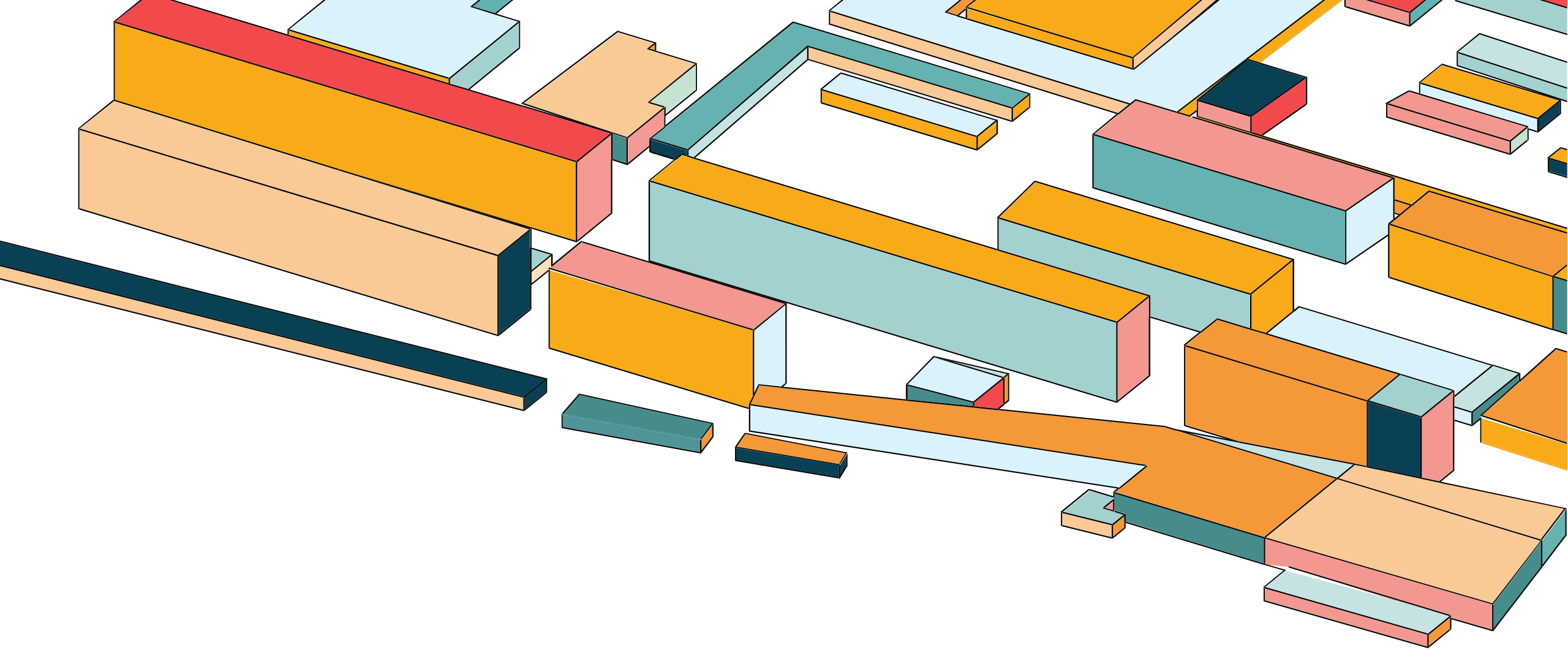
Release branch: Merged back to master and develop branch once passed QA (and optionally, Alpha & Beta releases)

GITFLOW BRANCHING STRATEGY

Hotfix: if v0.1 has a bug, create a hotfix branch for quick fixing, then release as patch v0.2

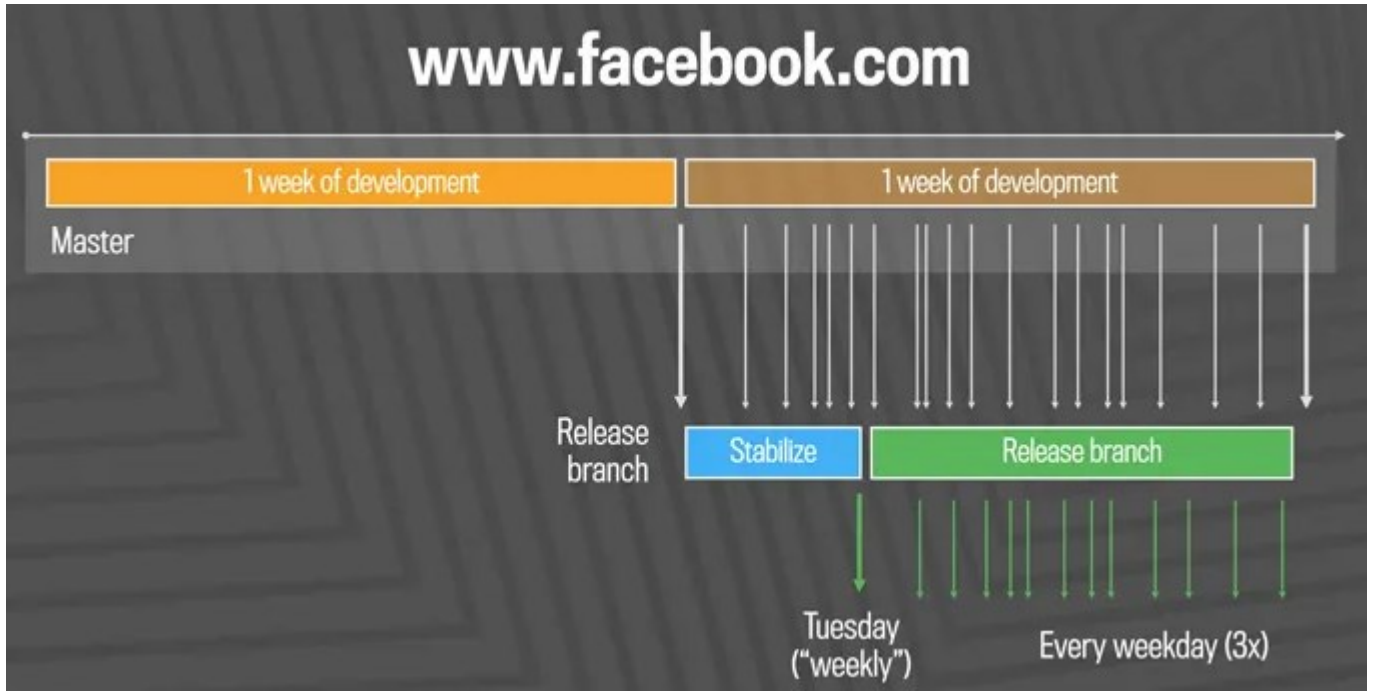


Hotfix: should also be merged to develop branch



CASE STUDY

CI/CD @ META

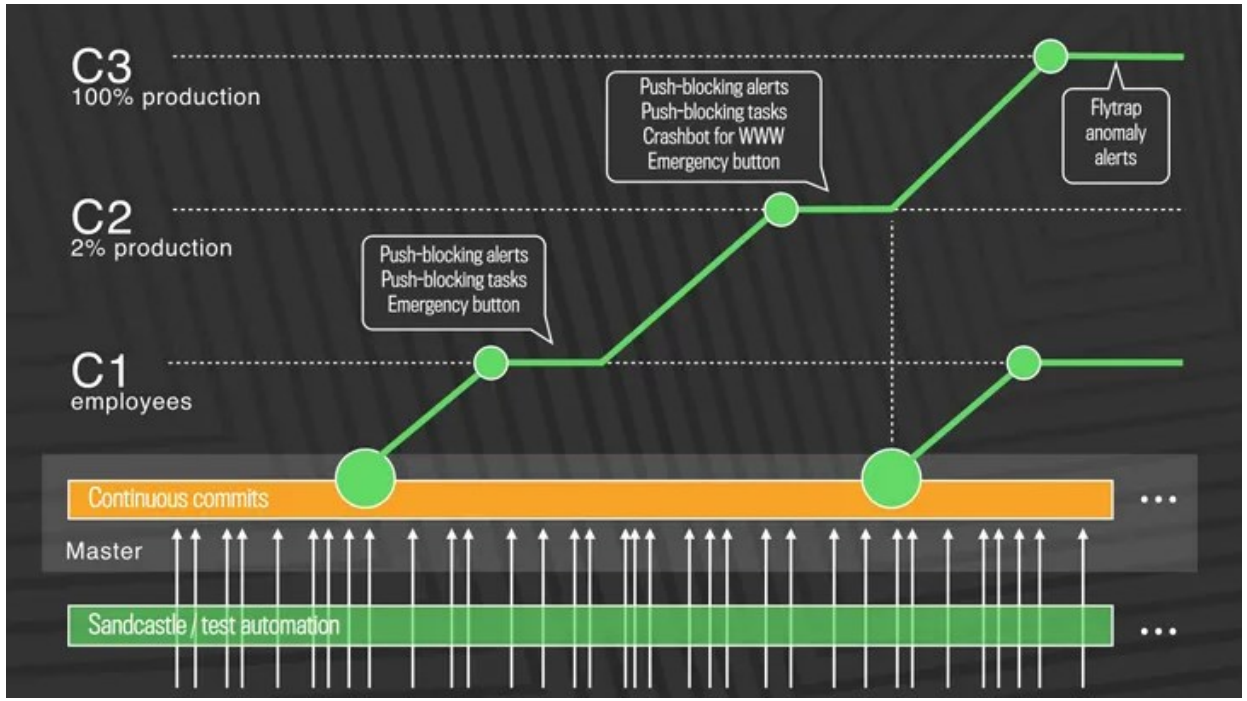


Before 2016

- Master and release branch strategy
- Engineers would request cherry-picks — changes to the code that had passed a series of automated tests — to pull from the master branch into one of the daily pushes from the release branch.

https://engineering.fb.com/2017/08/31/web/rapid-release-at-massive-scale/

CI/CD @ META

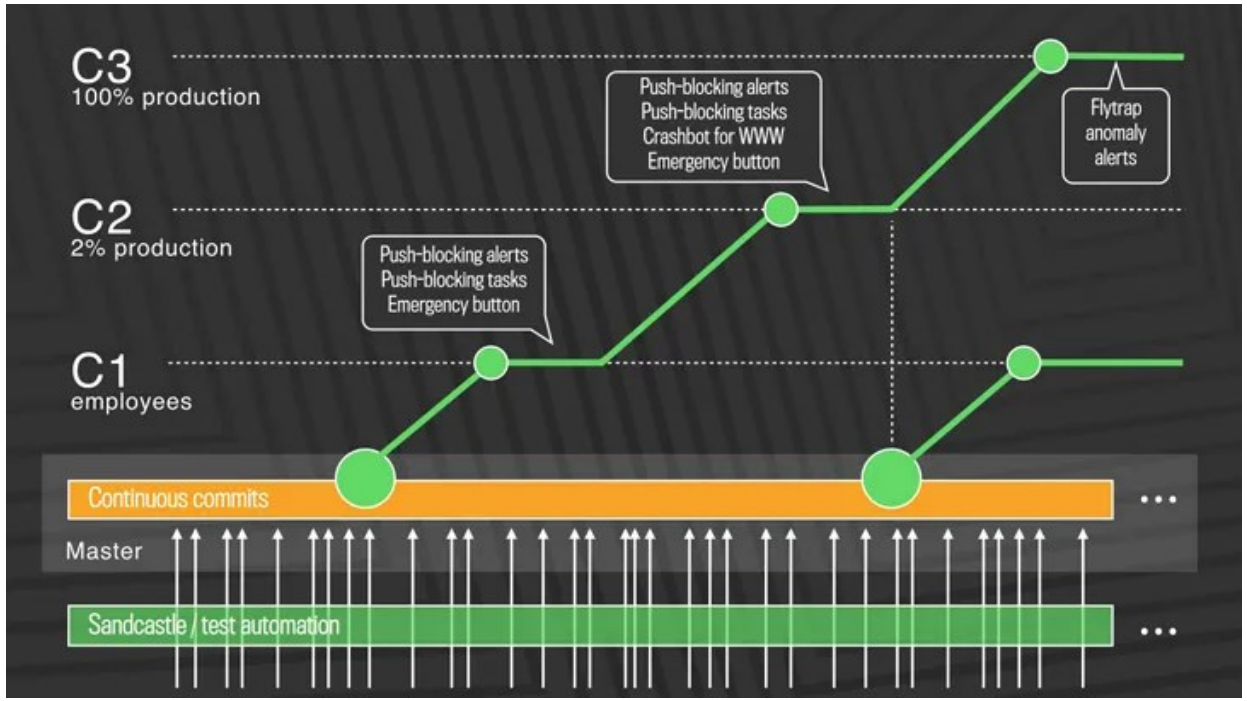


After 2016

- “Push from master” strategy

https://engineering.fb.com/2017/08/31/web/rapid-release-at-massive-scale/

CI/CD @ META

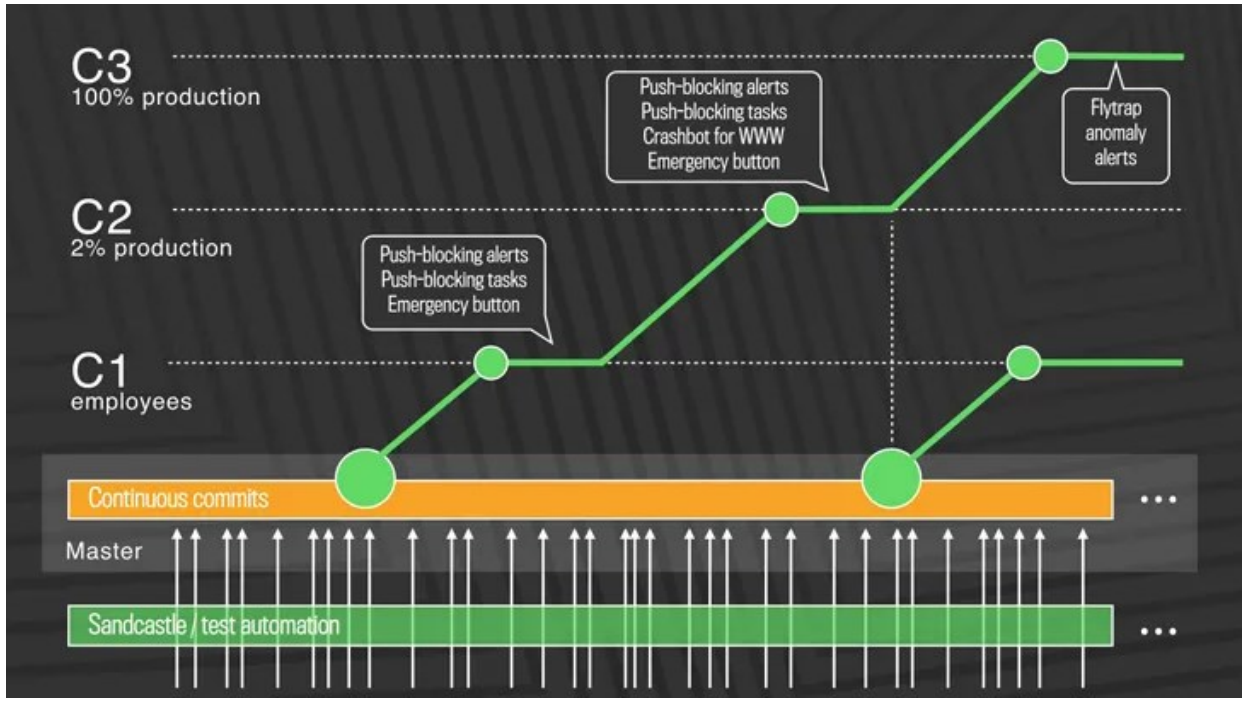


C1

- First, diffs that have passed a series of automated internal tests and land in master are pushed out to Facebook employees.
- In this stage, we get push-blocking alerts if we've introduced a regression, and an emergency stop button lets us keep the release from going any further.

https://engineering.fb.com/2017/08/31/web/rapid-release-at-massive-scale/

CI/CD @ META

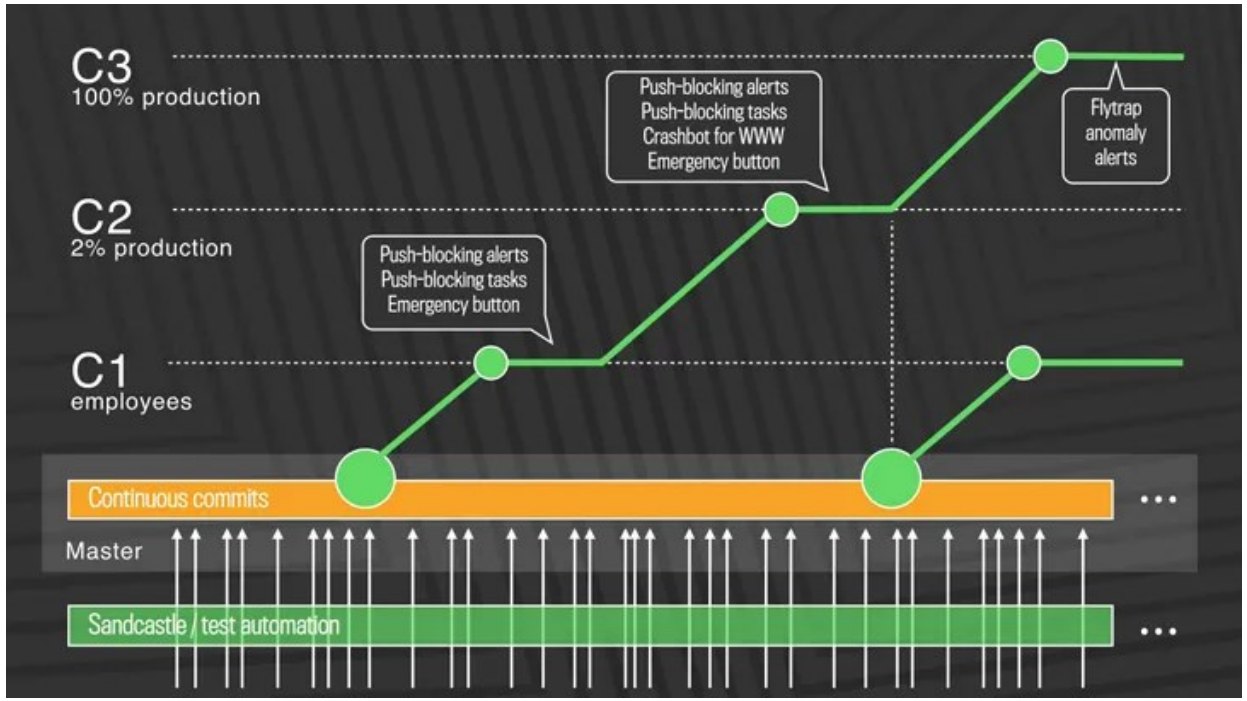


C2

- If everything is OK, we push the changes to 2% of production
- Again, we collect signal and monitor alerts, especially for edge cases that our testing or employee dogfooding may not have picked up.

https://engineering.fb.com/2017/08/31/web/rapid-release-at-massive-scale/

CI/CD @ META



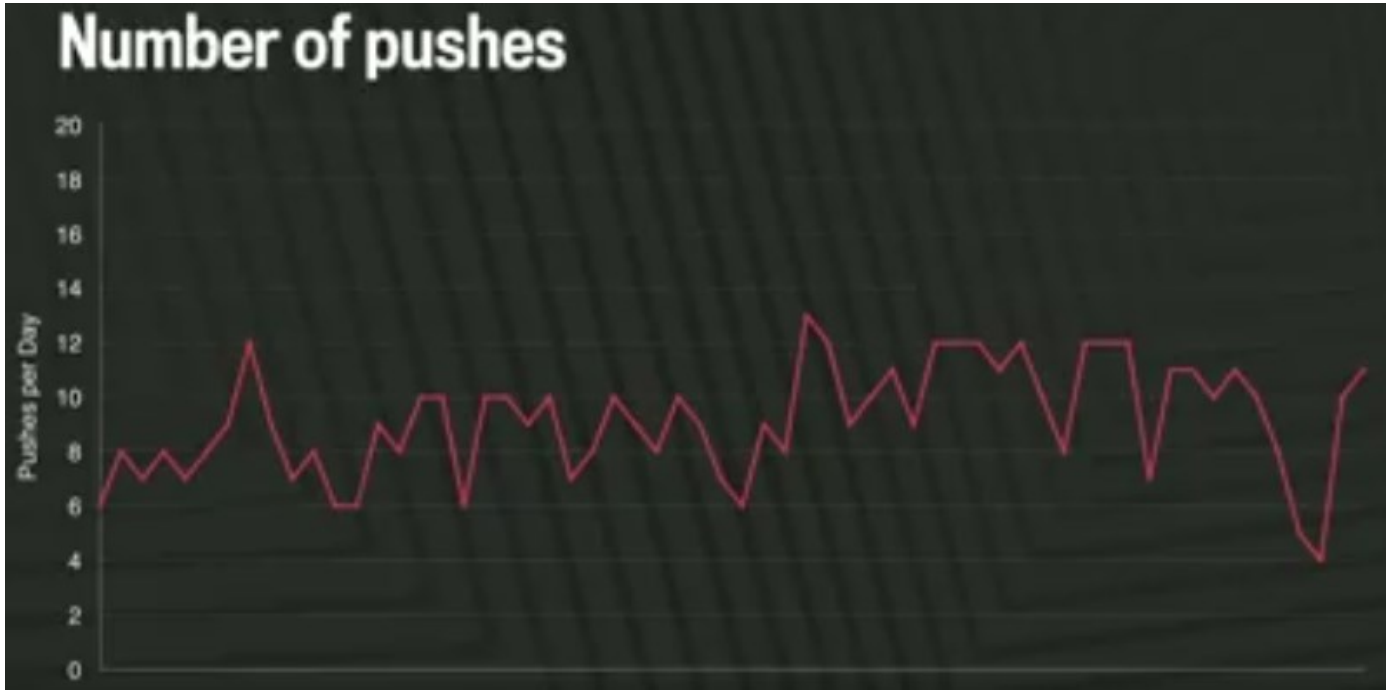
C3

- Finally, we roll out to 100 percent of production, where our Flytrap tool aggregates user reports and alerts us to any anomalies.

Each release is rolled out to 100 percent of production in a tiered fashion over a few hours, and the team can stop the push if they find any problems.

https://engineering.fb.com/2017/08/31/web/rapid-release-at-massive-scale/

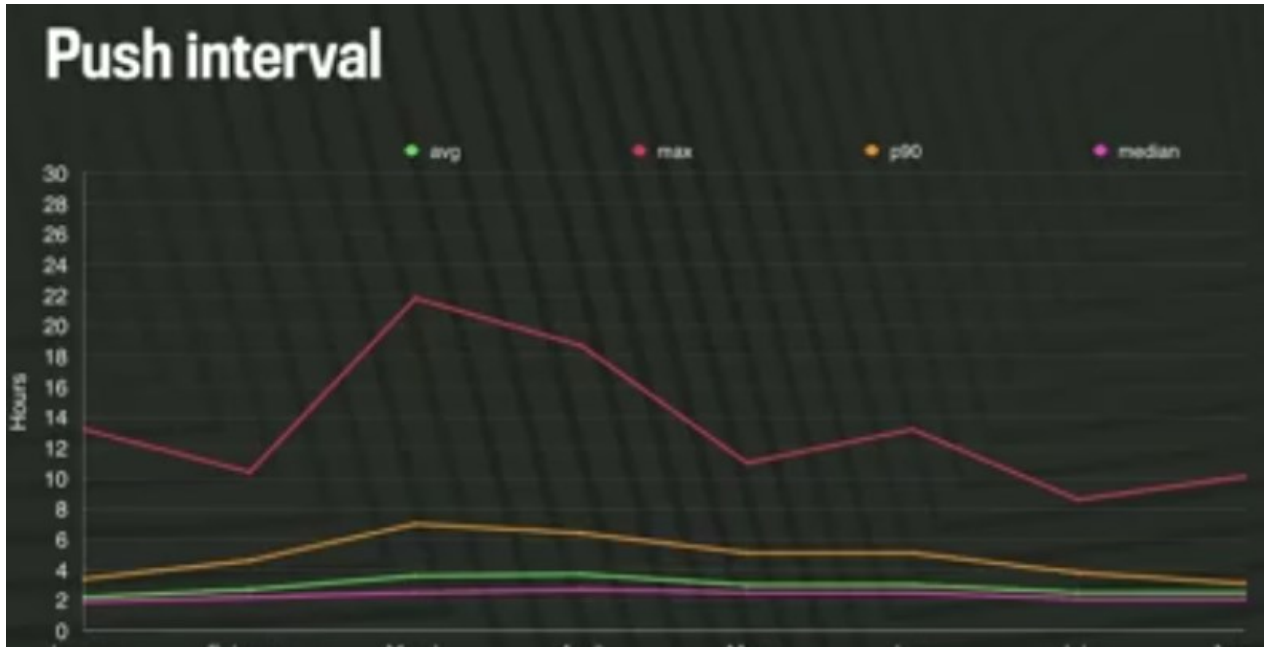
CI/CD @ META



# of daily pushes

https://engineering.fb.com/2017/08/31/web/rapid-release-at-massive-scale/

CI/CD @ META



As a developer, how long do I need to wait before my changes go public?

https://engineering.fb.com/2017/08/31/web/rapid-release-at-massive-scale/

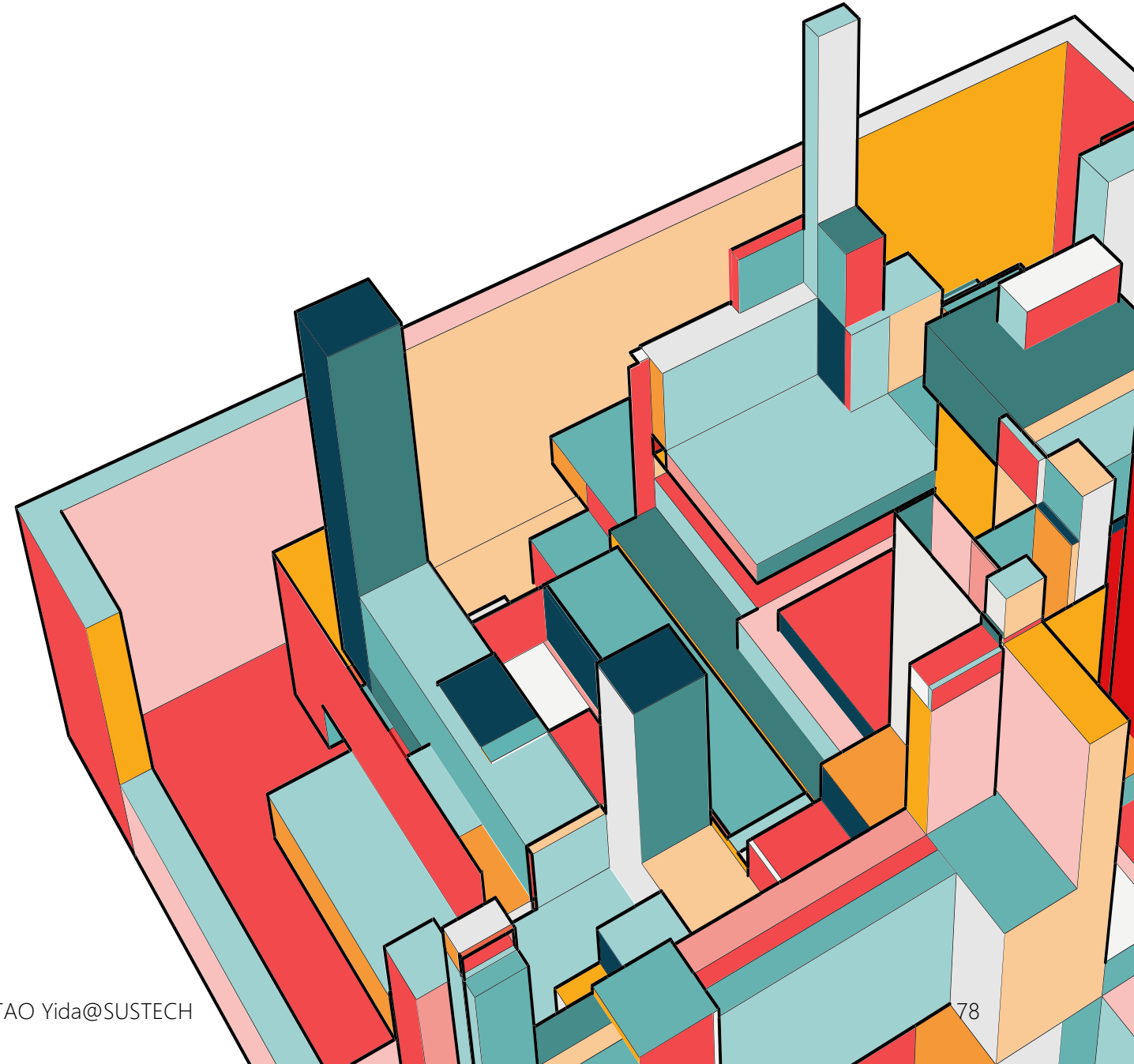
CI/CD @ META

As a developer, how do I know what's going on in the pipeline?

https://engineering.fb.com/2017/08/31/web/rapid-release-at-massive-scale/

READINGS

- 《持续交付2.0: 业务引领的DevOps精要》乔梁 著
- Continuous Delivery: Reliable Software Release through Build, Test, and Deployment Automation. Jez Humble and David Farley.
- Chapter 23-24. Software Engineering at Google by Winters et al.



NEXT

- Software infrastructure
- Cloud-native software