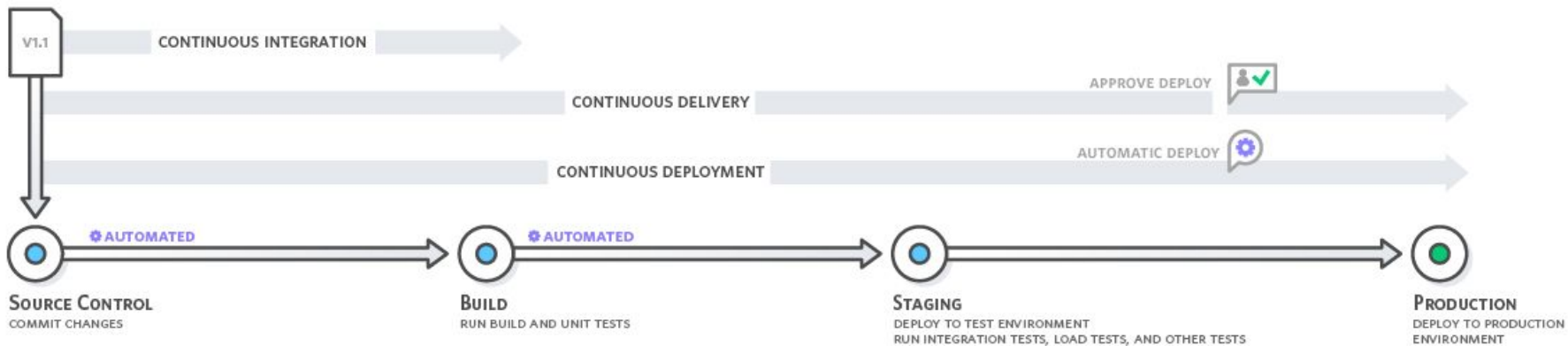


Continuous Integration Continuous Deployment

Assignment 3 Released (due 2/27)

Final Project Handout Released (due 3/17)

*I pushed some new code
to my Git repository! How
do I get it into production?*



*...is your code in a state that can
integrate with the production codebase?*

This branch is 46 commits ahead of, 24 commits behind `main` .

```
~/Documents/GitHub/infracourse/web-private-demo git:(main) 09:40 pm (0.027s)
```

```
git merge origin/project-work
```

```
fatal: refusing to merge unrelated histories
```

```
~/Documents/GitHub/infracourse/web-private-demo git:(main) 09:41 pm (0.078s)
```

```
git merge origin/a2-cody-edits
```

```
Auto-merging content/assignments/2.md
```

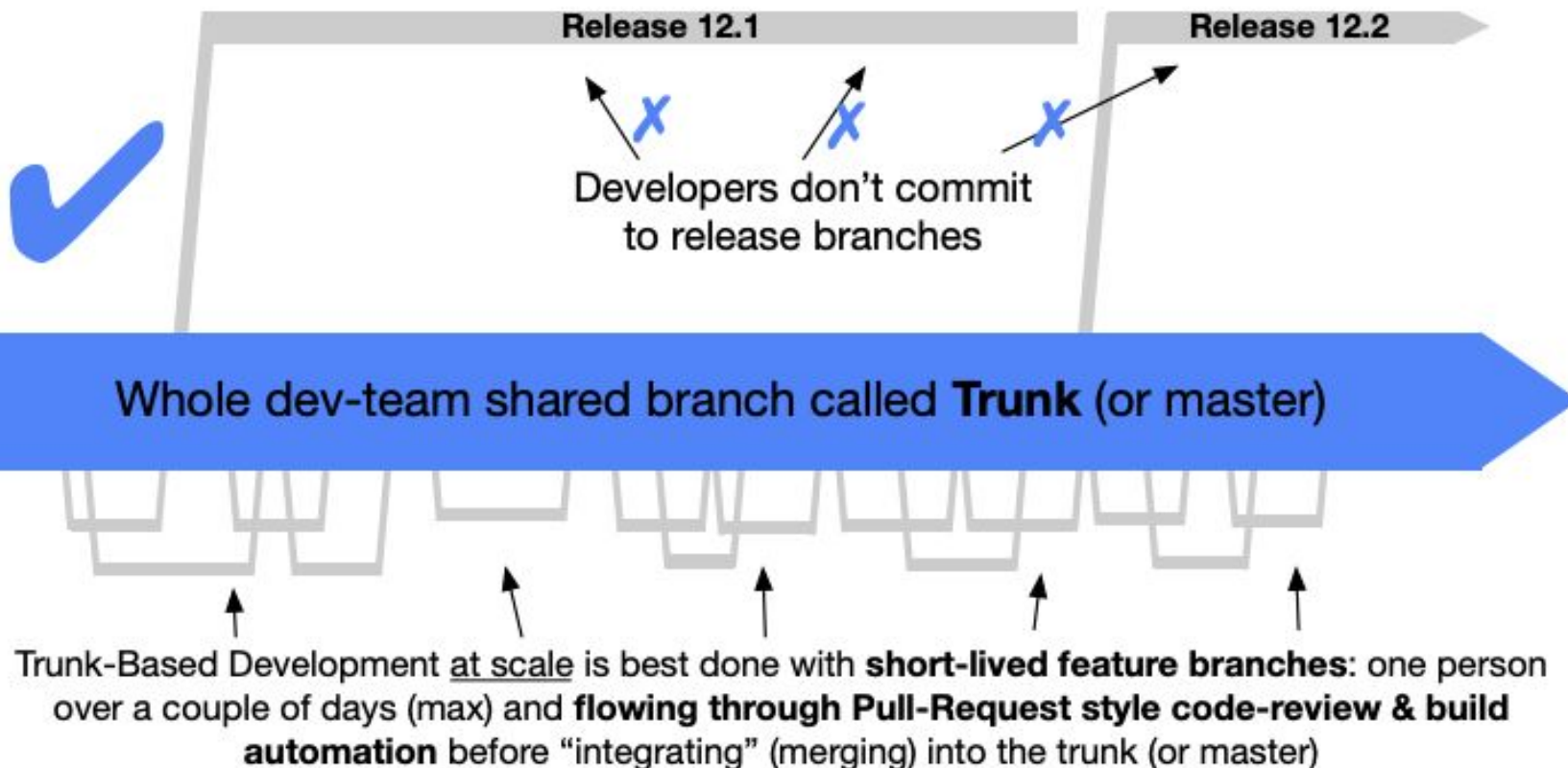
```
CONFLICT (content): Merge conflict in content/assignments/2.md
```

```
Automatic merge failed; fix conflicts and then commit the result.
```

Continuous Integration

General best practices surrounding clean codebase handling.

1. Ensure the main branch is always a working, production-ready software state
2. Ensure development branches track closely with the main branch
3. Build and test every candidate change to the main branch



Paul Hammant, Trunk-Based Development, 2013

*No merge conflicts! But does
your code actually work?*

Types of Production Software Testing

- **Unit testing:** Test that individual functions work as expected
 - e.g. for a function `sum(a, b) → c`, does `sum(1, 1)` return 2?
 - *Can be trivially automated*
- **Integration testing:** Test that components of an application work together
 - e.g. if a candidate change targets service A, does service B that needs data from A still work?
 - *Can be automated, but is more technically involved*
- **End-to-end testing:** Test that the entire application works as intended
 - e.g. if a candidate change targets service A, does the entire application still work?
 - *Can be automated, but is **very** technically involved*

*Automated testing requires
cloud infrastructure.*

Deployment Environments

- **Development environment:** Local (e.g. development laptop, virtualized codespace) testing for quick iteration
- **Staging environment:** Mirror of the production environment (i.e. latest mainline code changes)
 - Automated tests happen here
- **Production environment:** Live environment served to users
 - Availability and security is critical; should be handled cleanly and separately from dev

Mechanics of Automated Testing

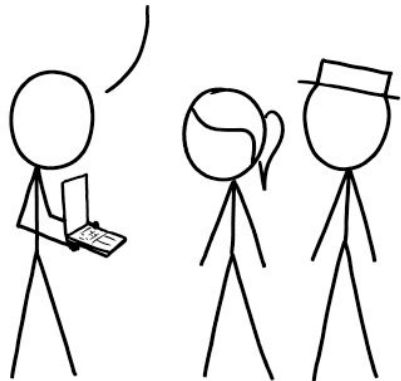
1. Watch for mainline candidate changes (usually PRs and commits to main)
2. Trigger build of software with new changes
 - a. Binary executable if compiled language
 - b. Usually a container build regardless
 - c. Only need to build the service the change targeted, if using microservices
3. Run tests and output results
 - a. Block a merge to mainline, or further deployment, if failures found

Hosted CI Platforms

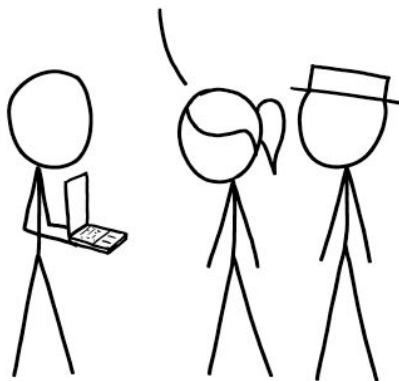
- Open source, self-hosted CI: Jenkins (2011)
 - Mostly intended for package releases rather than web services
 - Define build and test pipelines through UI or through Groovy script language
 - Unreliable, needs many third-party plugins to work, bad UI/UX
 - **Security:** remote code execution as a feature (e.g. January 2023 no-fly list breach)



CHECK IT OUT—I MADE A FULLY AUTOMATED DATA PIPELINE THAT COLLECTS AND PROCESSES ALL THE INFORMATION WE NEED.

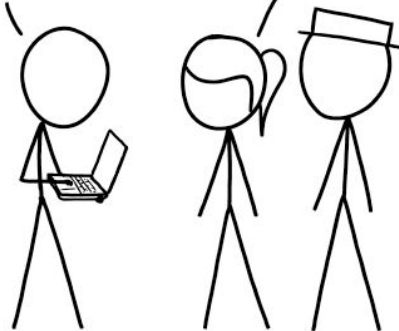


IS IT A GIANT HOUSE OF CARDS BUILT FROM RANDOM SCRIPTS THAT WILL ALL COMPLETELY COLLAPSE THE MOMENT ANY INPUT DOES ANYTHING WEIRD?



IT... MIGHT NOT BE.

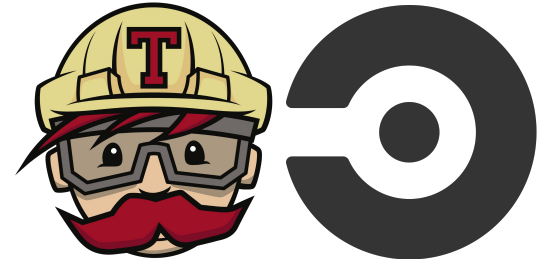
I GUESS THAT'S SOMETHING WHOOPS, JUST COLLAPSED. HANG ON, I CAN PATCH IT.



Hosted CI Platforms

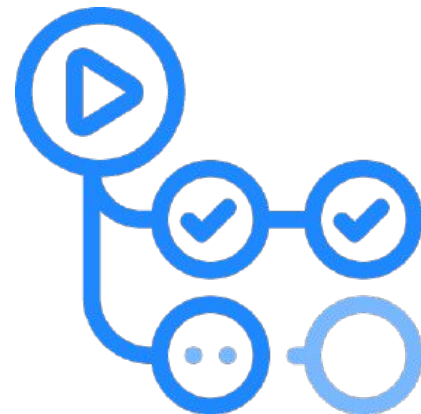
- Open source, self-hosted CI: Jenkins (2011)
 - Mostly intended for package releases rather than web services
 - Define build and test pipelines through UI or through Groovy script language
 - Unreliable, needs many third-party plugins to work, bad UI/UX
 - **Security:** remote code execution as a feature (e.g. January 2023 no-fly list breach)

- CI as a service: Travis CI, Circle CI (2011), AWS CodeBuild/CodePipelines (2014-15)
 - More secure and reliable
 - Build and test pipelines that can integrate better with version control and cloud environment



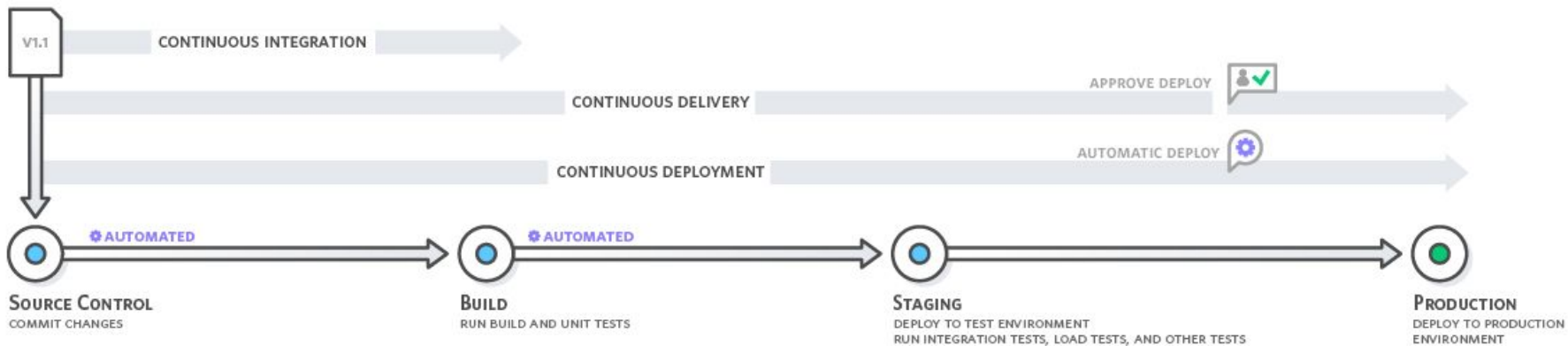
GitHub Actions

- Configure **workflows** to be run on any Git/GitHub **event**
 - e.g. pushed commit, opened pull request, manual triggers
- Workflows are made up of **jobs** (containers) with multiple **steps** (executions) run on **runners** (VMs)
 - All configured using YAML in the `.github/workflows` directory of a repository
 - Runners can be hosted by GitHub (charged by compute time) or self-hosted
- Introduced in 2018; quickly became industry standard
 - Often a starting point for any CI/CD setup, even if using external providers



Demo: GitHub Actions

Great, your code works! Let's continue...

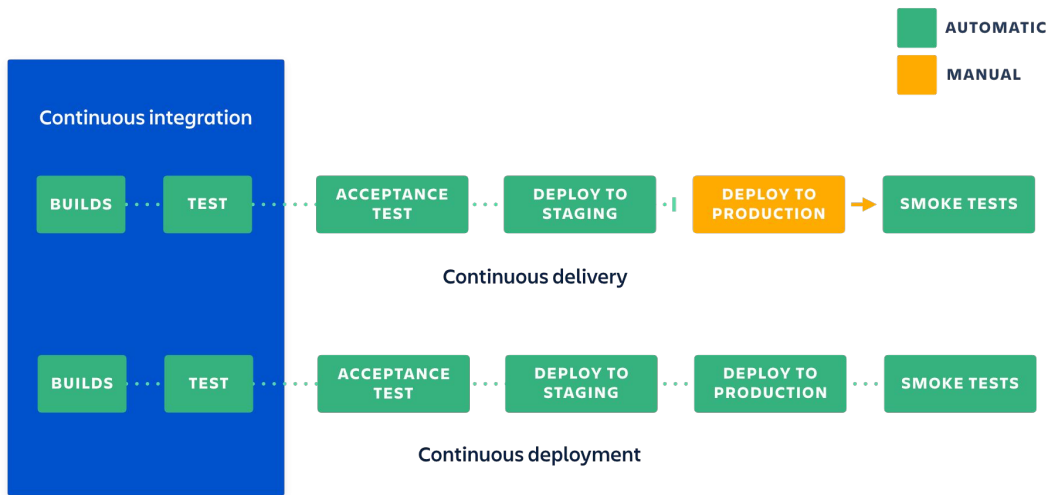


Motivation

- Traditional method of deployment: SSH into your production server/VM, run `git pull`, and restart the server
- Multiple problems with this approach:
 - **Inefficient:** every deploy is manual and takes human time
 - **Insecure:** Developers/DevOps engineers have direct SSH access to production, compromise would be bad
 - **Incompatible with serverless design:** With serverless architectures, you don't *have* a single production server

Continuous Delivery, Continuous Deployment

*Idea: Automate the deployment to a staging server.
Then automate the deployment to production.*



Configuring CD

- Key issue: once we have a built, release-ready artifact (from CI), how do we get it onto production cloud infrastructure?
- Consider two types of changes:
 - Application logic changed, but supporting infrastructure doesn't change
 - Supporting infrastructure changed

Configuring CD for Application Logic Changes

Insight: we only need to update compute resources here!

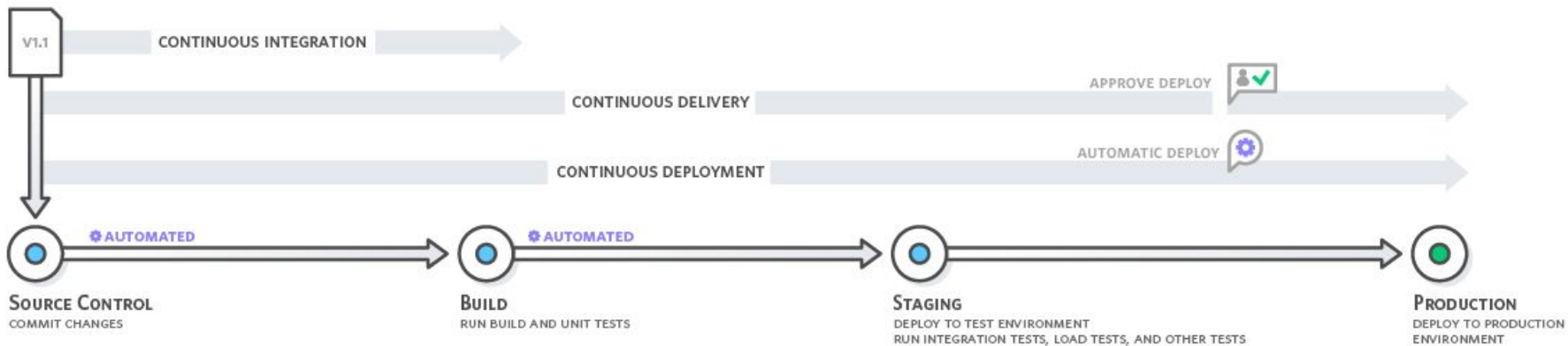
- In general: IaC best practice is to define the **containing resource** for frequently changed resources, without defining resource specifics

- e.g. if using container deployments with AWS ECS:
 - Deploy the updated container image to AWS Elastic Container Registry (ECR)
 - Force a new deployment using the updated container build
 - *Helpfully, AWS provides reusable GitHub Actions to do this.*

Configuring CD for Infrastructure Changes

- Consider whether *stateful* resources (e.g. databases, S3) need to change
 - Some IaC providers, including CDK and Terraform, support *lifecycle policies* that prevent stateful resource deletion if configured
- Separate IaC stacks by resource change frequency
 - e.g. *network stack* does not change often, *data stack* changes less frequently; *compute stack* changes often
 - Allows only redeploying more frequently changed resources, which is faster and less brittle
- Trigger the infrastructure change automatically
 - e.g. run `cdk deploy` or `terraform apply` from GitHub actions
 - At larger companies, this task often goes to infrastructure engineers for manual review

CI/CD has now been configured to get my code into production! How do I know it won't break anything?



Automatic Deployment vs Manual Review

- After code review: new code *should* be vetted such that it won't break production
 - Typical practice at medium+ companies is to require 1-2 approvals for a merge, from those who are well acquainted with that part of the codebase (“codeowners”)
- Automatically deploying any merged mainline code to production benefits:
 - Development velocity: after code review, no extra human effort is taken to deploy
 - If anything goes wrong, more eyes who could help fix it *recently* looked at the changed code
- For sensitive changes (e.g. compliance/regulation related, or infrastructure), automate change deployment *after* manual approval of staging resource

Controlling Code Paths in Production

Even if new code is deployed to production, it may not need to be immediately accessible to (all) users.

Feature flag: Gate access to new or changed features or functionality by centrally checking if that feature should be shown to a particular user.

```
if flag_provider.query("enable-analytics-cookies"):  
    response.set_cookie("analytics", generate_analytics_cookie())
```

This is complex machinery! How do we make this all secure?

Security Considerations for CI/CD

- Recall: testing and deploying requires *control plane* access to your cloud environment
 - Thus, CI/CD environment must be strongly isolated from dev environments, as well as other people's CI/CD instances (for SaaS)
 - **Challenge:** How do we securely authenticate the CI/CD control plane to the cloud environment?

- **Old way** (pre-2021): Store long-lived IAM user *access credentials* as encrypted *secrets* in CI/CD provider
 - **Problem:** if CI/CD provider is breached, then attackers have long-lived access to your cloud environment

Security

CircleCI warns customers to rotate 'any and all secrets' after hack

Carly Page @carlypage_ / 4:24 AM PST • January 5, 2023

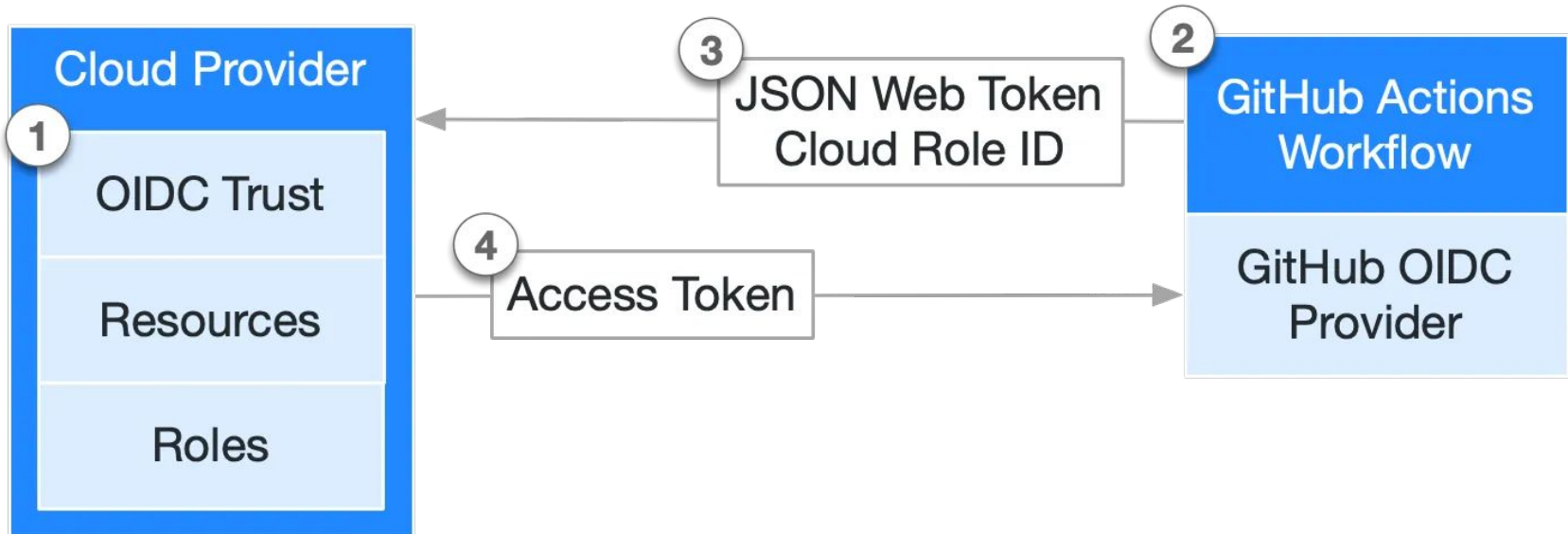
 Comment



 Image Credits: Boris Zhitkov / Getty Images

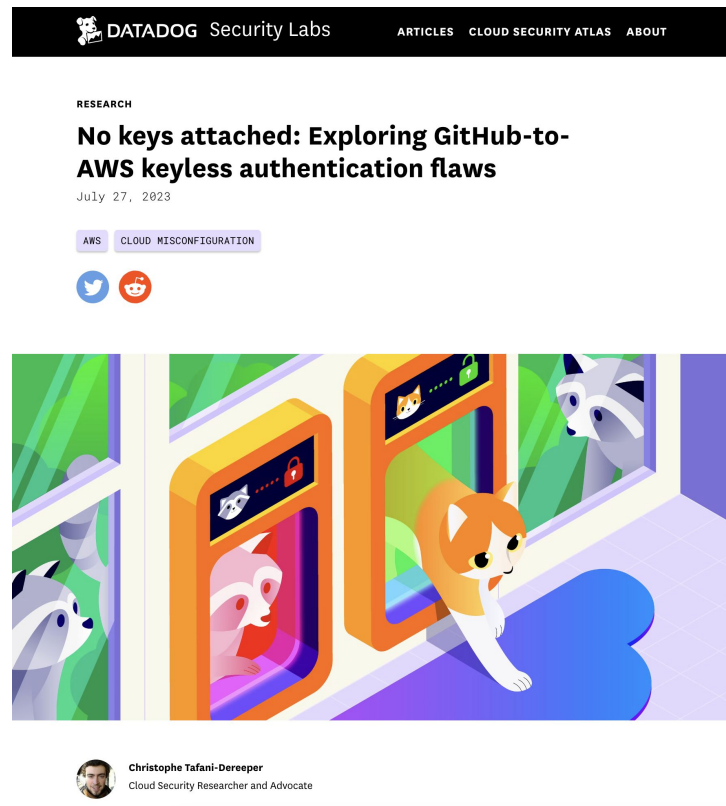
CI/CD → Cloud Environment, More Securely

- November 2021: GitHub introduces *OpenID Connect* support for Actions
 - Configure GitHub Actions to retrieve short-lived access tokens for deployment roles



Aside: GitHub Actions OIDC Footgun

- Recall: *IAM Role Trust Policy* allows specifying who can assume the role
 - For GitHub Actions, check token fields on the AWS side to make sure only the correct repository action assumes the role
- When trust policy is underspecified, *any repository* from *any owner* can assume the (privileged) deployment role
 - Allows attackers to have (short-lived, but renewable) cloud environment access
- UK government was affected in mid 2023



The image shows a screenshot of a web article from Datadog Security Labs. The header includes the Datadog logo and 'Security Labs' text, along with navigation links for 'ARTICLES', 'CLOUD SECURITY ATLAS', and 'ABOUT'. The article is categorized under 'RESEARCH' and has the title 'No keys attached: Exploring GitHub-to-AWS keyless authentication flaws', dated July 27, 2023. It is tagged with 'AWS' and 'CLOUD MISCONFIGURATION'. Below the title are social media sharing icons for Twitter and Reddit. The main content area features a colorful illustration of a cat walking through a series of orange and yellow rectangular frames, each containing a different cat's face. The background is a mix of green and purple geometric shapes. At the bottom left, there is a small circular profile picture of Christophe Tafani-Dereeper, identified as a 'Cloud Security Researcher and Advocate'.

CI/CD pipelines are attractive attack targets due to privileged cloud environment access, as well as access to the production codebase.

Anatomy of a Cloud Supply Pipeline Attack

🕒 5 min. read

Written by: **Nathaniel Quist**

Supply chain attacks targeting third-party vendors, open-source software, or other components within a supply chain have been on the rise. As organizations increasingly rely on complex supply chains and external partnerships, attackers are exploiting these connections as potential entry points. The increased adoption of open-source software and cloud-based services has also expanded the attack surface for supply chain threats. High-profile incidents, such as the SolarWinds attack in 2020, have demonstrated the potential impact and reach of supply chain attacks, compelling organizations to prioritize supply chain security and improve their defenses against such threats.

Researchers demo new CI/CD attack techniques in PyTorch supply-chain

News Analysis

Jan 12, 2024 • 11 mins

Supply Chain

Vulnerabilities

The proof of concept shows it's possible to upload malicious PyTorch releases to GitHub by exploiting insecure misconfigurations in GitHub Actions.

**Next Lecture: Fireside Chat on
Large-Scale Cloud Deployments (2/26)
GUEST LECTURE by Maria Zhang (Google)
Mandatory (graded) attendance**