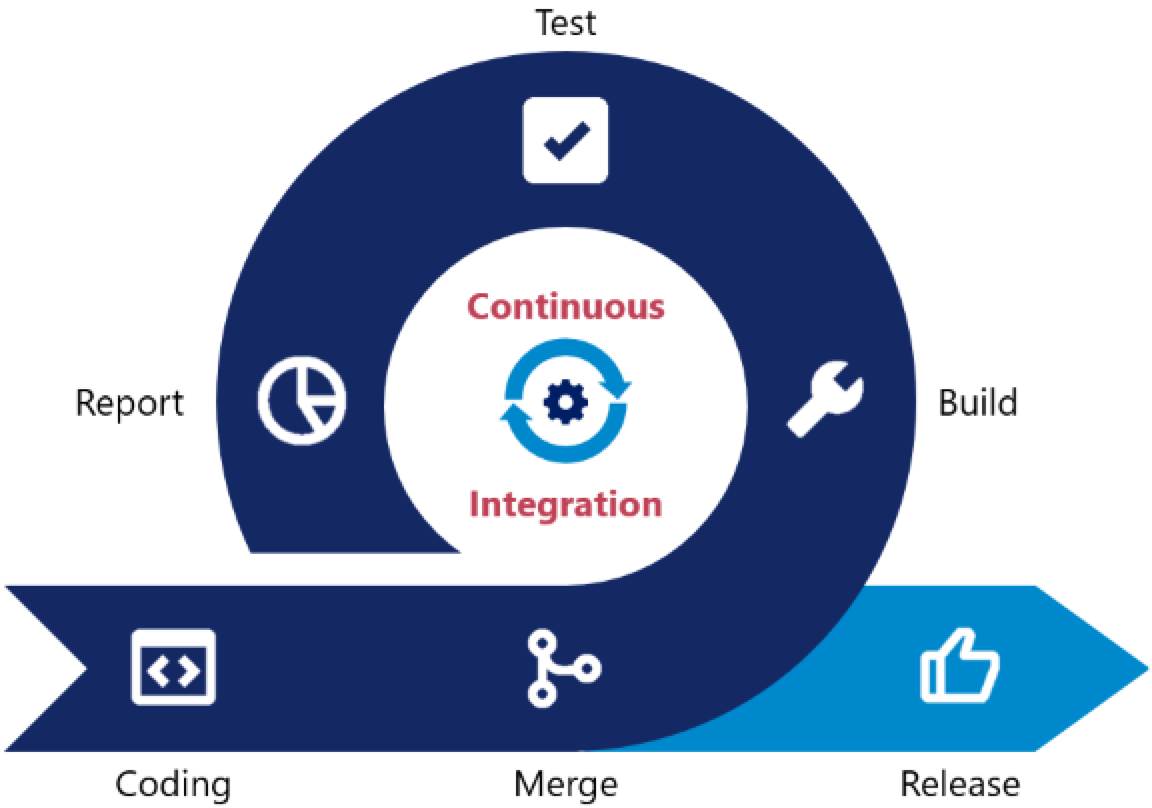
COMP1531

10.1- DevOps

Continuous Integration

Continuous integration: Practice of automating the integration of code changes from multiple contributors into a single software project.

Continuous Integration



Continuous Integration

Key principles and processes:

1. Write tests:
    - Broad tests: unit, integration, acceptance, UI tests
2. Use code coverage checkers
3. Merge and integrate code as often as possible
4. Ensure the build always works (i.e. is "green")

How it works

- Typically tests will be run by a "runner", which is an application that works with your version control software (git) to execute the tests.

Continuous integration, gitlab

Gitlab, like many source control tools, has a way of doing continuous integration. An overview is here and a start guide is here.

A simple example can be found here.

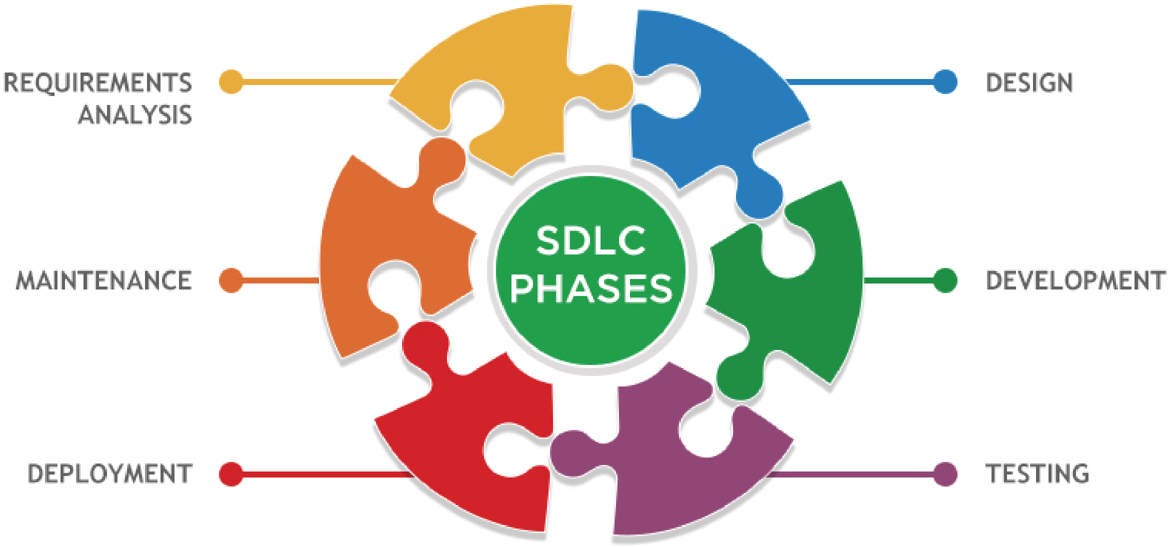
CI: Readings

More details here:

- https://about.gitlab.com/product/continuous-integration/
- https://www.atlassian.com/continuous-delivery/continuous-integration/how-to-get-to-continuous-integration

Software Deployment

Deployment: Activities relating to making a **software system available for use**.



Diagrams sourced from atlassian, gitlab, microsoft

Simple example: CSE

Every CSE student has a **public_html** folder that is exposed to the internet.

Historical Deployment

Historically, **deployment** was a much less frequently occurring process.

Code would be worked on for days at a time without being tested, and deployed sometimes years at a time. This is largely due to software historically being a physical asset

Something changed

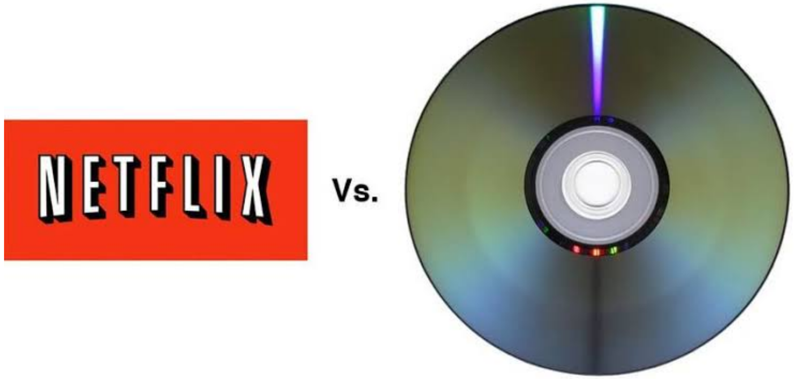
Two major changes have occurred over the last 10 years:

- Increased prevalence of web-based apps (no installs)
- Improvement to internet connectivity, speed, bandwidth

These changes (and more) have allowed for the pushing of updated software to **users** to be substantially more possible. Subsequently, users have come to expect more rapid updates.

A movement from software as an asset, to software as a service, has catalysed this transition

Software as a service (Sass)



Service vs Asset

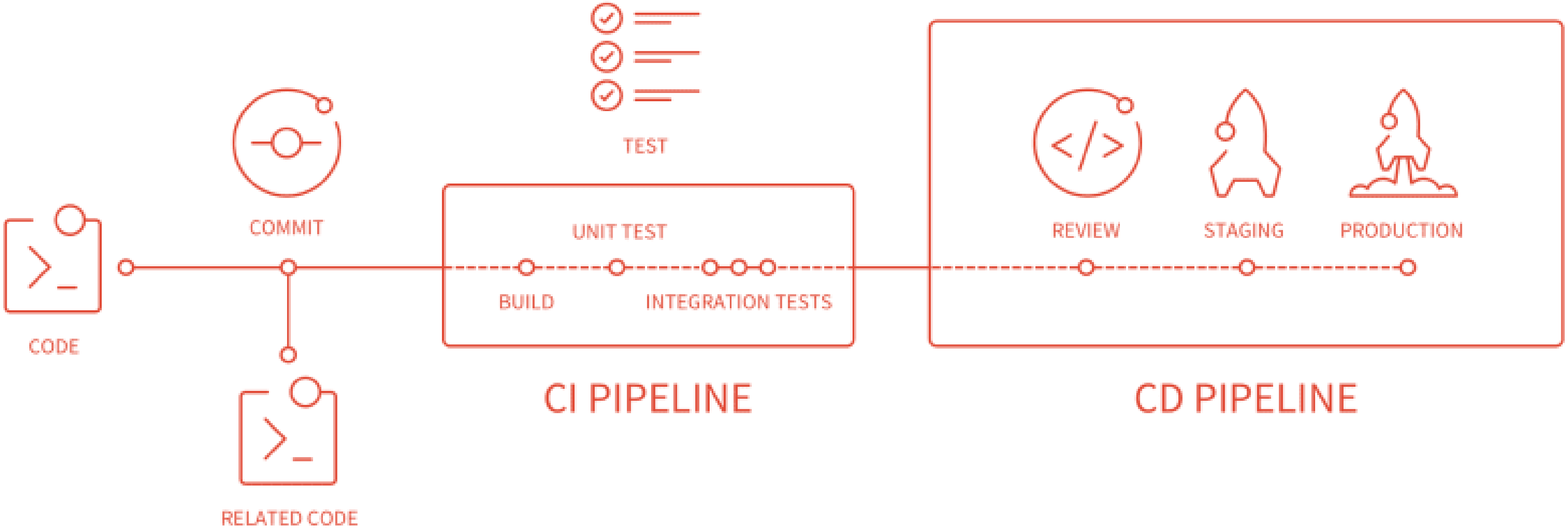
A simple case study can be found in Microsoft's movement of Windows from shipping a product, to shipping a service.

Cloud services

- Numerous cloud services offer the ability to "easily" deploy your web applications
  - Amazon Web Services
  - Google App Engine
  - Heroku

Modern Deployment

To achieve rapid deployment cycles, modern deployment isn't as simple as pushing code. Rather, a heavily **integrated** and **automated** approach is preferred.



Continuous Delivery

Continuous delivery: Allows accepted code changes to be deployed to customers quickly and sustainably. This involves the **automation of the release process such that releases can be done in a "button push"**.

Different deployments

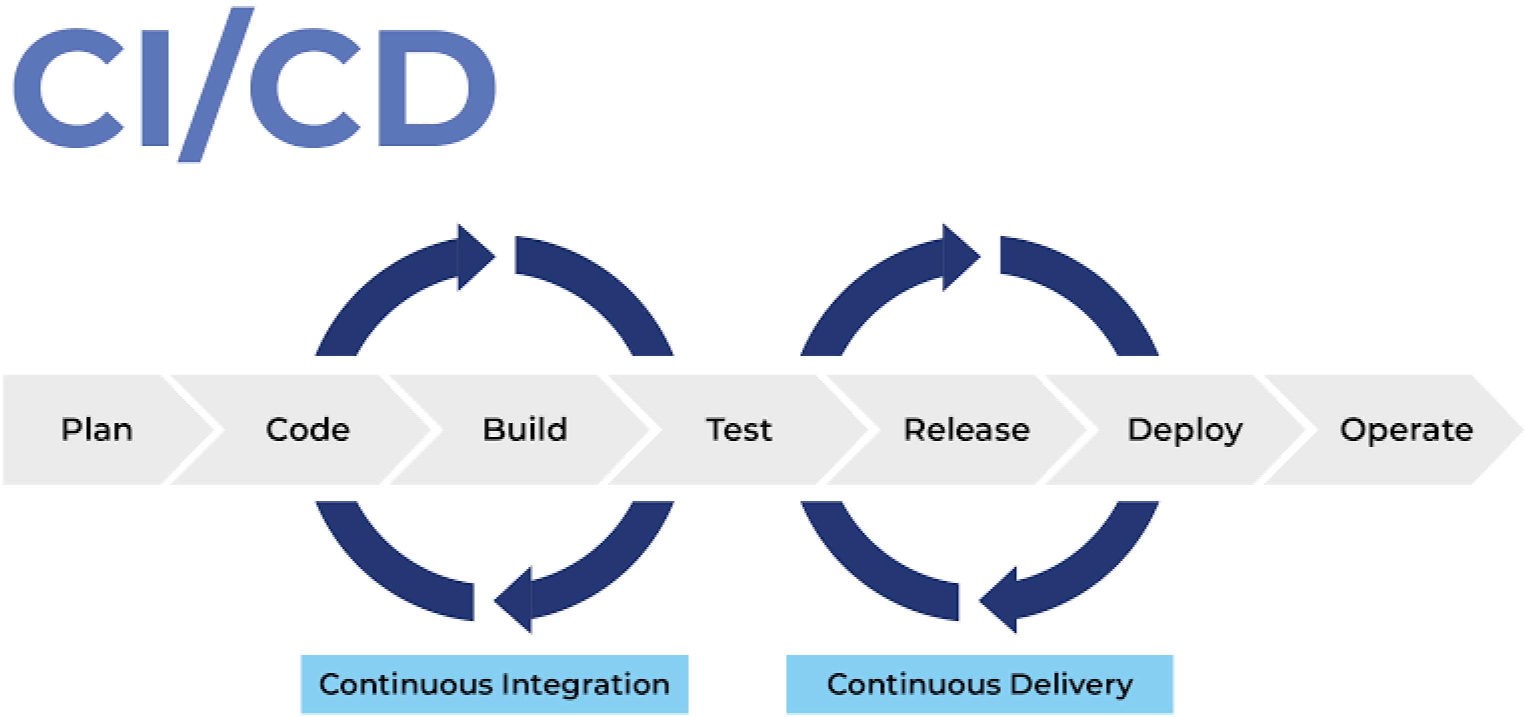
It is common to have 3 core tiers:

- **dev**:
  - released often, available to developers to see their changes in deployment
- **test**:
  - As close to release as possible, ideally identical to prod
- **prod**:
  - Released to customers, ideally as quickly as possible

Continuous Delivery

- Many companies will have a daily or weekly "ship"
- Often there is some "sign off" process before things are finally shipped
- Since the process is highly controlled, less likely to make mistakes during testing

CI/CD relationship



CD: Readings

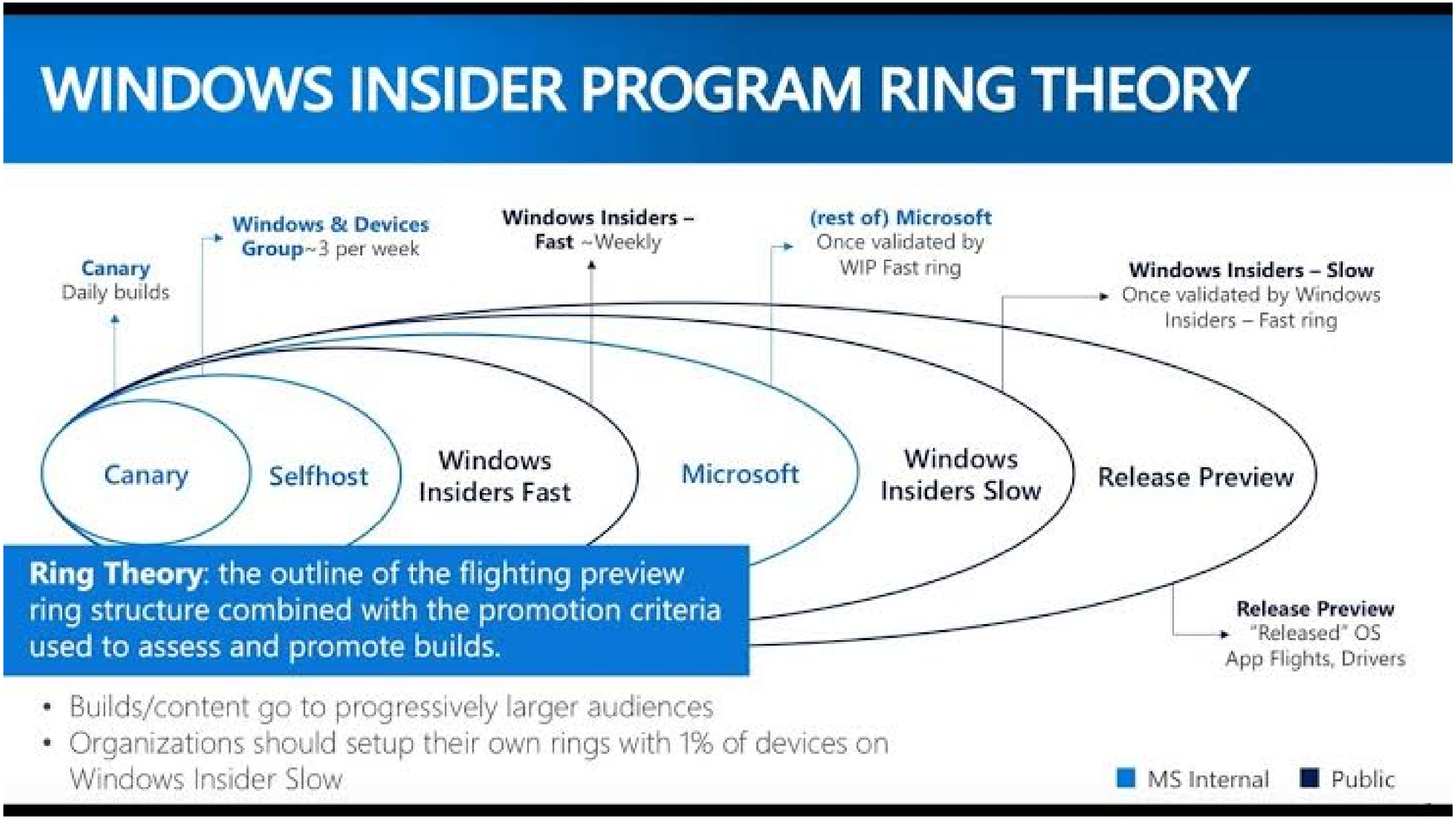
- https://www.atlassian.com/continuous-delivery/principles
- https://about.gitlab.com/product/continuous-integration/

Flighting

Continuous delivery is concerned with automatically pushing code out to dev, test, prod.

Flighting is a term used predominately in larger software projects to describe moving builds out to particular slices of users, beyond the simplicity of "dev", "test", "prod"

Flighting



WINDOWS INSIDER PROGRAM RING THEORY

Canary
Daily builds

Windows & Devices Group~3 per week

Windows Insiders – Fast ~Weekly

(rest of) Microsoft
Once validated by
WIP Fast ring

Windows Insiders – Slow
Once validated by Windows
Insiders – Fast ring

Release Preview
"Released" OS
App Flights, Drivers

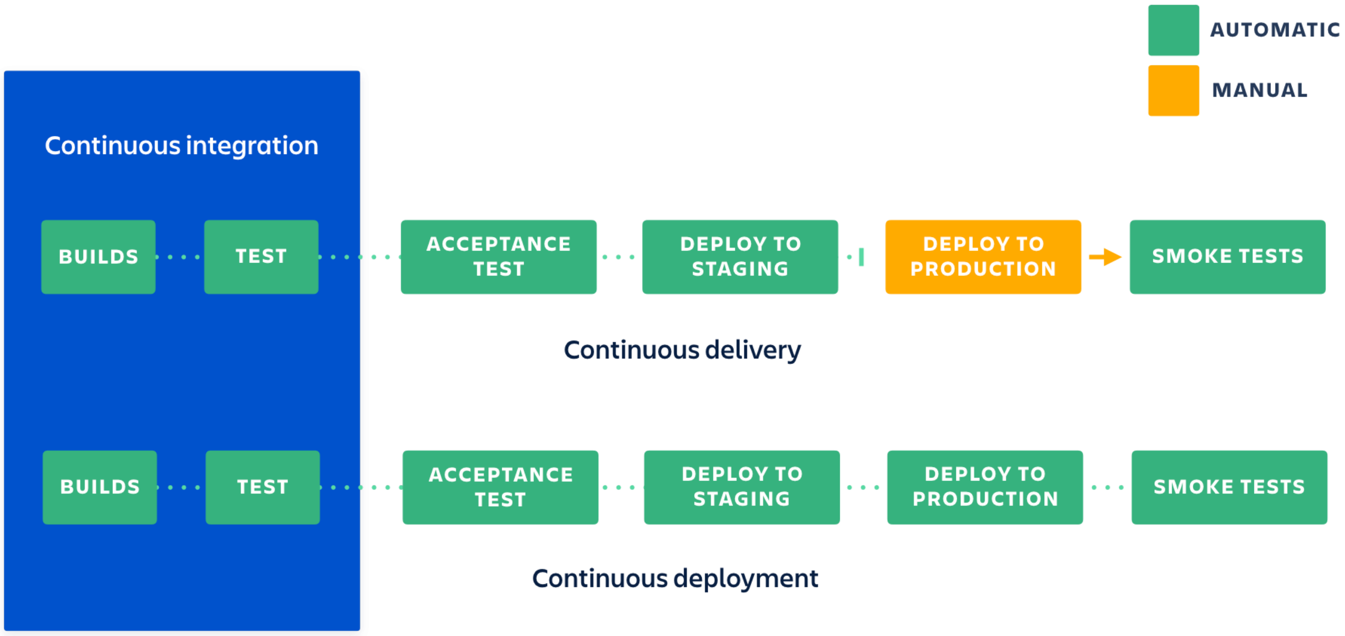
Ring Theory: the outline of the flighting preview ring structure combined with the promotion criteria used to assess and promote builds.

- Builds/content go to progressively larger audiences
- Organizations should setup their own rings with 1% of devices on Windows Insider Slow

■ MS Internal ■ Public

Continuous Deployment

Continuous Deployment is an extension of Continuous Delivery whereby changes attempt to flight toward production automatically, and the only thing stopping them is a failed test



🟩 AUTOMATIC

🟧 MANUAL

Continuous integration

BUILDS → TEST → ACCEPTANCE TEST → DEPLOY TO STAGING → DEPLOY TO PRODUCTION → SMOKE TESTS

Continuous delivery

BUILDS → TEST → ACCEPTANCE TEST → DEPLOY TO STAGING → DEPLOY TO PRODUCTION → SMOKE TESTS

Continuous deployment

A/B Testing

Is a randomised scientific experiment with multiple variants (typically two). It consists of one independent variable, with all other variables controlled.

Consists of having two "versions" randomly distributed to end-users, and then monitoring the results. These versions can either be:

- Managed within the same instance
- Sent to different instances via a load balancer

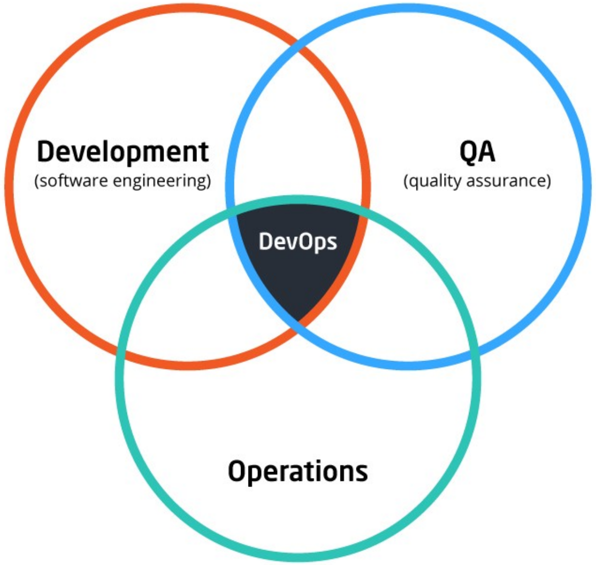
- Further details
- Examples of AB testing

CD: Further Reading

- https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment

DevOps

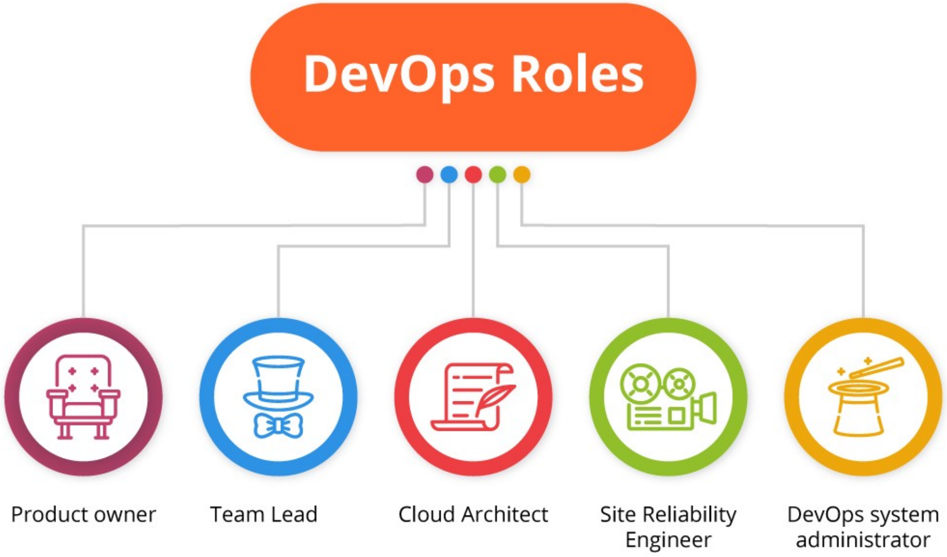
A decade ago, the notion of dev ops was quite simple. It was a role dedicated to gluing in the 3 key pillars of deploying quality assured software



DevOps is a set of practices intended to reduce the time between committing a change to a system and the change being placed into normal production, while ensuring high quality [Wikipedia. Yes, Wikipedia]

DevOps

As development teams become less silo'ed, modern DevOps is less a role, and more a series of roles or aspect of a role.



Source & Reading: https://hackernoon.com/devops-team-roles-and-responsibilities-6571cfb56843

Maintenance & Monitoring

Maintenance: After deployment, the use of analytics and monitoring tools to ensure that as the platform is used and remains in a healthy state.

Monitoring often has two purposes:

- *Preserving user experience*: Monitoring errors, warnings, and other issues that affect performance or uptime.
- *Enhancing user experience*: Using analytical tools to monitor users or understanding their interactions. Often leads to customer interviews and user stories

Maintenance

Maintenance: After deployment, the use of analytics and monitoring tools to ensure that as the platform is used and remains in a healthy state.

Health is defined by developers, but often consists of:

- Monitoring 4XX and 5XX errors
- Ensuring disk, memory, cpu, and network is not overloaded

Often these aren't actively monitored, but rather monitored with alerts and triggers