AARHUS UNIVERSITET

Microservices and DevOps

DevOps and Container Technology

Continuous Integration

Henrik Bærbak Christensen

Fowler: Practices

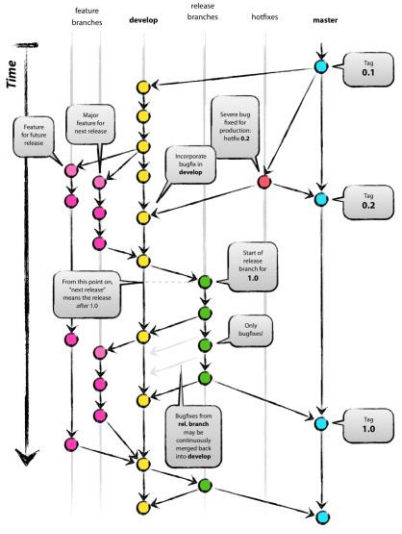
- CI requires a lot of practices to be in place…
- [Fowler, 2006]
- Premise:
  - Monolith focus…

Practices of Continuous Integration

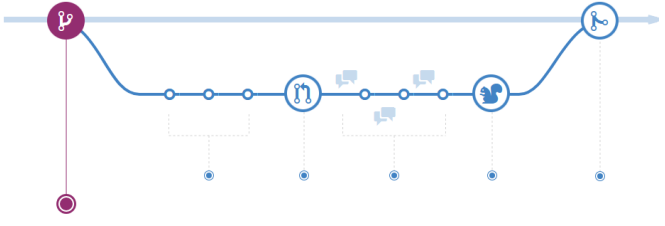
Maintain a Single Source Repository.
Automate the Build
Make Your Build Self-Testing
Everyone Commits To the Mainline Every Day
Every Commit Should Build the Mainline on an Integration Machine
Fix Broken Builds Immediately
Keep the Build Fast
Test in a Clone of the Production Environment
Make it Easy for Anyone to Get the Latest Executable
Everyone can see what's happening
Automate Deployment

Maintain Single Repository

- **Use** a strong software configuration management tool
  - Git, Subversion, …
- Define a *branching and release management strategy*



  - Ex:
    - http://nvie.com/posts/a-successful-git-branching-model/
    - https://guides.github.com/introduction/flow/



AARHUS UNIVERSITET

Automate Build

- **Use** a build tool
  - Gradle, Maven, Ant, …
- Litmus test on New Machine
  - *Check out source, issue a **single** command, and it should be running!*
  - Does your SkyCave pass the litmus test?

AARHUS UNIVERSITET

Self-Testing Build

- **Make** the build do
  - Compile, **run (all) test suites**, run
- Built into the Test-Driven Development process…
- Note
  - Run all **unit test suites** on every build on dev machine
    - But not the **integration test suites** that exercise the out-of-process tests. They are too slow
  - Use a CI approach to **run all test suites** that include the integration test suites

AARHUS UNIVERSITET

Commit to Master Every Day

- ‘High frequency’ is the ideal !
- **Commit** to the integration/master branch frequently

- Integration is communication and early feedback
  - Finding the integration defect today is better than tomorrow
- In line with TDD principle
  - *Take small steps!*

- *GitHub Flow* is a simple branch model supporting this

AARHUS UNIVERSITET

Every Commit is Integration

- Every commit **must** trigger integration build on build server
  - But – commit to which branch?

- Manually
  - Go to integration server; checkout out, and build (=test)

- Continuous Integration Server
  - Dedicated software and machine
  - Bitbucket/Gitlab pipelines…

AARHUS UNIVERSITET

Fix Broken Builds Immediately

- You **must** fix any integration defects immediately as they appear
  - Kent Beck: ”Nobody has a higher priority task than fixing the build”
- In CI, you strive to have a deliverable product at all times!

- One resort: *Revert to last known good build…*

AARHUS UNIVERSITET

Keep the Build Fast

- **Make** the build fast, so defects are detected fast
  - What is fast? 10 minutes? One hour?
  - Warstory: The 15 minute full compile in the 1990'ies
    - Meant: you forgot why you started it!
- Use pipelines and diverse test sets
  - Smoke testing — Find obvious/big defects
  - Unit testing — Much is stubbed (like DBs!)
  - Comprehensive testing — Real DBs are started, filled, and tested
- Two staged pipeline
  - First-stage: Just unit tests ("very fast=10min" ?);
  - Second-stage: Comprehensive (hours);

AARHUS UNIVERSITET

Test in Clone of Production Env

- **Make** the test environment look like the production environment as much as possible
  - Same DBs, same version, same configuration, etc.

- Not always possible
  - Consider NetFlix or Twitter ☺

AARHUS UNIVERSITET

Easy to Get Latest Executable

- Every stakeholder **must** have access to latest product
  - Users, sales, developers, managers, …

- Get feedback from the ones that will use it!

AARHUS UNIVERSITET

Everyone Can See What's Happening

- **Make** every change visible; everybody can see the current state
  - Alas: communication between stakeholders!
- DashBoards on the integration server shows 'what is happening'

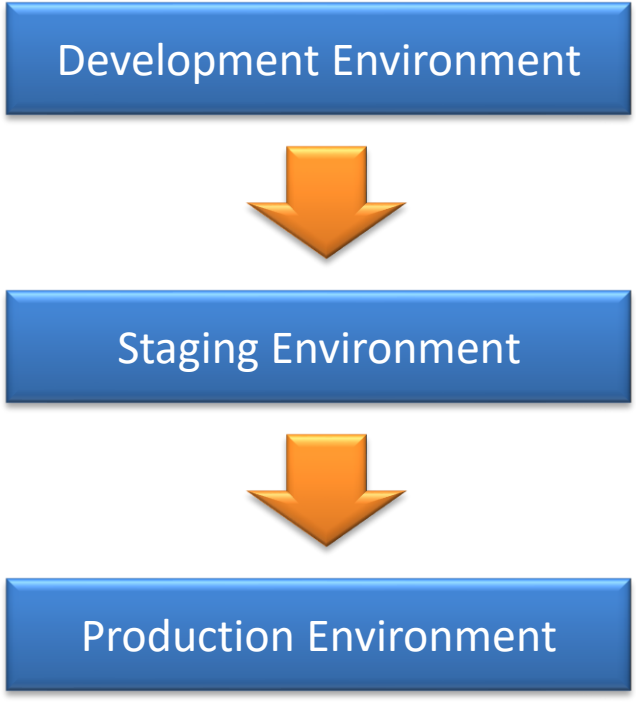
- Fowler warstory
  - Calendar with green/red square for ok/broken builds per day

  - Induce a common goal of 'getting all the days green'

AARHUS UNIVERSITET

Automate Deployment

- **Make** the deployment automatic so you can easily and quickly move executables between environments
  - Make scripts that does this movement fast, easy, and agile…
  - Today: Use IaC and tooling…
- Again: Speed means we *will* do it!
- Major driver for Docker…



AARHUS UNIVERSITET

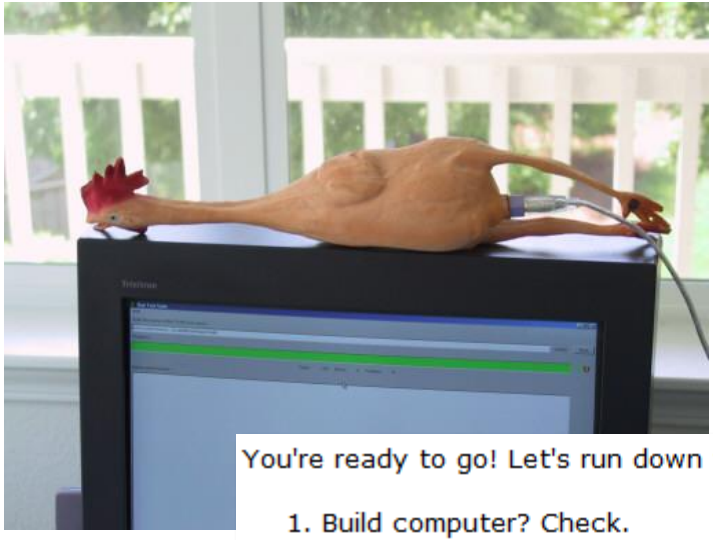
Discussion

- CI means
  - Reduced risk
  - Increases awareness, visibility, of trouble spots
    - Increases confidence
  - Combats ‘Broken Windows Syndrome’

- Liabilities
  - Tools suite in place
  - Training and habits must be in place

Discussion

- CI does *not require a build-server*. It is a *practice!*



You're ready to go! Let's run down the pre-launch checklist:

1. Build computer? Check.
2. Ridiculous toy? Check.
3. Annoying bell? Optional.
4. Automated build? Check.
5. Group agreement? Check.

AARHUS UNIVERSITET

The Three Question

According to Jez Humble's
"Continuous Delivery"

AARHUS UNIVERSITET

Are we really doing it?

Prerequisites for Continuous Integration

Continuous integration won't fix your build process on its own. In fact, it can be very painful if you start doing it midproject. For CI to be effective, the following practices will need to be in place before you start.

Check In Regularly

The most important practice for *continuous integration* to work properly is frequent check-ins to trunk or mainline. You should be checking in your code at least a couple of times a day.

Create a Comprehensive Automated Test Suite

If you don't have a comprehensive suite of automated tests, a passing build only means that the application could be compiled and assembled. While for some teams this is a big step, it's essential to have some level of automated testing to provide

Don't Check In on a Broken Build

The cardinal sin of continuous integration is checking in on a broken build. If the build breaks, the developers responsible are waiting to fix it. They identify the cause of the breakage as soon as possible and fix it. If we adopt this strategy, we will always be in the best position to work out what caused the breakage and fix it immediately. If one of our colleagues has made a check-in and broken the build as a result, then to have the best chance of fixing it, they will need a clear run at the problem. They don't want us checking in further changes, triggering new builds, and compounding the failure with more problems.

AARHUS UNIVERSITET

MicroService Take…

Single Source – Single Build

- All services in **one** Repo, all build in one process
  - Benefits/liabilities?

Questions
- Ownership?
- Speed?
- Broken builds?
- What needs deploying?

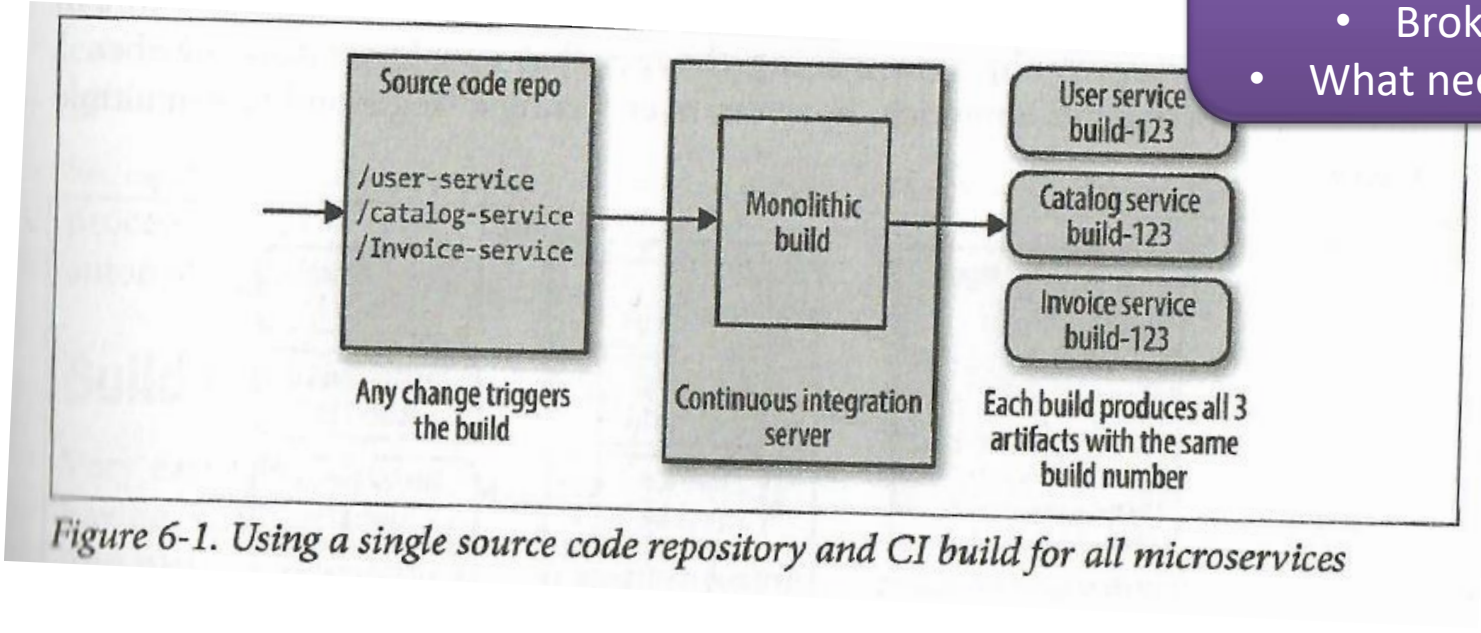


Figure 6-1. Using a single source code repository and CI build for all microservices

Single Source – Multi Build

- All services in one repo, build per service
  - Benefits/liabilities?

Questions
- Ownership?
- Speed?
- Broken builds?
- What needs deploying?

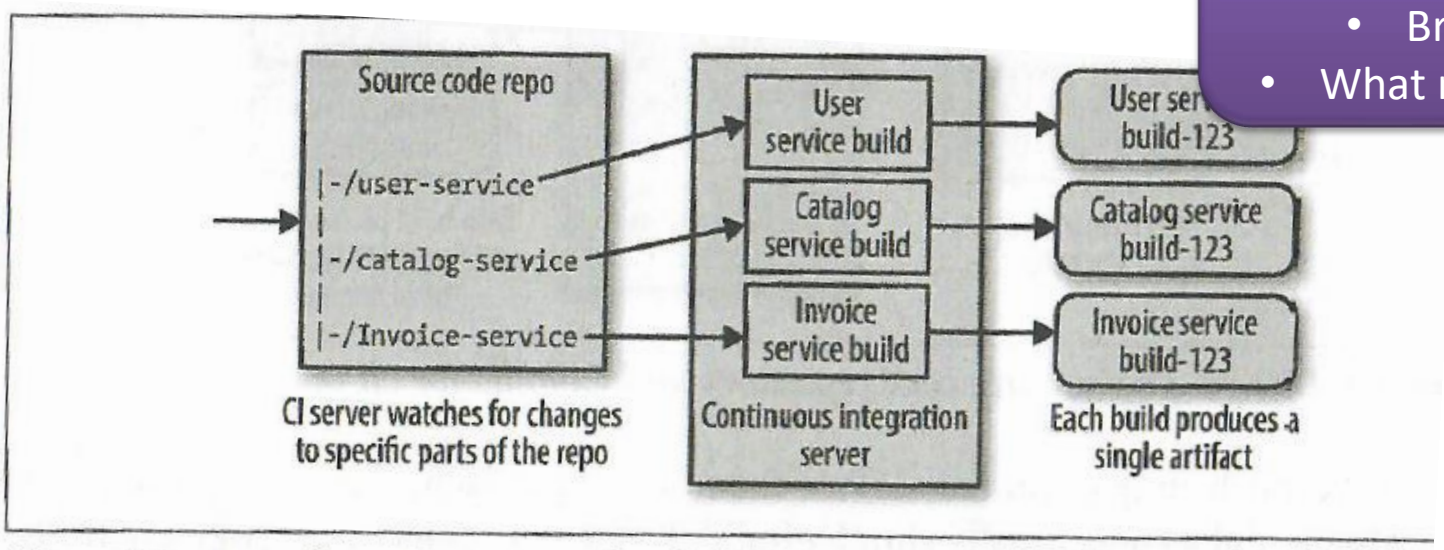


Figure 6-2. A single source repo with subdirectories mapped to independent builds

Multi Source – Multi Build

- One service per repo, one build per service
  - Benefits/liabilities?

Questions
- Ownership?
- Speed?
- Broken builds?
- What needs deploying?

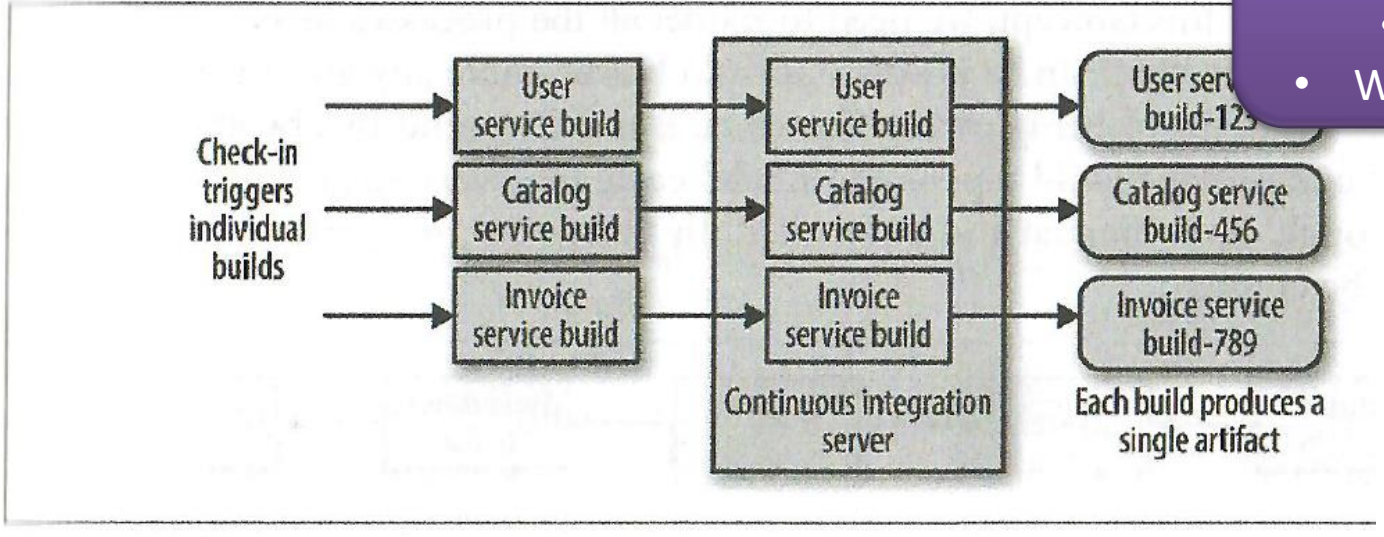
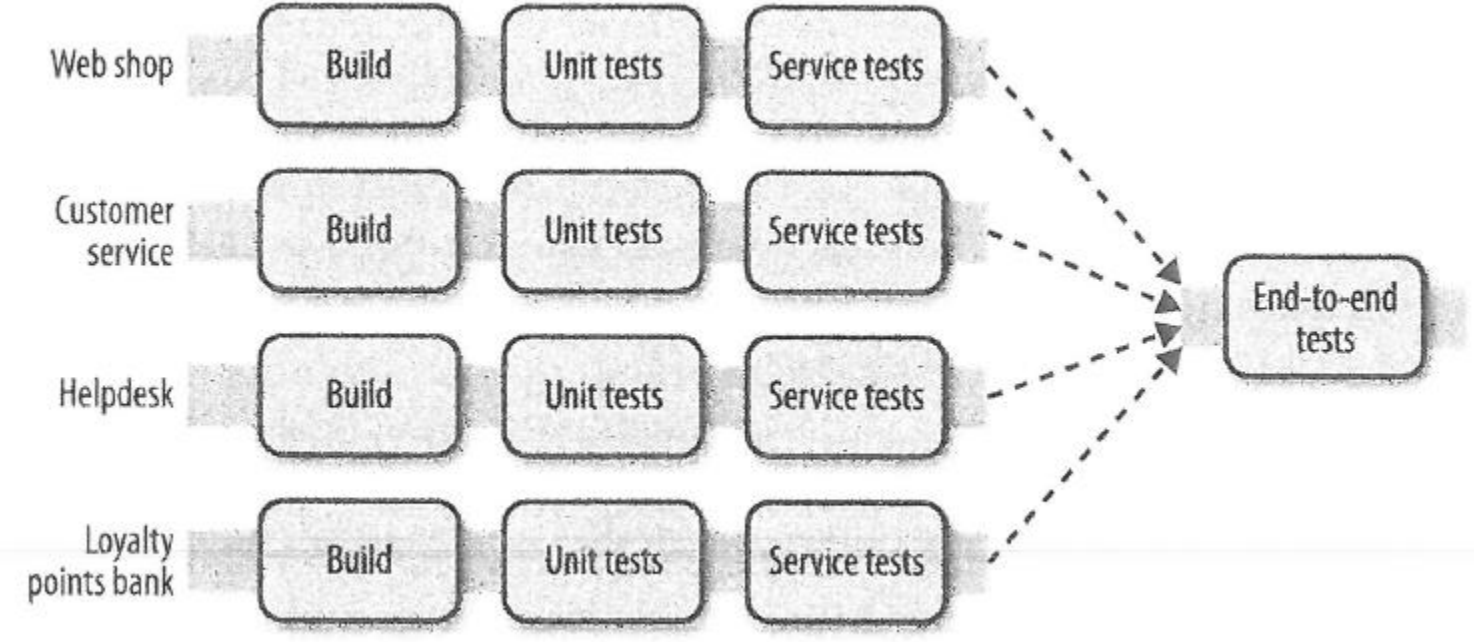


Figure 6-3. Using one source code repository and CI build per microservice

Test Journeys

- Triggering the E2E tests / journeys is also a bit more involved…



AARHUS UNIVERSITET

Discussion

- For completeness
  - Which model is missing?
  - Does this model have some virtue?

- SkyCave is monolith, but…
  - Which model does is sort of lend itself to?

Summary

Practices of Continuous Integration

- Maintain a Single Source Repository.
- Automate the Build
- Make Your Build Self-Testing
- Everyone Commits To the Mainline Every Day
- Every Commit Should Build the Mainline on an Integration Machine
- Fix Broken Builds Immediately
- Keep the Build Fast
- Test in a Clone of the Production Environment
- Make it Easy for Anyone to Get the Latest Executable
- Everyone can see what's happening
- Automate Deployment

Summary

- Microservices add some complexity
  - Single source – single build
  - Single source – multi build
  - Multi source – multi build
  - Multi source – single build (?)