Microservices

DevOps

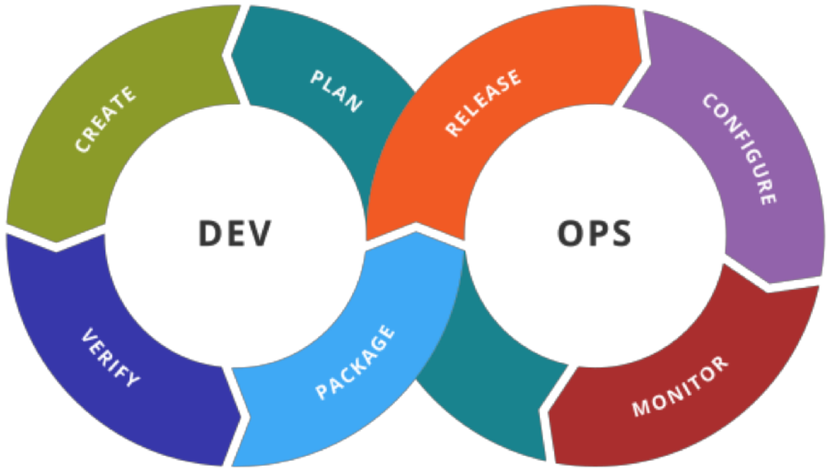
Content

1. DevOps Introduction
2. Pipeline (continuous integration / continuous delivery / continuous deployment)
3. Popular CI Tools
4. Infrastructure as Code

DevOps Introduction

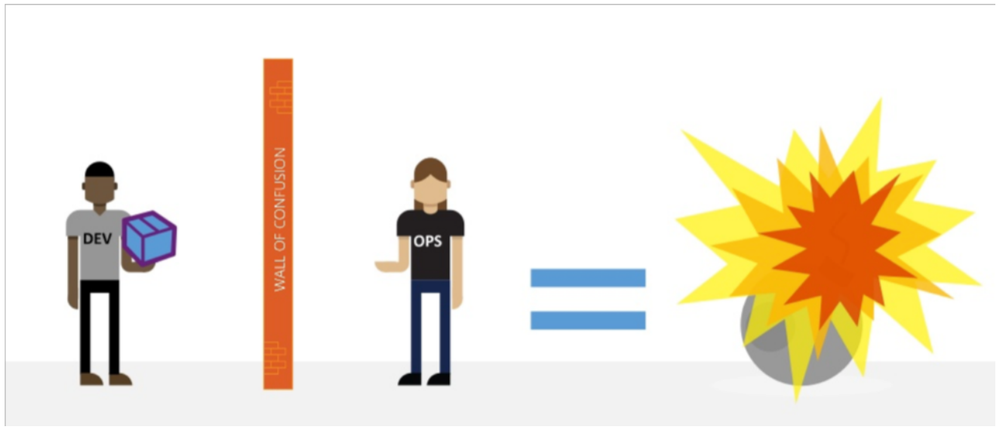
- DevOps is a software engineering practice that aims at unifying software development (Dev) and software operation (Ops)
  - People, processes and tools
  - working together
  - to enable continuous delivery of value
  - to the end users
  - fast(er)

Cultural change required
We just focus on the software delivery process process!



DevOps - Challenges

- development vs. operation
- agile vs. stability
- a lot iterations
- a lot of releases
- monitoring of all services
- code quality

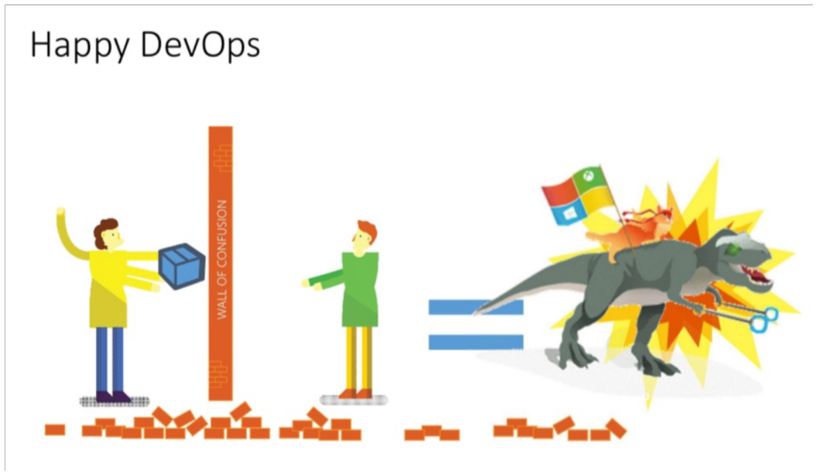


Goals

The goals of DevOps span the entire delivery pipeline.
- Improved deployment frequency
- Faster time to market
- Lower failure rate of new releases
- Shortened lead time between fixes
- Faster mean time to recovery
- Easier onboarding for new developers

Dev: Create Change, Add or modify features

Ops: Create stability, Create or enhance services



Plan

- general project management tasks
- backlog
- documentation
- scrum planning
- Retrospectives
- use mvp's – don't develop the hole software without test! (from lean startup)
- event storming, story mapping

Code

— coding with git (hopefully not with svn anymore)
— code reviews

build

- Continuous Integration
- different build tools for your project
- package manager

test

- Code coverage report
- automatic test
- acceptance testing
- Integration testing

release

- Packaging like building a jar or a docker container
- Pre-Deployment-Staging
- Release automation

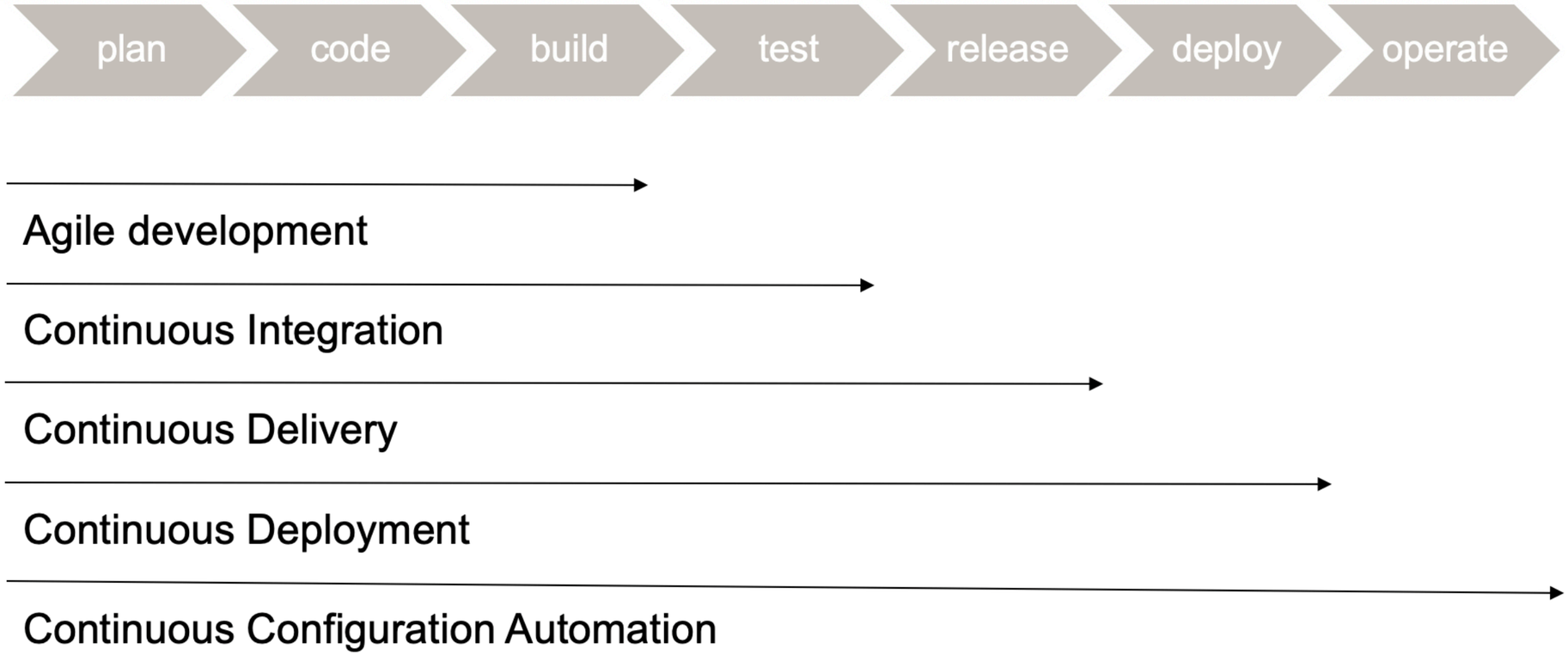
deploy

- configuration
- production staging
- Infrastructure as a Code (IaaC)

operate

- logging
- tracing
- exception handling
- performance monitoring
- support / service desk
- feature toggles
- metrics

DevOps Pipelines



Automate Almost Everything for Microservice

- The build
- database change (flyway) or Events when Event Sourced
- deployment to test/staging/production environments
- tests
- monitoring / remediation plans
- Infrastructure as code
- Service discovery, DNS, Load Balancing, Auto Scaling, …

Your continuous deployment pipeline should be a model of your process for getting software from version control into the hands of your users.

Popular Pipeline Tools

- Circle CI
- Travis CI
- GitLab CI
- Bamboo
- Codeship
- Jenkins

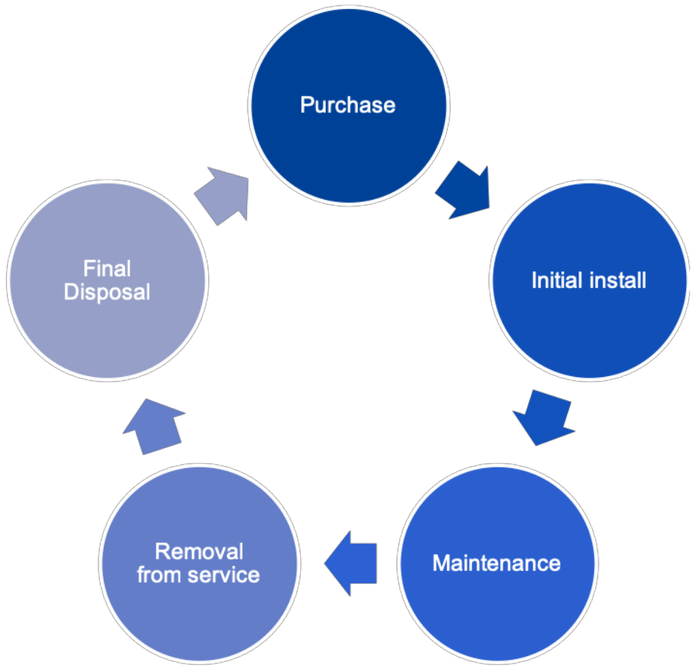
Circle CI Sample

— In Exercise 1 we will use circle ci

```
version: 2
jobs:
  build:
    docker: # use the docker executor type; machine and macos executors are also supported
      - image: circleci/node:4.8.2 # the primary container, where your job's commands are run
    steps:
      - checkout # check out the code in the project directory
      - run: echo "hello world" # run the `echo` command
```

Infrastructure as Code - History

- In the past, administrators have taken care of each server for its entire lifecycle
- Every server was kind a „piece of art“
- Every server hosted a large number of services
- To be able to restore a server, administrators created full backups of every server (e.g. the /etc directory of Linux servers)



Introduction - Intro

- Define the configuration of your whole infrastructure as code
- Whenever you don’t need a server any more delete it and restore it if needed (only data backup and code is required)
- It’s easy (and required) to put all your infrastructure code into a version control system
- Test your infrastructure code as you test your program code!
- New servers can be bootstrapped full-/ or semi-automatic
- It doesn’t matter if you’re building a Docker container or if you’re installing a virtual or physical server – infrastructure code may be applied to all of them
- Focused on managing a large number of servers (instead

IaC Fundamentals

- Server or agent collects facts about target system
- Configuration may be applied multiple times leading to the same result (means no change if not necessary)
- Process of applying configuration may be forced
- Configuration describes a desired state of a machine (configuration files, installed packages, existence of users, running services, …)
- Most of the systems abstract the concrete operating system (e.g. the concrete package manager)
- Most of the systems are resource orientated to describe (Packages, Files, Users/Groups, Services)