

Continuous {Integration, Delivery, Deployment}

Release Engineering for Machine Learning Applications
(REMLA, CS4295)



Sebastian Proksch
S.Proksch@tudelft.nl



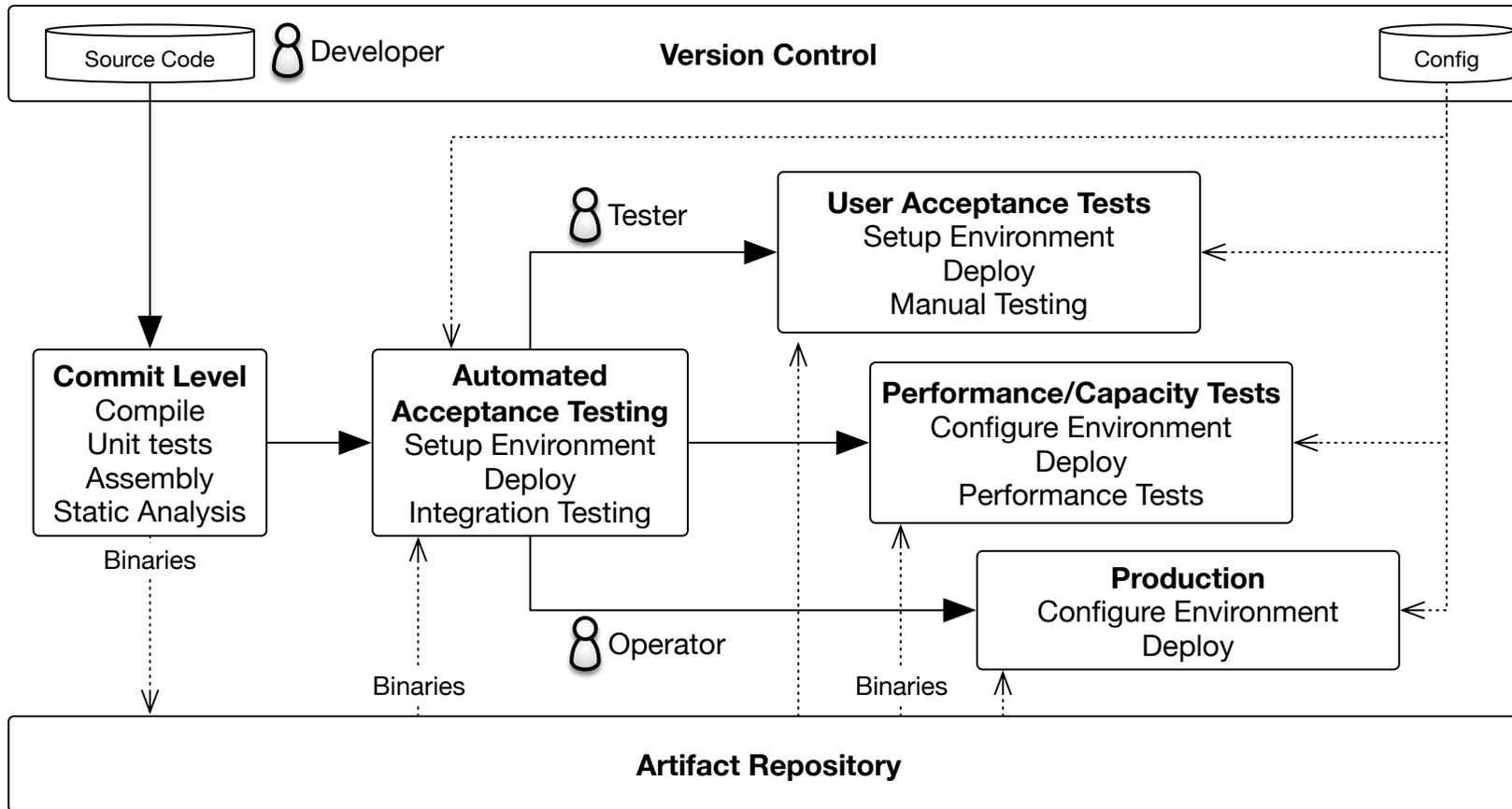
Luís Cruz
L.Cruz@tudelft.nl

Goal of today...

- Differentiate the various continuous X buzzwords
- Understand the setup and the goals of a basic delivery pipeline
- Use semantic versioning in your projects
- Understand challenges in dependency resolution
- Get a first glimpse in the capabilities of GitLab that go beyond a simple build server

Continuous <Enter Buzzword here>

A Basic Delivery Pipeline



Environment gets more “production-like” from left to right.

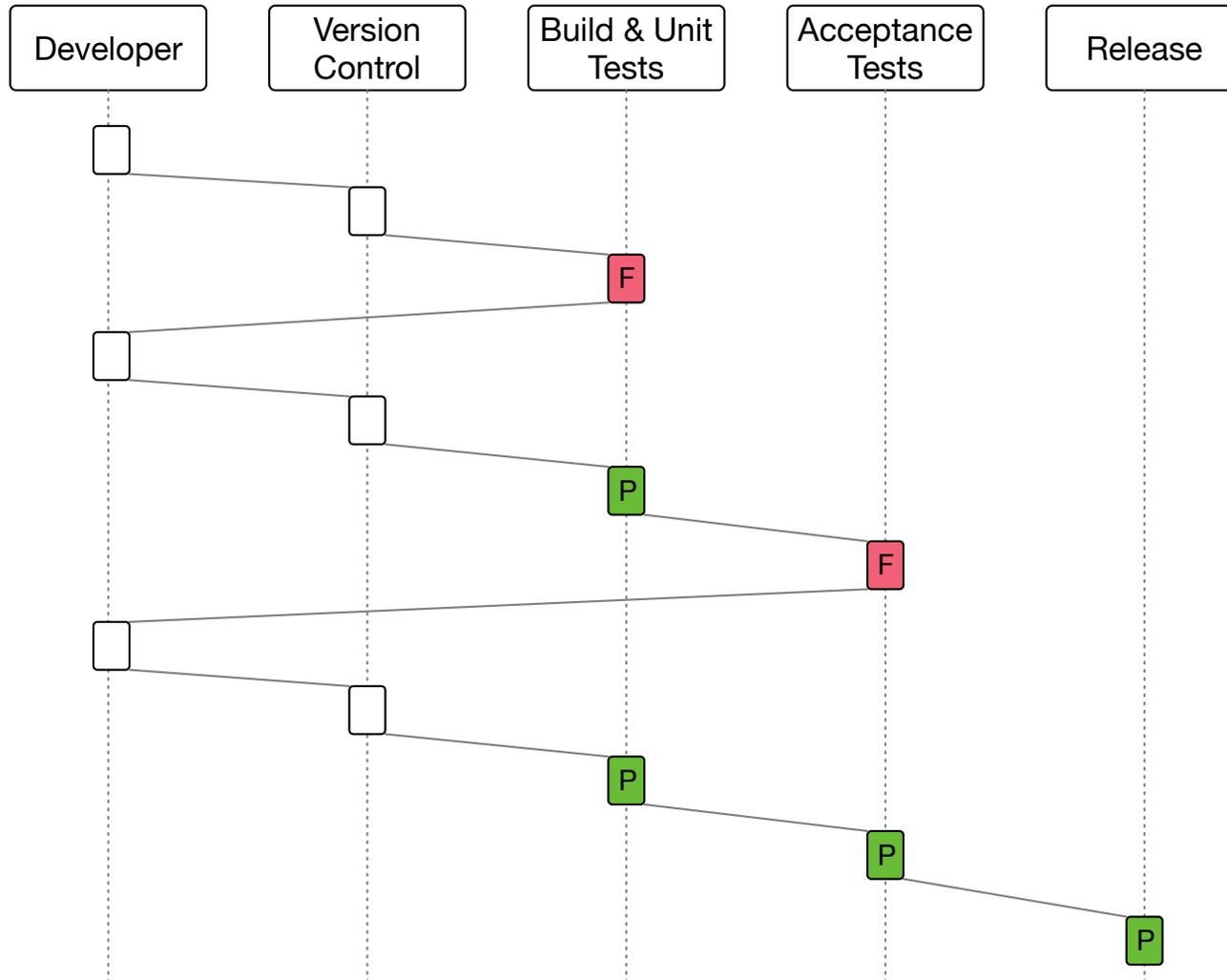
Modern Processes merge roles (“DevOps”)

Configuration Management

- Infrastructure as Code
(Environment Setup)
- Configuration as Code
(Component Interaction)

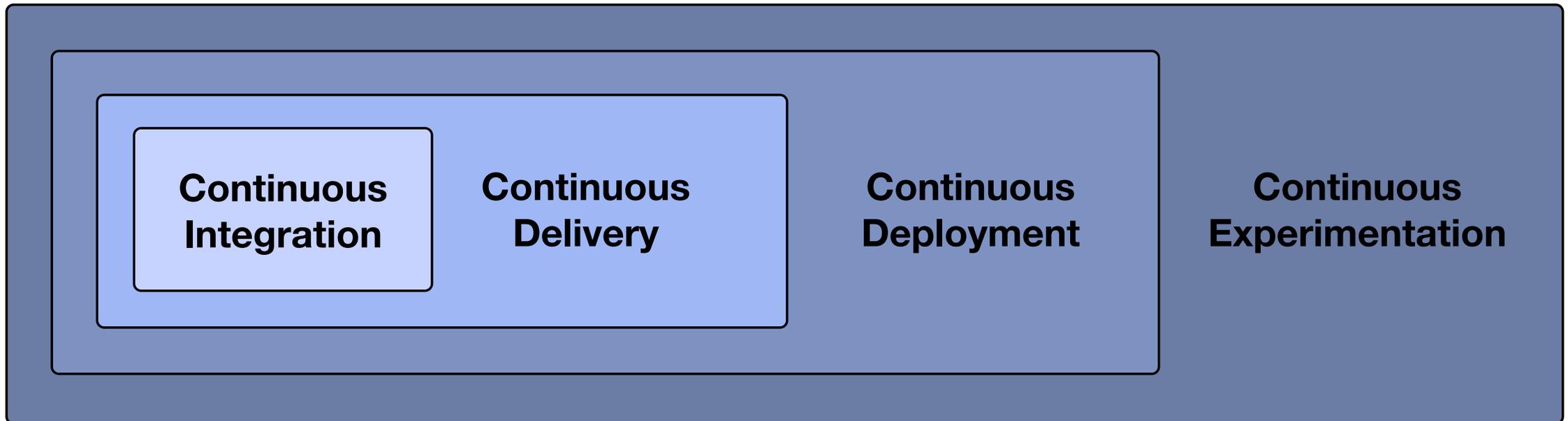
We will cover this topic, when we talk about Kubernetes.

Deployment Workflow



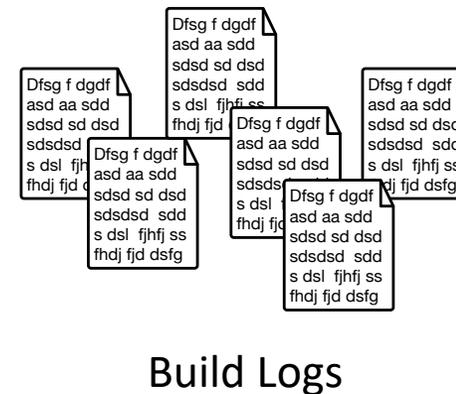
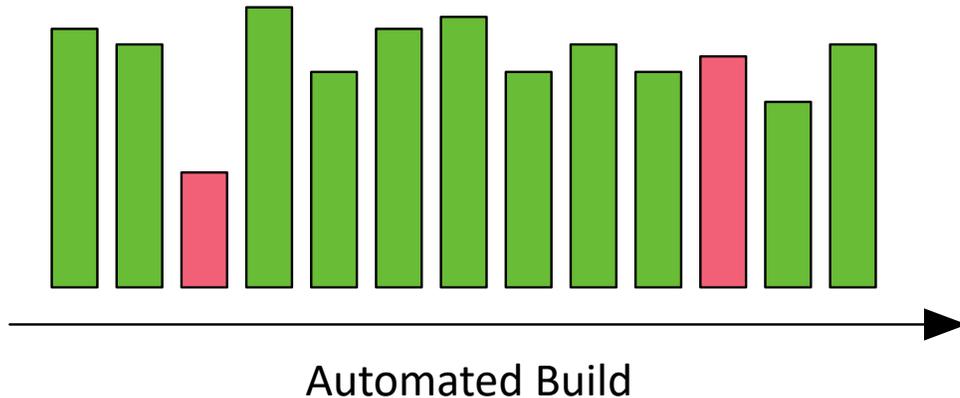
Automation is key
for timely feedback!

Continuous ...



Continuous Integration

- Integrate changes
 - Trunk-based development (Fowler's definition)
 - Pipeline-centric development (everybody else's def.)
- Quality assurance
- Maintainability



Continuous Delivery

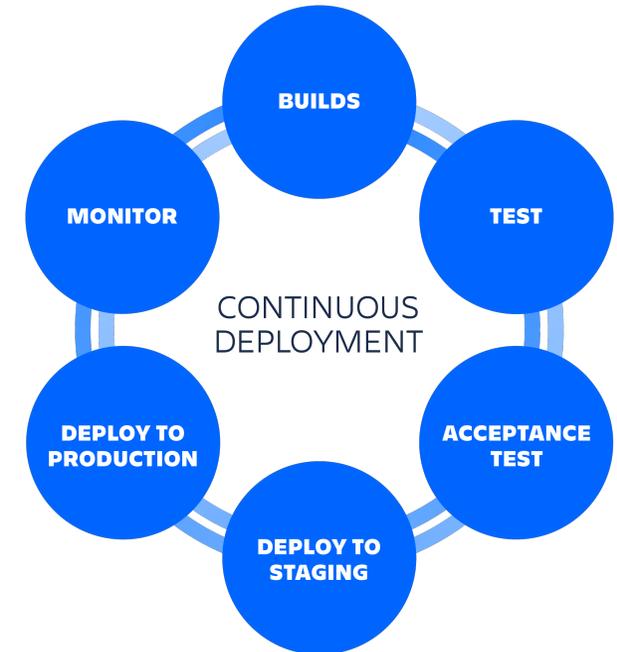
- Automate packaging
- Configuration
- Versioning
- Automated acceptance testing
- “One-click Deployments”



“make every change releasable”

Continuous Deployment

- Automated User Acceptance Tests
- Prepare/configure environment
- Infrastructure as code



“release every change”

Continuous Experimentation

- “Testing on live”
- Release to subsets of your users
- Monitor effect of changes
- Release decision based on monitoring results

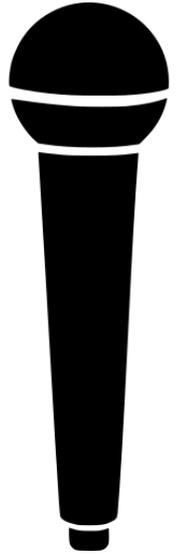


“release every *potential* change”

Application Types

What does Continuous Delivery mean for...

- Webapps?
- Machine Learning Models?
- Downloaded Application?
- Mobile App/App Stores?
- Software library?

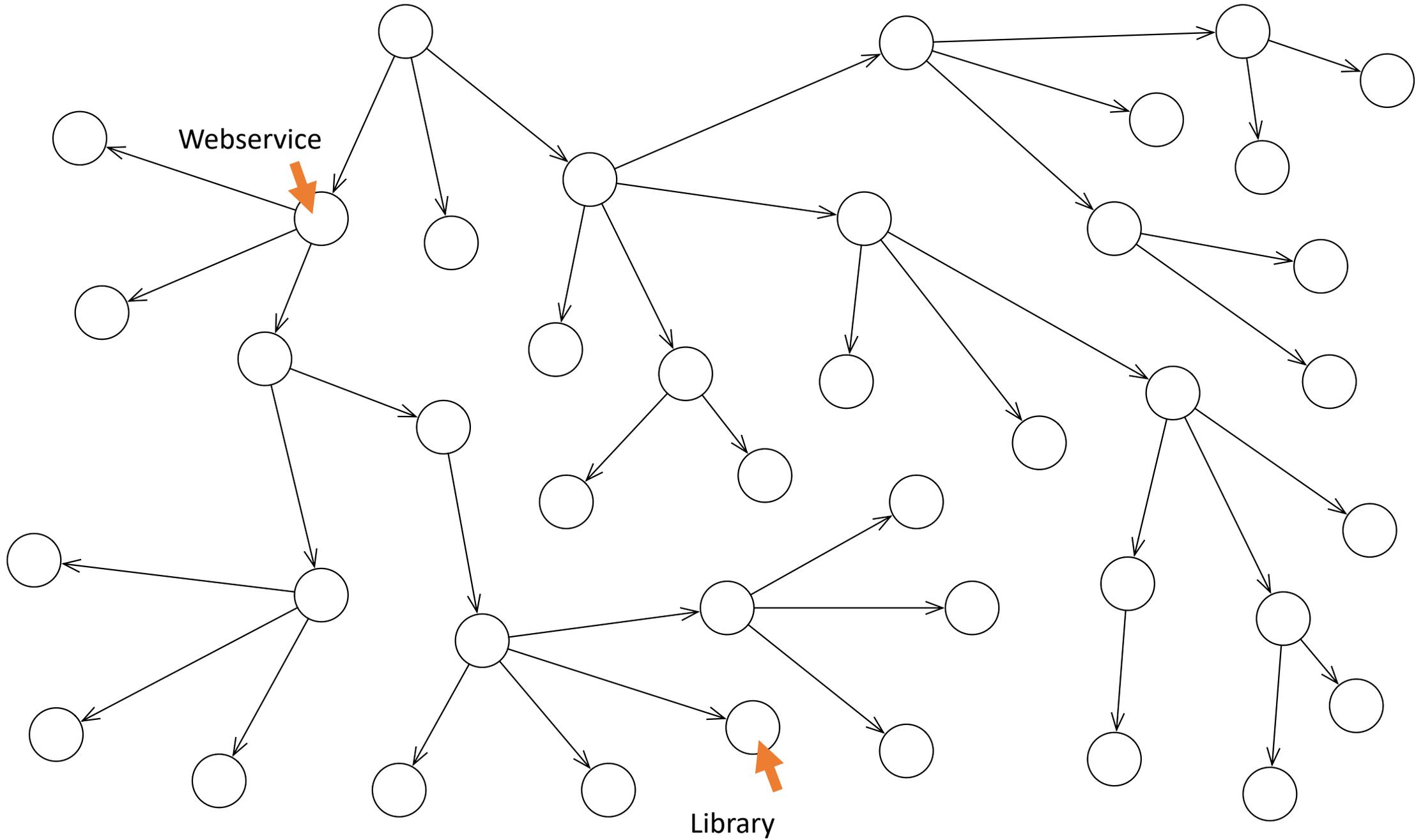


What are the challenges/ benefits/
requirements/... when you want to
apply CD for various application types?

Release Personas / Stereotypes

- Standalone Installation
 - Isolated (“Text Editor”)
 - Collaborative (“Discord”)
- Webservices (“No installation”)
- Integrated
 - Plugins
 - Libraries

There is no “one size fits all”
solution for CD.



Focus of this course

- Libraries (Friday)
- Webservices (next week, containerization)
- ...? (maybe you have another idea for your project)

Versioning

Unique Versions



<code>master</code>	First pass for next OOPP iteration	a8f3507	Sebastian Proksch ·
	Last version of last year.	83c57f2	Sebastian Proksch ·
	Refactor path information, to make them con...	69f4b4f	Sebastian Proksch ·
	Generation of group templates for a given .cs...	1bdfec9	Sebastian Proksch ·
	Final version before publishing the rubrics.	967d6c8	Sebastian Proksch ·
	First release candidate.	a7d211f	Sebastian Proksch ·
	Incorporated feedback and first complete ver...	4fa9bdd	Sebastian Proksch ·
	Split down monolithic rubric into multiple files.	ed31817	Sebastian Proksch ·
	Initial version of the rubric-template generator.	8ff4fc2	Sebastian Proksch ·

Ordered Versions

V·T·E

Windows 10 versions

Version	Codename	Marketing name	Build	Release date	Support until (and support status by color)			
					Home Pro Pro Education Pro for Workstations	Enterprise Education	LTSC ^[a]	Mobile
1507	Threshold 1	N/A	10240	July 29, 2015	May 9, 2017		October 14, 2025	N/A
1511	Threshold 2	November Update	10586	November 10, 2015	October 10, 2017	April 10, 2018	N/A	January 9, 2018
1607	Redstone 1	Anniversary Update	14393	August 2, 2016	April 10, 2018 ^[b]	April 9, 2019 ^[c]	October 13, 2026	October 9, 2018
1703	Redstone 2	Creators Update	15063	April 5, 2017 ^[d]	October 9, 2018 ^[e]	October 8, 2019 ^[f]	N/A	June 11, 2019
1709	Redstone 3	Fall Creators Update	16299 ^[g]	October 17, 2017	April 9, 2019	October 13, 2020 ^[h]		January 14, 2020
1803	Redstone 4	April 2018 Update	17134	April 30, 2018	November 12, 2019	May 11, 2021 ^[i]	January 9, 2029	N/A
1809	Redstone 5	October 2018 Update	17763	November 13, 2018 ^[j]	November 10, 2020 ^[k]			
1903	19H1	May 2019 Update	18362	May 21, 2019	December 8, 2020			
1909	19H2	November 2019 Update	18363	November 12, 2019	May 11, 2021	May 10, 2022		
2004	20H1	May 2020 Update	19041	May 27, 2020	December 14, 2021			
20H2	20H2	October 2020 Update	19042	October 20, 2020	May 10, 2022	May 9, 2023		
21H1	21H1	TBA	19043	TBA	18 months			

Legend: Old version, unsupported^[l] Older version, supported^[m] Latest version^[n] Latest preview version^[o]

[\[show\]](#)

Semantic Versioning

Given a version number **MAJOR.MINOR.PATCH**, increment...

- the **MAJOR** version when you make incompatible API changes,
- the **MINOR** version when you add functionality in a backwards compatible manner, and
- the **PATCH** version when you make backwards compatible bug fixes.

Additional labels for **pre-release** and build **metadata** are available as extensions to the MAJOR.MINOR.PATCH format.

Unique & Ordered & “Meaningful”

Example Versions

- 1 (non-semantic)
- 1.2 (non-semantic)
- 1.2.3
- 2.43.0-beta
- 0.1.3-SNAPSHOT (Maven specific?!)
- 0.0.1-20210419-134715

Interpretation and order of this version identifiers depends on the concrete ecosystem.

Dependency Resolution

Well defined Maven Version?

```
<dependency>  
  <groupId>commons-io</groupId>  
  <artifactId>commons-io</artifactId>  
  <version>1.4</version>  
</dependency>
```

Maven Version Ranges

Range	Meaning
<code>(, 1.0]</code>	$x \leq 1.0$
<code>1.0</code>	"Soft" requirement on 1.0 (just a recommendation - helps select the correct version if it matches all ranges)
<code>[1.0]</code>	Hard requirement on 1.0
<code>[1.2, 1.3]</code>	$1.2 \leq x \leq 1.3$
<code>[1.0, 2.0)</code>	$1.0 \leq x < 2.0$
<code>[1.5,)</code>	$x \geq 1.5$
<code>(, 1.0], [1.2,)</code>	$x \leq 1.0$ or $x \geq 1.2$. Multiple sets are comma-separated
<code>(, 1.1), (1.1,)</code>	This excludes 1.1 if it is known not to work in combination with this library

Deterministic Version Resolution?

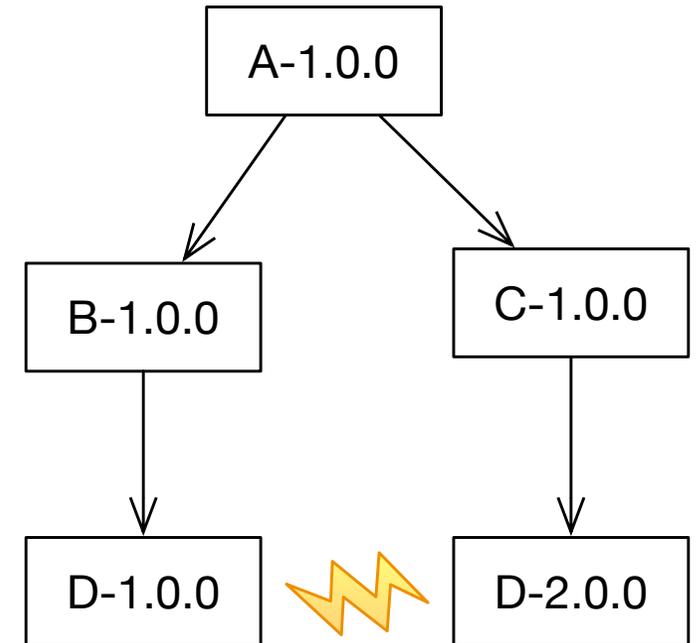
```
<dependency>  
  <groupId>commons-io</groupId>  
  <artifactId>commons-io</artifactId>  
  <version>[,1.4)</version>  
</dependency>
```

Lesson Learned #1: Fix Versions!

- Fixing specific versions is essential
- Builds become repeatable and deterministic
- Can be auto-updated, e.g., through automated PRs
- “Latest” is not a fixed version

Unexpected Effect of Transitive Dependencies

- Transitive dependencies might conflict
- Dependency resolution
 - Try to solve constraints (Matcher)
 - Use newest match
 - If no match, use declaration that is "nearest" to A
 - Might(!) work!
- Sometimes, resolution in A necessary



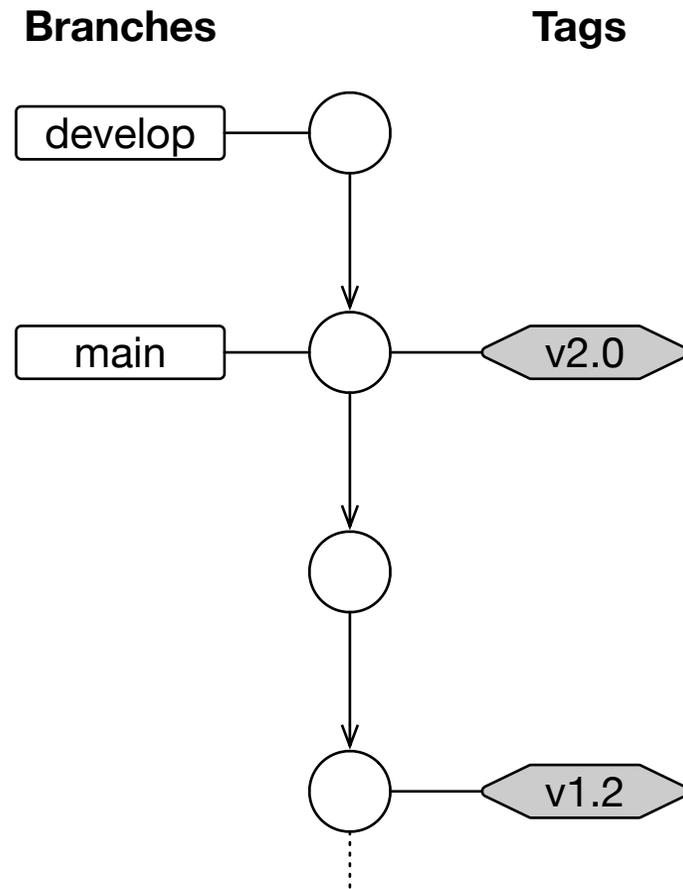
LL#2: Minimize Dependencies

- Unnecessary dependencies bloat client
- Excessive dependencies increase conflict potential
- Dependencies might not be available
(Java vs. Android)
- Use commons packages for utils and data structures

Managing Versions

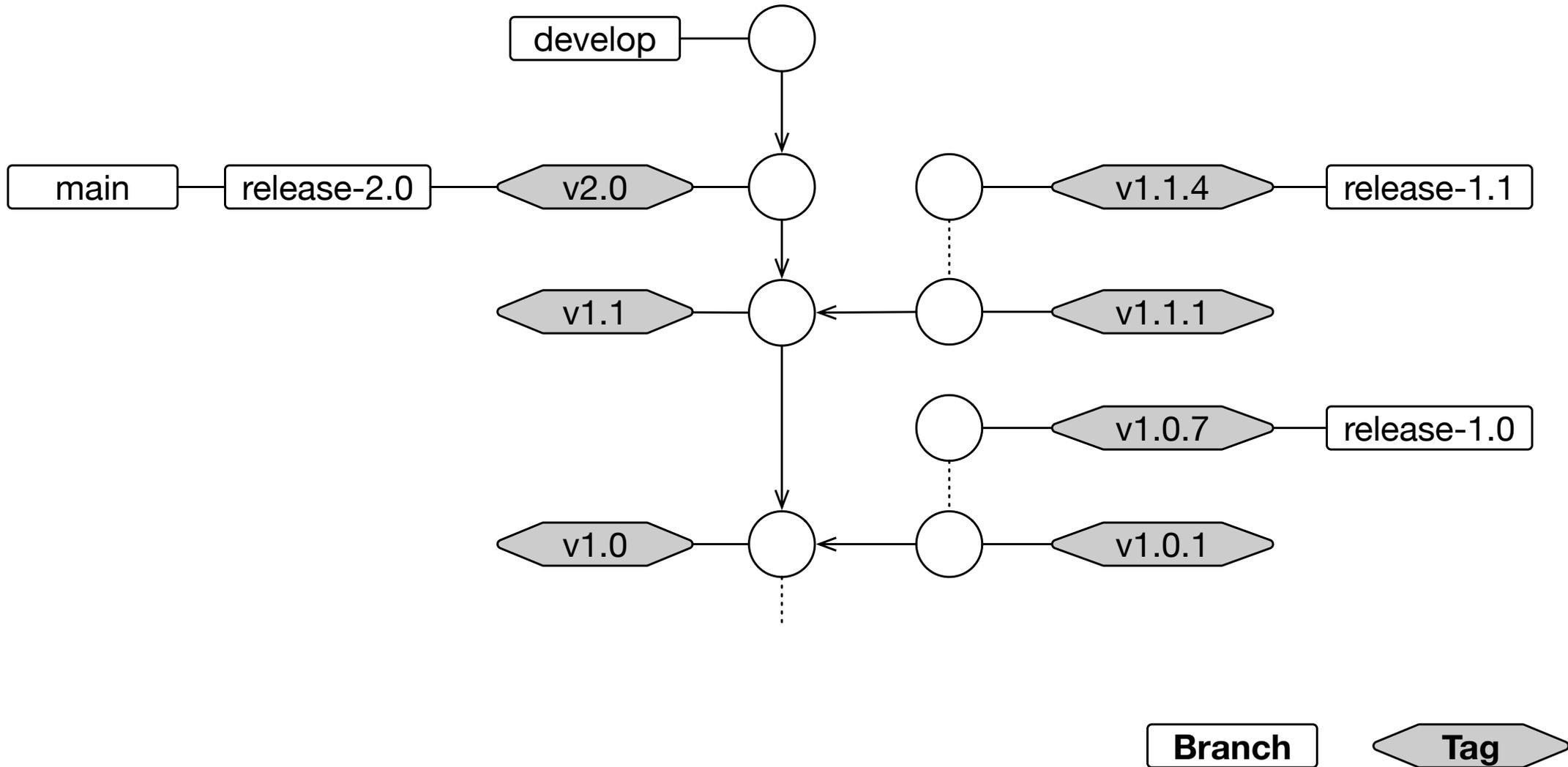
Trunk-Based Development + Release Tags

(GitHub Model)



What about
bugfix releases?

Release Branches



Time-based Versions

- Pre-releases (“Snapshots”)
- Timed Builds (e.g., “Nightlies”)
 - CI/CD evangelists would say they are not necessary
 - Used when releasing is expensive
 - .. or environment changes frequently
(or at least more frequent than product)
- Usually used in addition to proper versioning scheme

Where to store the version information?

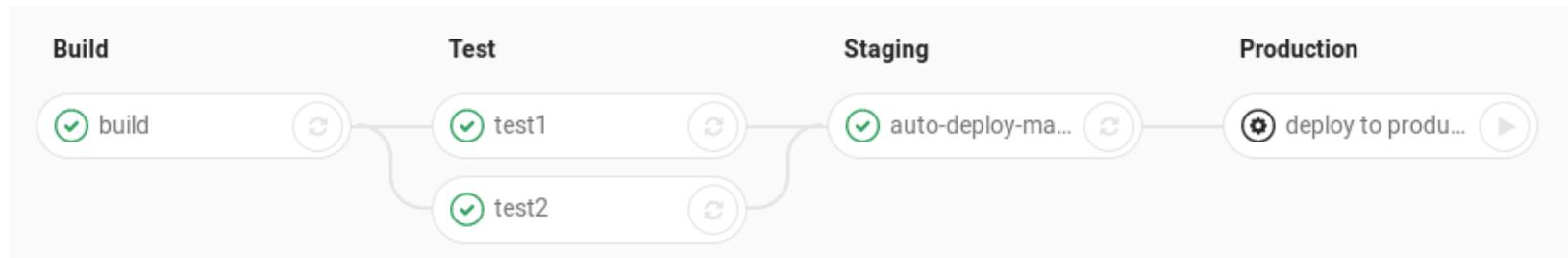
- As a static field in a class?
- In a file? pom.xml?
- As a repository tag?
- What happens should you forget to update?
- Who is bumping version numbers?
- Who gets to decide on major/minor/patch?
- Does the build server get write access to the repo?

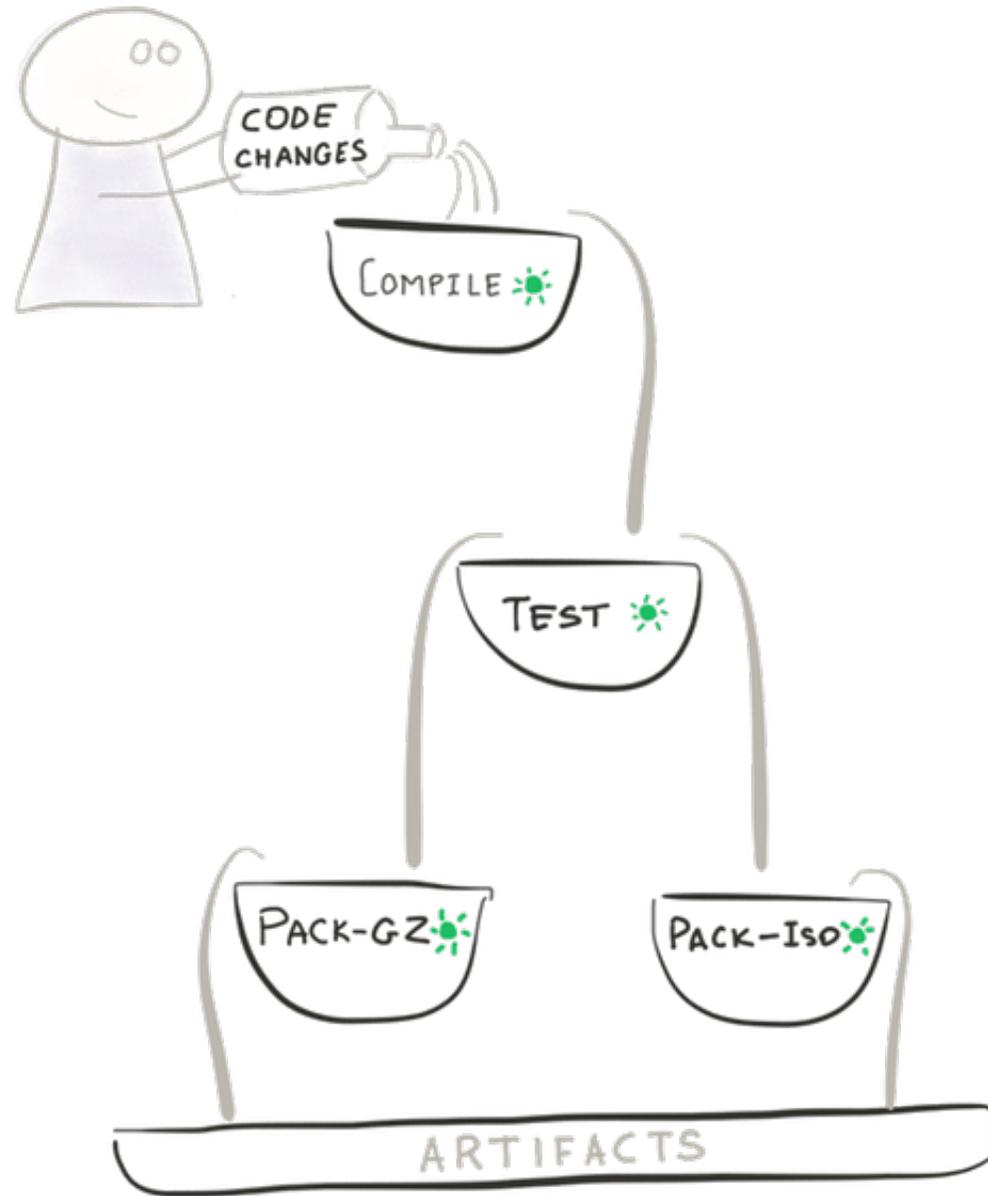
Again, no single solution
is always superior...

GitLab Pipelines

Stages & Jobs

- Jobs are smallest unit of a pipeline
- Stages group jobs
- Default stages exist (build, test, deploy)
- A pipeline orchestrates execution of stages and jobs
- Stages are run sequentially, jobs in parallel





Trigger

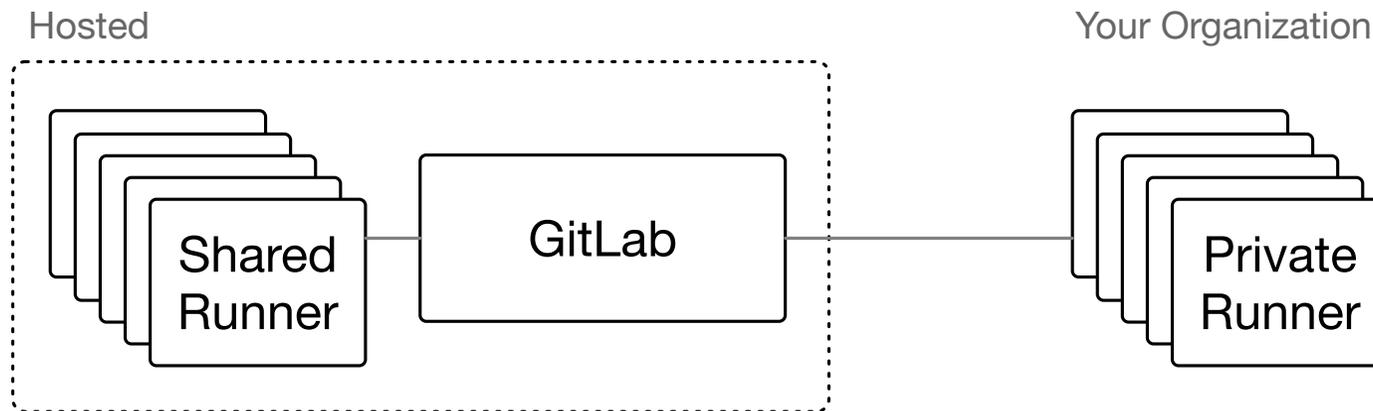
- All jobs are considered when a pipeline triggers
 - Commit
 - Manual trigger
 - API trigger
- Execution can be controlled though
- Use **only** / **except** to limit job to certain branches or tags
- Use **when** as start criterion, by default: `on_success`
(`always`, `manual`, `delayed`, `on_failure`)

Artifacts

- Every job gets access to the checked-out repository
- All changes in the working dir are isolated
- Use `artifact` directive to preserve outputs for upcoming stages (e.g., packaged jar file)
- Use `cache` directive to preserve outputs between job executions (e.g., downloaded Maven dependencies)

Pipeline Execution

- Pipeline steps are executed on runners in containers
- Base image can be changed
(e.g., Java image for compilation, Docker image for packaging, ...)



Script Line

- Prepare environment with `before_script` statements
- Actual task is done by invoking all lines in `script`
- Can be bash script, a build tool, a provisioning tool, ...

```
default:
  image: python:3.8
  before_script:
    - apt-get update
    - apt-get install -y python3-pip
    - pip install -r requirements.txt
  stages:
    - test
    - dummy
  test:
    script:
      - coverage run -m pytest
      - coverage report
```

Environments

- GitLab can deploy to environments
- Provides a full history of deployments to each environment.
- Tracks your deployments, so you always know what is deployed on your servers.
- Useful for Kubernetes, e.g., gradual rollouts

... and much more.

<https://docs.gitlab.com/ee/ci/README.html>

Keyword	Description
after_script	Override a set of commands that are executed after job.
allow_failure	Allow job to fail. A failed job does not cause the pipeline to fail.
artifacts	List of files and directories to attach to a job on success.
before_script	Override a set of commands that are executed before job.
cache	List of files that should be cached between jobs.
coverage	Code coverage settings for a given job.
dependencies	Restrict which artifacts are passed to a job.
environment	Name of an environment to which the job belongs.
except	Limit when jobs are not created.
extends	Configuration entries that this job extends.
image	Use Docker images.
include	Include external YAML files.
inherit	Select which global defaults all jobs inherit.
interruptible	Defines if a job can be canceled.
needs	Execute jobs earlier than the stage.
only	Limit when jobs are created.
pages	Upload the result of a job to use as a website.
parallel	How many instances of a job should be run.
release	Instructs the runner to generate a release artifact.
resource_group	Limit job concurrency.
retry	When and how many times a job should be retried.
rules	List of conditions to evaluate and actions to take.
script	Shell script that is executed by a job.
secrets	The CI/CD secrets the job needs.
services	Use Docker services images.
stage	Defines a job stage.
tags	List of tags that are used to select jobs.
timeout	Define a custom job-level timeout.
trigger	Defines a downstream pipeline trigger.
variables	Define job variables on a job level.
when	When to run job.

GitLab Docs 13.10 [Get free trial](#)

- Issues
- Merge requests
- Operations
- CI/CD
 - Get started
 - Pipelines
 - Jobs
 - Variables
 - Environments and deployments
 - Runners
 - Cache and artifacts
 - .gitlab-ci.yml reference**

- Validate syntax
- Pipeline Editor
- Include examples
- Docker
- Integrate a database

GitLab Docs > GitLab CI/CD > Keyword reference for the .gitlab-ci.yml file

Keyword reference for the .gitlab-ci.yml file ALL TIERS

This document lists the configuration options for your GitLab `.gitlab-ci.yml` file.

- For a quick introduction to GitLab CI/CD, follow the [quick start guide](#).
- For a collection of examples, see [GitLab CI/CD Examples](#).
- To view a large `.gitlab-ci.yml` file used in an enterprise, see the [.gitlab-ci.yml file for gitlab](#).

When you are editing your `.gitlab-ci.yml` file, you can validate it with the [CI Lint](#) tool.

Job keywords

A job is defined as a list of keywords that define the job's behavior.

The keywords available for jobs are:

On this page

- Job keywords
 - Unavailable names for jobs
 - Custom default keyword values
- Global keywords
 - stages
 - workflow
 - workflow:rules templates
 - Switch between branch pipelines and merge request pipelines
- include
 - Variables with include
 - include:local
 - include:file
 - Multiple files from a project
 - include:remote
 - include:template
 - Nested includes
 - Additional includes examples

Deploy Targets

File Share

- Programs
- Websites
- Use simple transfer utilities, like `ftp` or `scp`

Artifact Repository

- A released artifact has an unique name
(Maven: Group ID, Artifact ID, version)
- Central Repository to store and manage artifacts
 - “Mavenized” Git Repository
 - Package/Container Registry of Repository Providers
 - Specialized Repository Software, like Artifactory
 - Maven Central
- Gives clients a unified access to well-defined deps

Application Server / Cluster

- Deployment/Configurations
- Often either a final, manual trigger...
- ... or webhooks to trigger updates
- Automated update is key for continuous experimentation
- GitLab: `when: delayed` trigger for gradual rollout

Outlook

Follow-up Problems

- Client heterogeneity
- Data Format Migration
- Endpoint Versioning
- API Design & API Deprecation
- Detecting Breaking Changes (Semantic)
- ...

Conclusion

After today's lecture, you can...

- tell the difference between continuous integration, delivery, deployment, and experimentation
- describe a basic delivery pipeline
- explain the concepts behind semantic versioning
- describe basic release/version strategies
- describe the basic elements of a GitLab pipeline and you know how to find more information