Foto: Thomas Josek

Software Engineering

Deployment and Operations

Software & Systems Engineering | Prof. Dr. Andreas Vogelsang | 20.12.2023

@andivogelsang

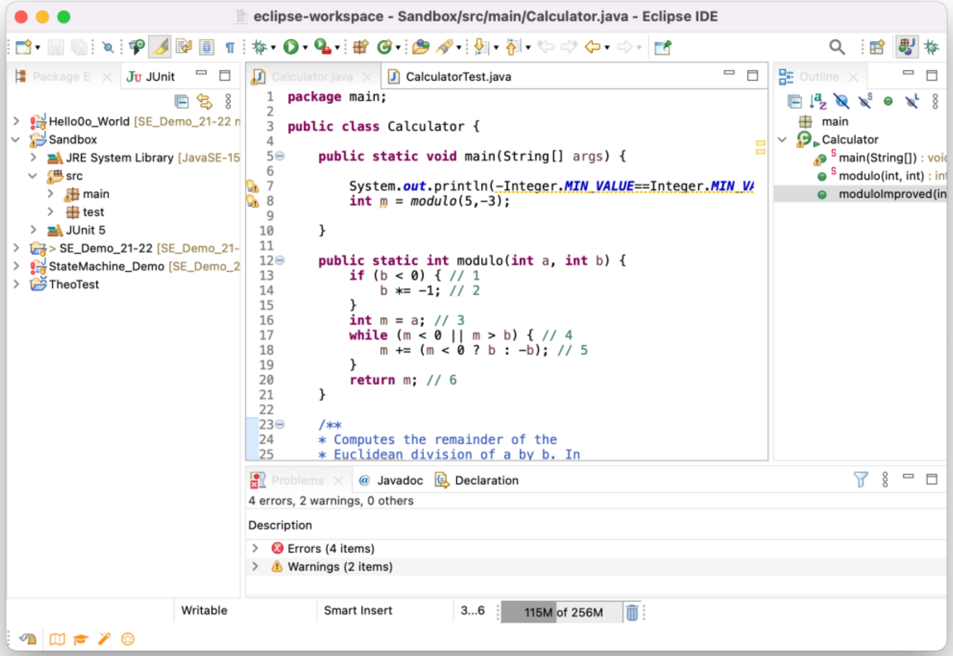
vogelsang@cs.uni-koeln.de

Learning Goals for Today

- Know what packaging and deployment is
- Know how to manage project dependencies
- Know containerization as a technique to package and deploy software systems
- Know the principle of DevOps and its main characteristics
- Know what Continuous Integration, Delivery, and Deployment is

Packaging and Deployment

Packaging and Deployment



```
package main;

public class Calculator {

    public static void main(String[] args) {

        System.out.println(-Integer.MIN_VALUE==Integer.MIN_VA
        int m = modulo(5,-3);

    }

    public static int modulo(int a, int b) {
        if (b < 0) { // 1
            b *= -1; // 2
        }
        int m = a; // 3
        while (m < 0 || m > b) { // 4
            m += (m < 0 ? b : -b); // 5
        }
        return m; // 6
    }

    /**
    * Computes the remainder of the
    * Euclidean division of a by b. In
```

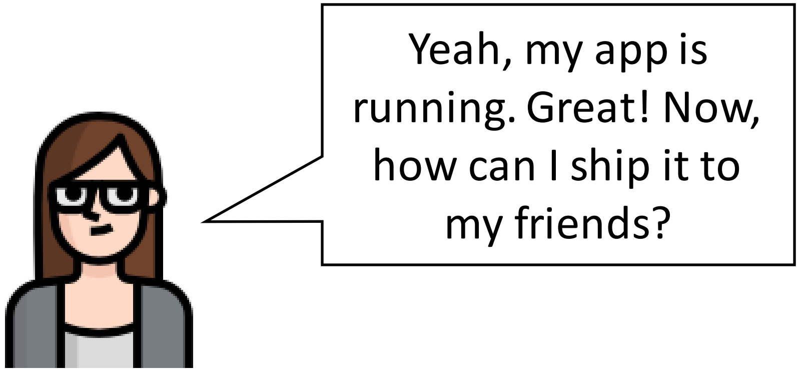
4 errors, 2 warnings, 0 others

Description

- Errors (4 items)
- Warnings (2 items)

What happens if you click on “run” in your IDE?

- **Compilation**: Transform the source code into a target language (machine code or intermediate language)
- **Execution:** Either directly on the machine or within an interpreter (virtual machine)


Yeah, my app is running. Great! Now, how can I ship it to my friends?

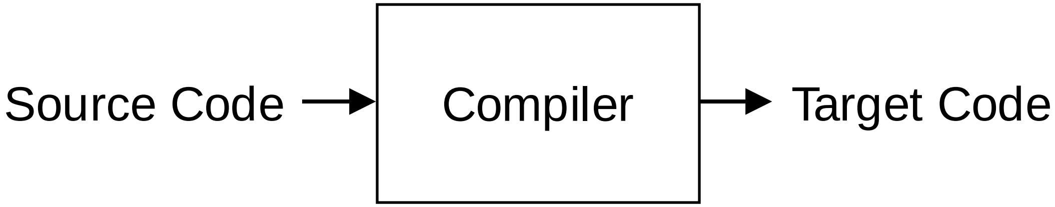
Compilation vs. Interpretation

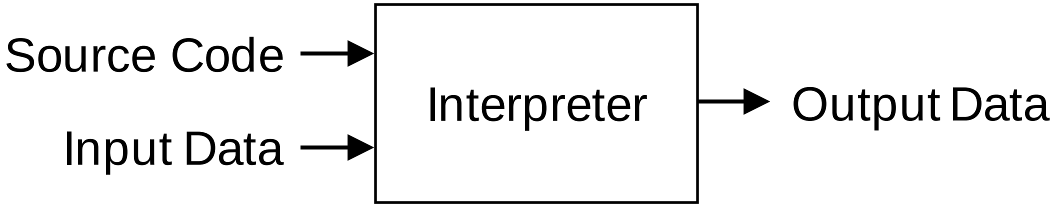
Compilation

- C/C++/Go/Rust/Swift to machine code
- Java/Groovy/Kotlin/Scala/Clojure to Java bytecode

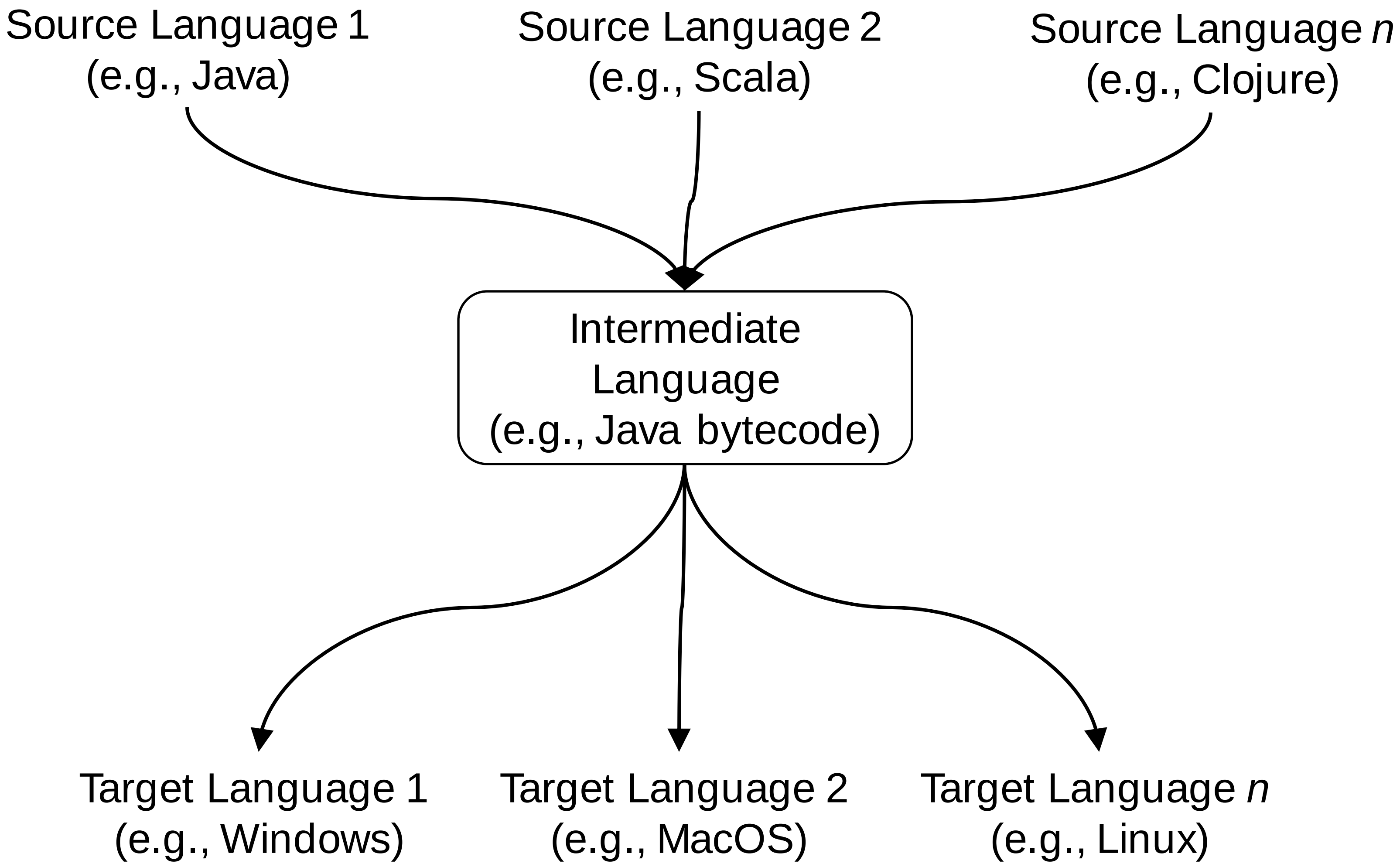
Interpretation

- of source code: Ruby/Python/Perl/PHP/Matlab
- of bytecode: Java Virtual Machine (JVM)





The Power of Intermediate Languages



Source Language 1
(e.g., Java)

Source Language 2
(e.g., Scala)

Source Language *n*
(e.g., Clojure)

Intermediate
Language
(e.g., Java bytecode)

Target Language 1
(e.g., Windows)

Target Language 2
(e.g., MacOS)

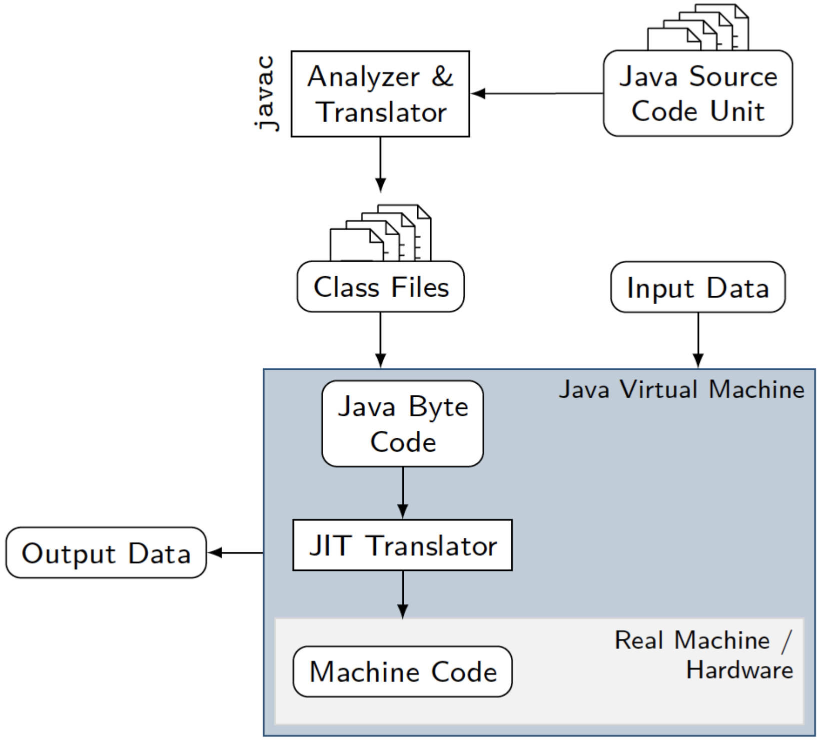
Target Language *n*
(e.g., Linux)

Universität
zu Köln

Java Compilation and Execution

Goals of Compiler Optimizations

- fast execution
- low memory / energy consumption
- small binaries (fast start/download/updates)
- desired for both compiler (compile time) and compiled program (run time)



Compile Time vs. Run Time

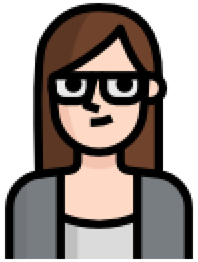
run time: when program or software is executed

compile time: during (ahead-of-time) compilation

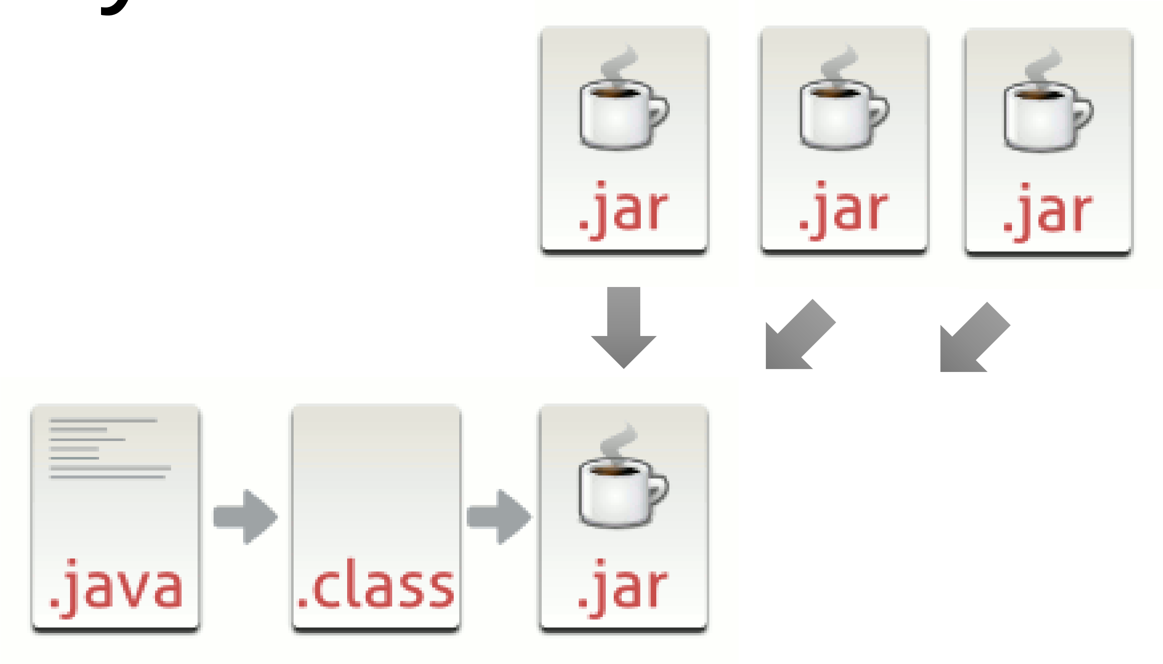
Just-in-time Compilations

- often executed code is compiled at run time
- warm-up time: execution is slower when new code is executed

Packaging and Deployment

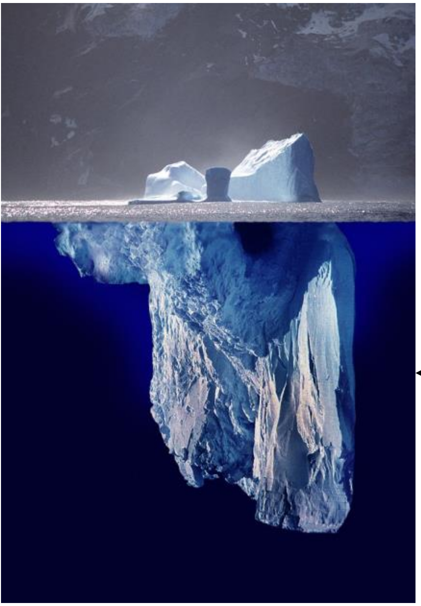


Nice. So, I just have to compile my Java program and everyone with a JVM can execute my Java application?



META-INF/MANIFEST.MF

```
Manifest-Version: 1.0
Main-Class: Application
Class-Path: core.jar lib/
```



Your code →

← Dependencies

Build Automation Tools

Build automation

Build automation is the process of automating the creation of a software build and the associated processes including compiling computer source code into binary code, packaging binary code, and running automated tests.

Maven™

pom.xml

```
<project>
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.mycompany.app</groupId>
    <artifactId>my-app</artifactId>
    <version>1.0</version>
    <description>Maven example</description>

    …

    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.12</version>
            <scope>test</scope>
        </dependency>
    </dependencies>
</project>
```

Build Automation Tools – Maven

Maven Commands

1. **clean**: delete target directory
2. **validate**: validate if the project is correct (e.g., check code formatting)
3. **compile**: compile source code; classes stored in `target/classes`
4. **test**: run unit tests
5. **package**: take the compiled code and package it in its distributable format, e.g. JAR, WAR
6. **verify**: run any checks to verify the package is valid and meets quality criteria (e.g., integration tests)
7. **install**: install the package into the local repository
8. **deploy**: copies the final package to the remote repository

Execution order

Maven runs the commands in order from top to bottom except for `clean` (e.g., `verify` includes `validate`, `compile`, `test`, `package`)

Convention over configuration

Maven depends a lot on conventions (e.g., where tests, libs, sources are located)

System Building

System Building [Sommerville]

System building is the process of creating a complete, executable system by compiling and linking the system components, external libraries, configuration files, and other information.

Building involves three platforms

- **development system**: compilers and editors used on the developer's system to test prior to commit
- **build server**: server to build and distribute executable versions, triggered by commits or schedule (i.e., nightly builds)
- **target environment**: intended platform for executable system (e.g., ECU in a car)

Tooling for System Building

- **build script generation**: identify dependent components, automated generation or tool support for creation and editing
- **version control system integration**: checkout required versions of components
- **minimal recompilation**: determine which parts need to be recompiled
- **executable system creation**: compilation and linking
- **test automation**: run automated tests (e.g., unit tests)
- **reporting**: reports about success or failure of builds and tests
- **documentation generation**: release notes, help pages

Continuous Integration

Continuous Integration [Sommerville]

“Agile methods recommend that very frequent system builds should be carried out, with automated testing used to discover software problems. Frequent builds are part of a process of continuous integration [...].”

Continuous Integration Tools

- Jenkins (2011–)
- Travis CI (2011–)
- GitLab (2014–)
- GitHub (2018–)

Steps in Continuous Integration

- clone/fetch from version control
- if feasible: build and run automated tests, if it fails others are responsible
- apply changes
- build and run automated tests locally, if it fails continue editing
- if local tests pass, commit to feature branch in version control
- commit triggers build server, if it fails continue editing
- if tests pass (and code review approves changes), merge branch into main development branch

Infrastructure/Configuration as Code

Infrastructure/Configuration as Code

Infrastructure and build configuration are managed in files in the version control system

- **Basic Info:** Language and Language Version, Repositories, Compiler
- **Build Process:** Steps necessary to build the system (from source code to executable)
- **Quality Assurance:** Execution of Tests and Checks
- **Deployment:** Copy the built executable to a (productive) system

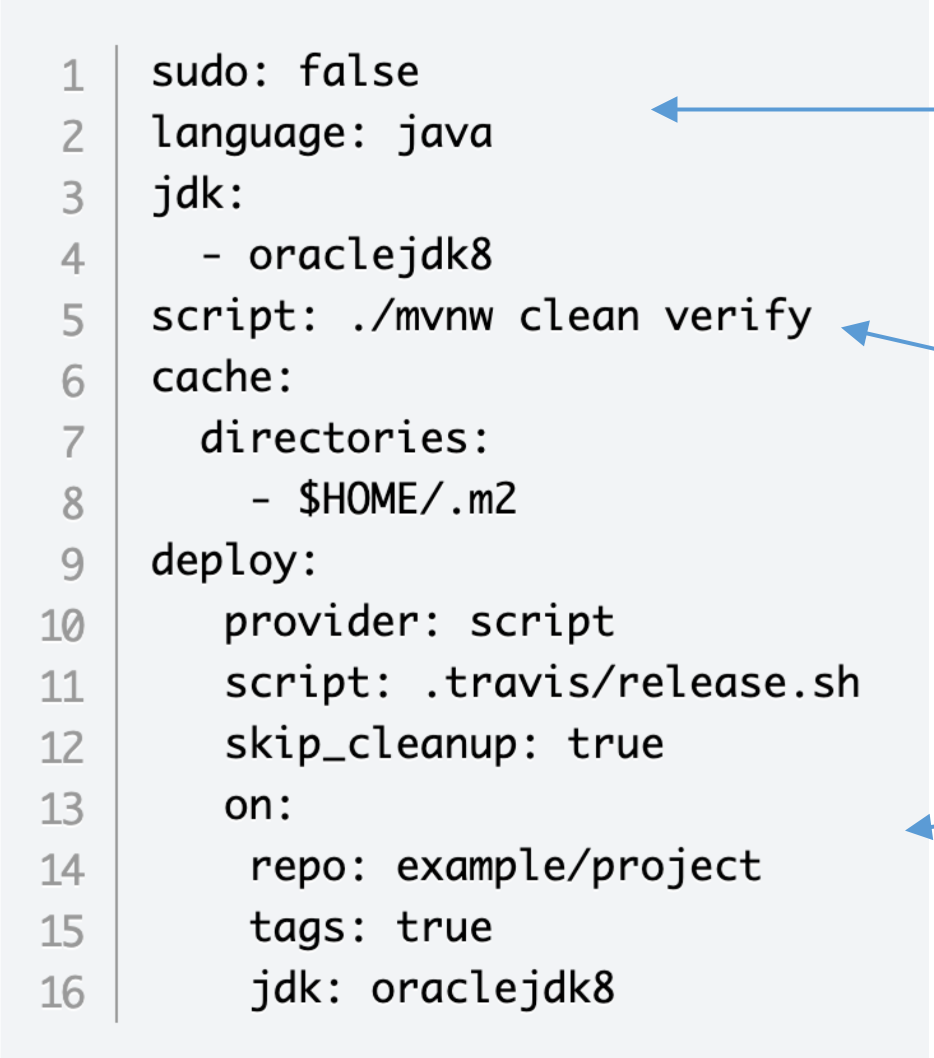
Motivation / Advantages

Consistent and shared infrastructure for testing, development, and deployment.

Example – Configuration in Travis CI

.travis.yml

Example for a Java-Maven Project



```yaml
sudo: false
language: java
jdk:
  - oraclejdk8
script: ./mvnw clean verify
cache:
  directories:
    - $HOME/.m2
deploy:
   provider: script
   script: .travis/release.sh
   skip_cleanup: true
   on:
    repo: example/project
    tags: true
    jdk: oraclejdk8
```

Basic Info

Compiling the code,
execution of (Unit)-Tests,
packaging (e.g., as .jar),
maybe integration test

Deployment defined in a script (release.sh);
Script is executed whenever a version is assigned a tag

DevOps

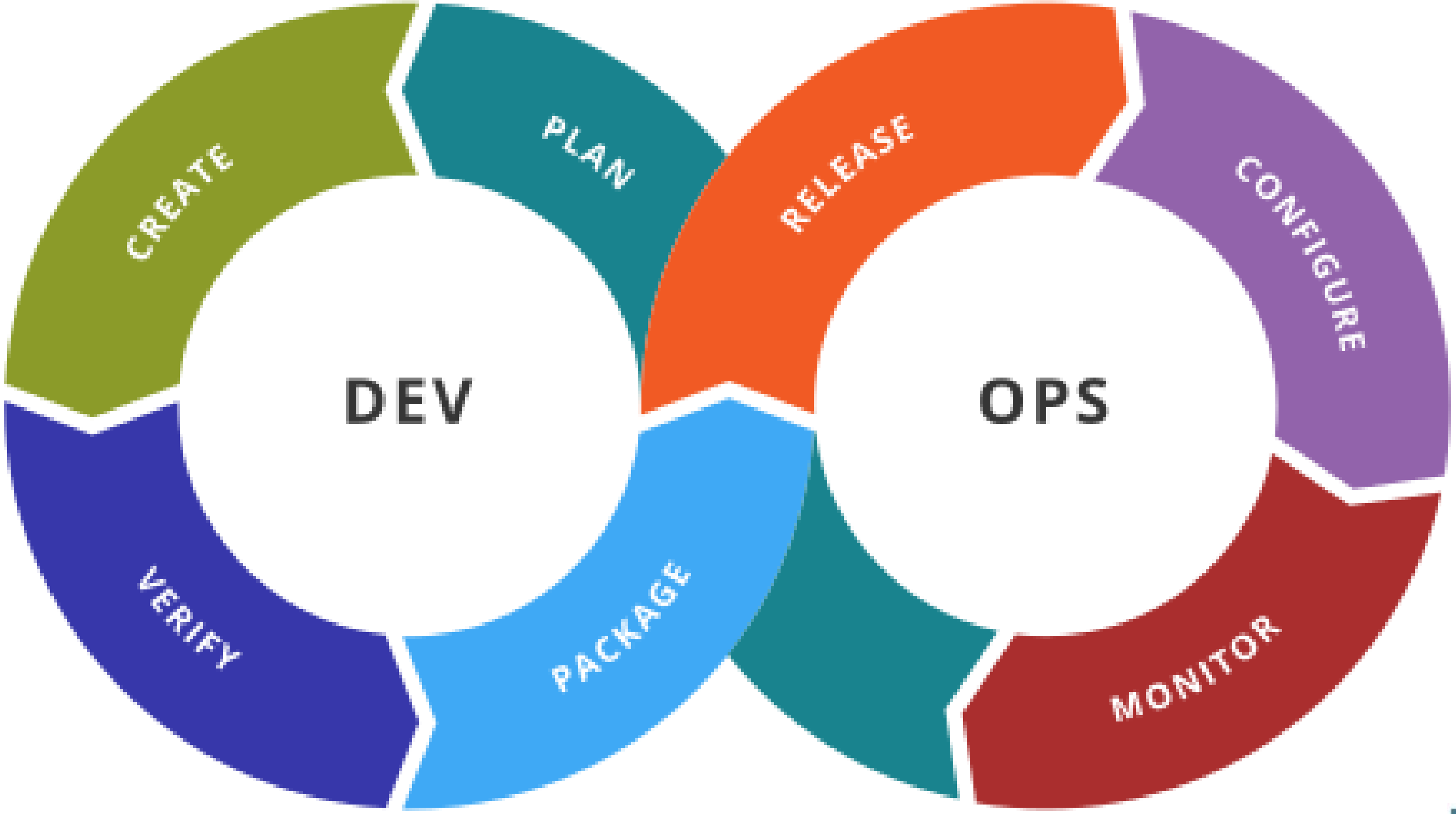
Motivation / Problem

- If software fails:
  - programmers blame administrators for misconfiguration
  - administrators blame programmers for erroneous software
- Programmers want frequent updates
- Administrators follow the slogan: “never change a running system”
- Customers and users want a single responsibility
- Shorter update cycles

DevOps

- Promoted in agile development
- **Dev**: development by programmers
- **Ops**: operation (Betrieb) by administrators
- **DevOps**: teams that are responsible for both, development and operations
- Goal: avoid blaming each other by shared responsibility

DevOps

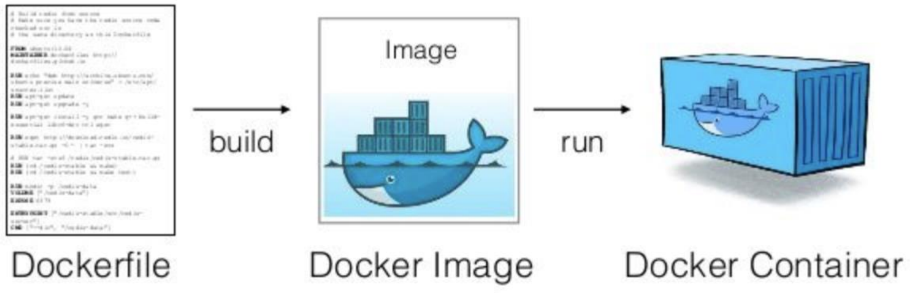

PLAN
CREATE
VERIFY
PACKAGE
RELEASE
CONFIGURE
MONITOR
DEV
OPS

It works on my machine…

Software Containerization

Software Containerization

Containerization is the packaging of software code with just the operating system (OS) libraries and dependencies required to run the code to create a single lightweight executable—called a **container**—that runs consistently on any infrastructure.



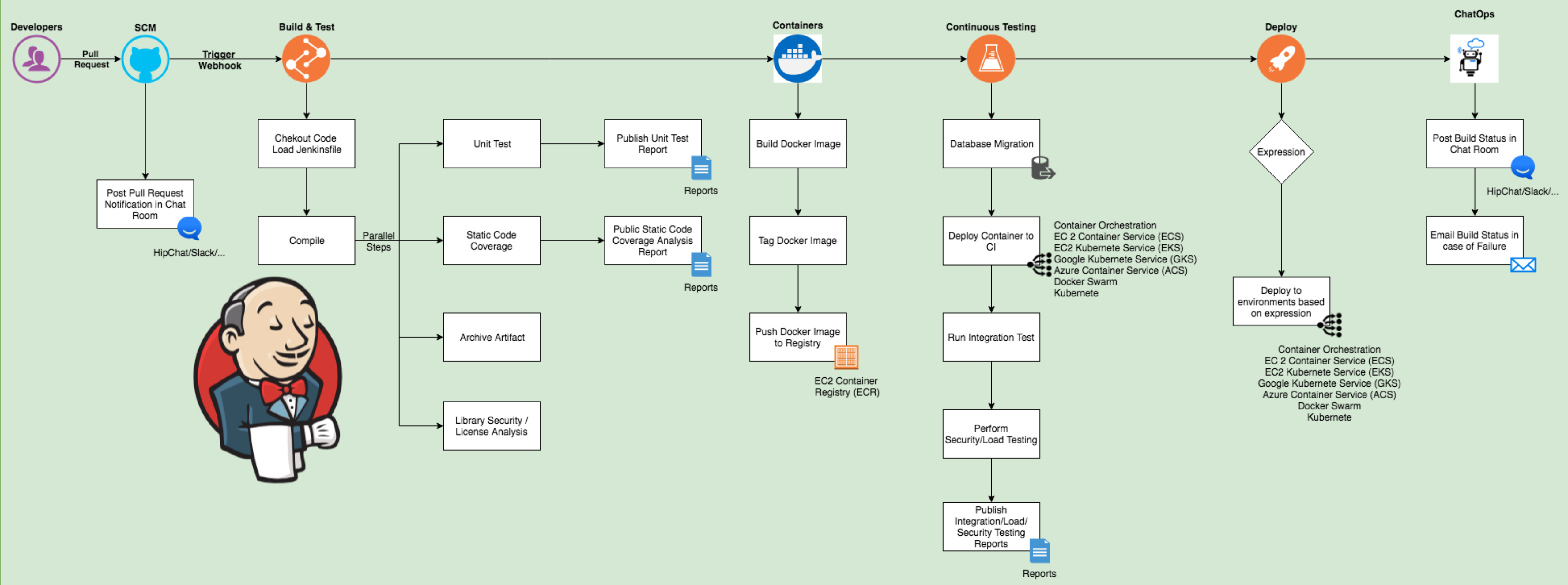
Docker

- Lightweight virtual machine
- Contains entire runnable software, incl. all dependencies and configurations
- Used in development and production
- Sub-second launch time
- Explicit control over shared disks and network connections

Terms

- **Container**: A runtime instance of a docker image
- **Image**: A package with all the dependencies and information needed to create a container
- **Dockerfile**: A text file that contains instructions for building a Docker image.

Deployment Pipeline



Developers

Pull Request

SCM

Post Pull Request Notification in Chat Room

HipChat/Slack/...

Trigger Webhook

Build & Test

Chekout Code Load Jenkinsfile

Compile

Parallel Steps

Unit Test

Static Code Coverage

Archive Artifact

Library Security / License Analysis

Publish Unit Test Report

Reports

Public Static Code Coverage Analysis Report

Reports

Containers

Build Docker Image

Tag Docker Image

Push Docker Image to Registry

EC2 Container Registry (ECR)

Continuous Testing

Database Migration

Deploy Container to CI

Container Orchestration
EC 2 Container Service (ECS)
EC2 Kubernete Service (EKS)
Google Kubernete Service (GKS)
Azure Container Service (ACS)
Docker Swarm
Kubernete

Run Integration Test

Perform Security/Load Testing

Publish Integration/Load/ Security Testing Reports

Reports

Deploy

Expression

Deploy to environments based on expression

Container Orchestration
EC 2 Container Service (ECS)
EC2 Kubernete Service (EKS)
Google Kubernete Service (GKS)
Azure Container Service (ACS)
Docker Swarm
Kubernete

ChatOps

Post Build Status in Chat Room

HipChat/Slack/...

Email Build Status in case of Failure

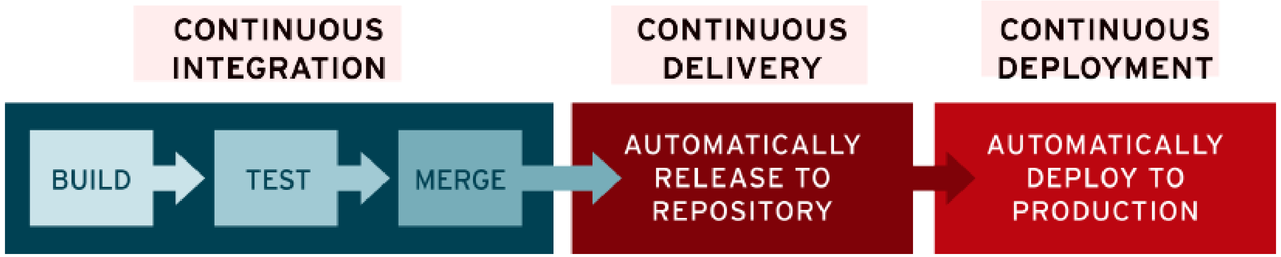
Continuous-X

Continuous-X

Continuous Integration: the practice of merging all developers' working copies to a shared mainline several times a day

Continuous Delivery: the practice of releasing a new version of the software several times a day

Continuous Deployment: the practice of deploying a new version of the software several times a day

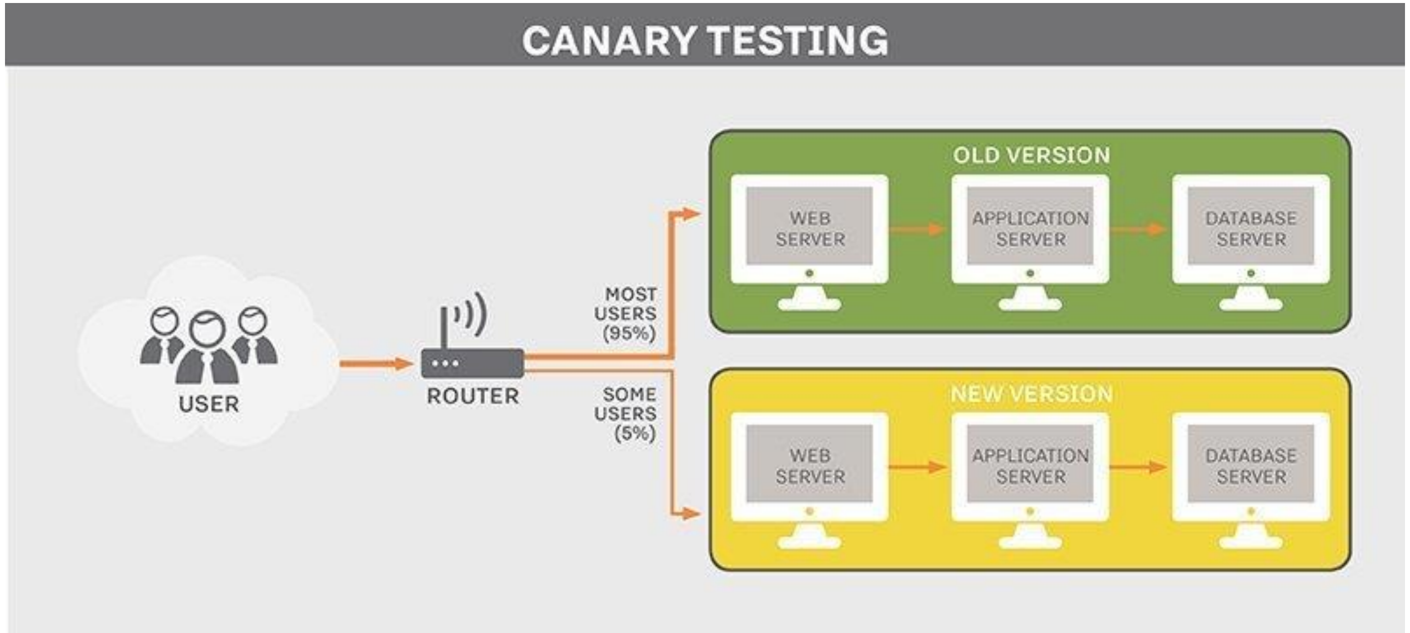


Deployment in Practice

Canary Releases

Canary Releases

Canary release is a technique to reduce the risk of introducing a new software version in production by slowly rolling out the change to a small subset of users before rolling it out to the entire infrastructure and making it available to everybody.



Application

- Mostly applied to test changes in the back-end (e.g., new algorithms, large refactoring, redesigns)
- Usually, a step in the regular development process

Universität zu Köln

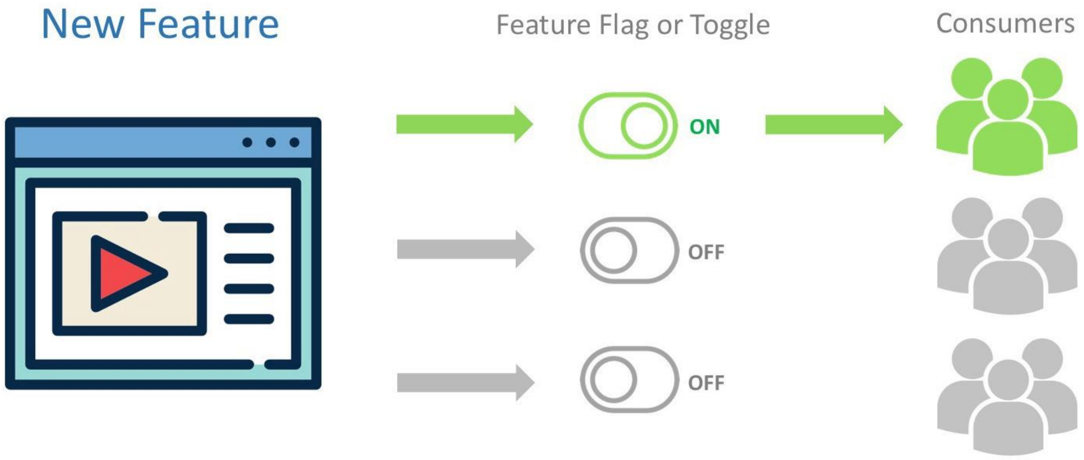
Dark Launches

Dark Launches

users aren’t aware they are testing the new feature; often, nothing highlights the new functionality — hence the term *dark* launching.

Feature Toggles

Mechanism to activate or deactivate features in code

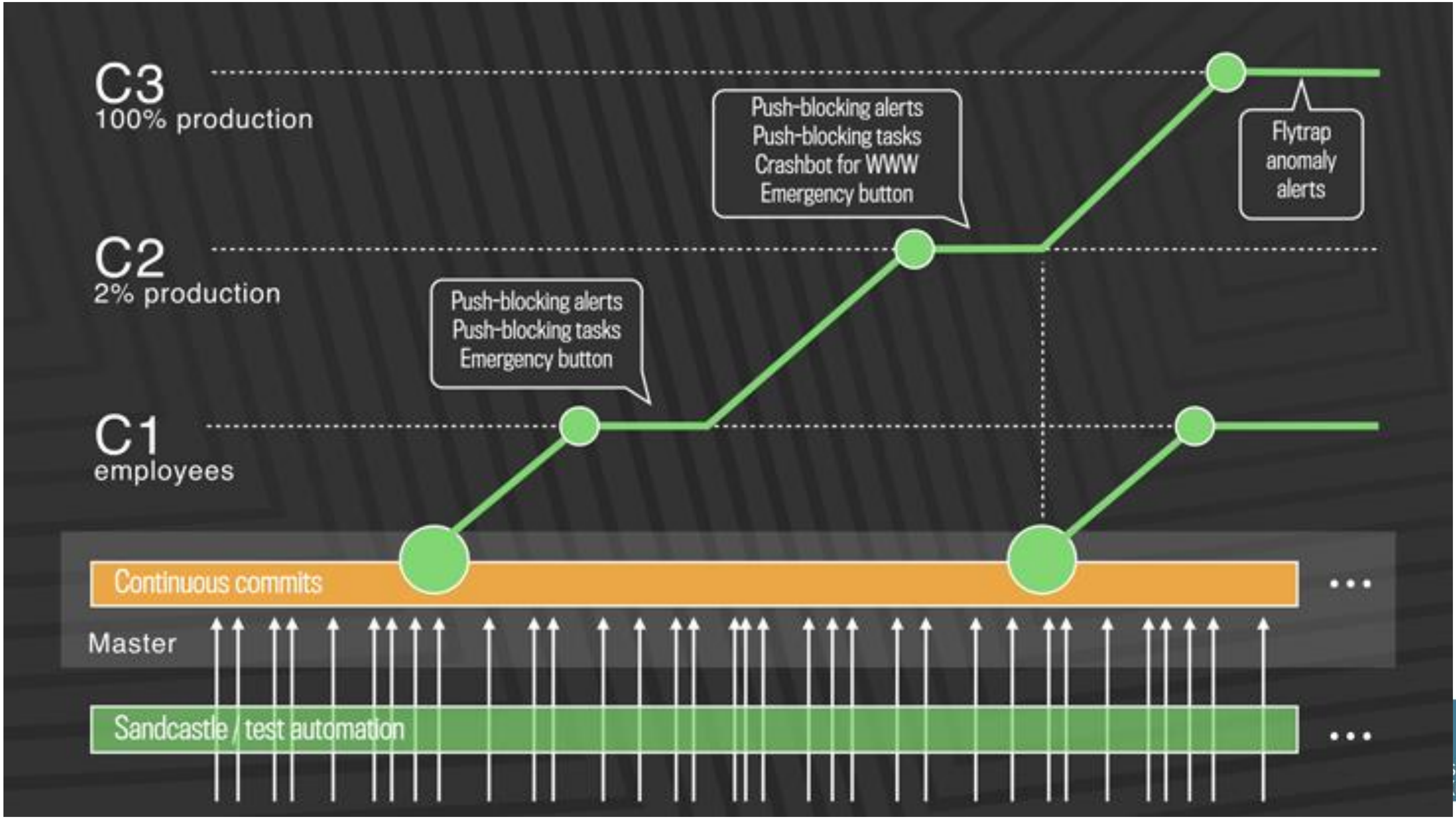


Justin Baker, 2016

Application

- Mostly applied to test new user features (new UI elements, new functionality)
- Usually, only done for selected features

Canary Releases at Facebook



Canary Releases at Netflix

- 60,000 configuration changes per day. 4,000 commits per day.
- Every commit creates an Amazon Machine Image (AMI)
- The AMI is automatically deployed to Red/Black Cluster.
- Automatic canary tests are executed, if OK, change to new version, if not, rollback the commit.

Summary

Deployment and Operations

- Packaging and deploying software is challenging
- Automation is key to manage dependencies, configurations, and deployment
- DevOps principles bring development and operations closer together
- Continuous-X allows for faster and more secure evolution of software systems