Continuous Delivery for Java

Why?

Agenda

- Agile
- Continuous delivery
- DevOps
- Docker
- Git
- Build tools
- Artifact repositories
- Jenkins
- Code quality
- Testing
- Testing on production
- Monitoring
- Security
- Issue tracking

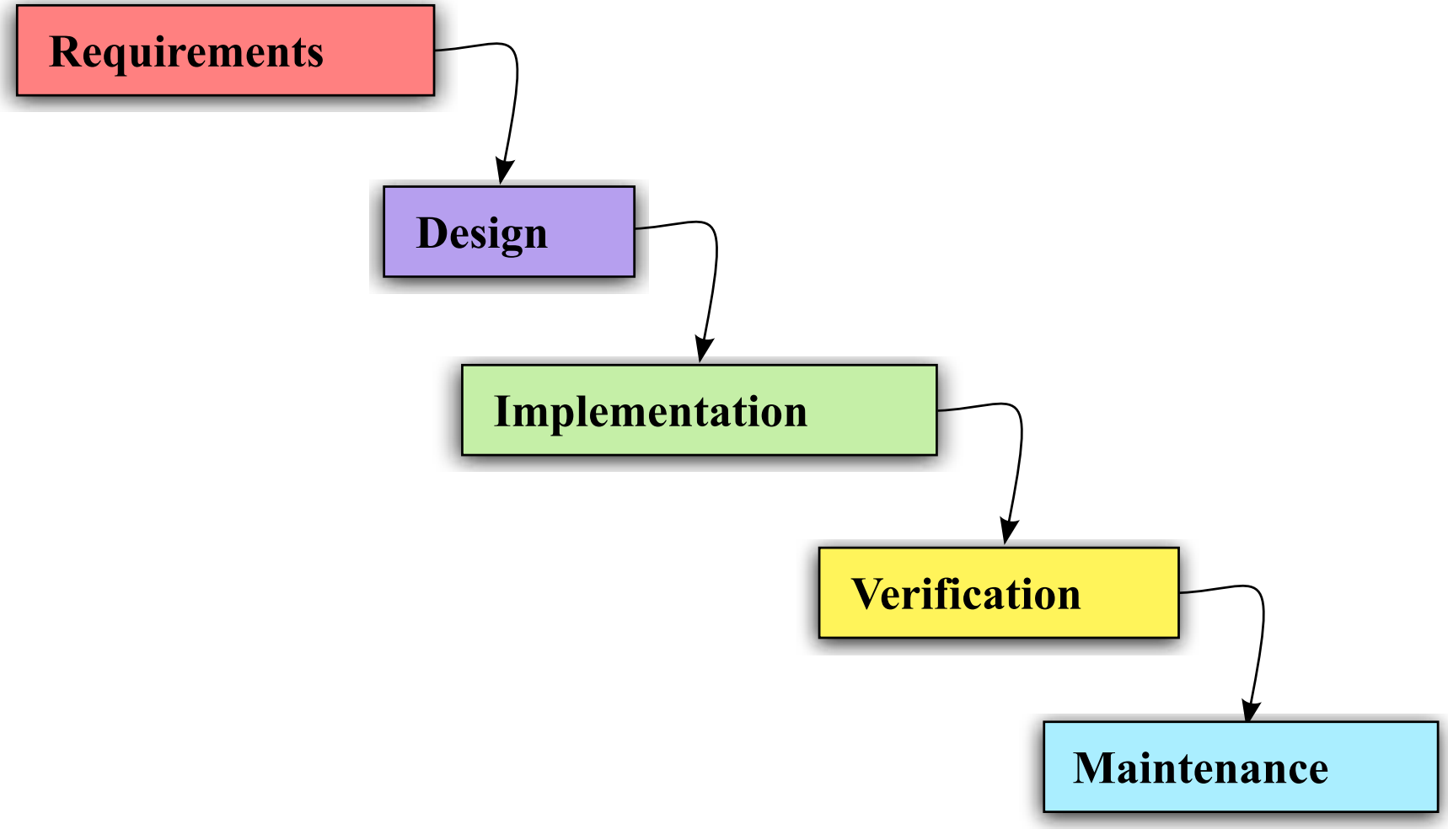
- Issue tracking
- Documentation
- Conclusion

Rules of engagement

- Training hours
- Lunches
- Phones
- Training material
- Evaluation
- It's your course!
- Have fun!

Agile

Before Agile


Requirements
Design
Implementation
Verification
Maintenance

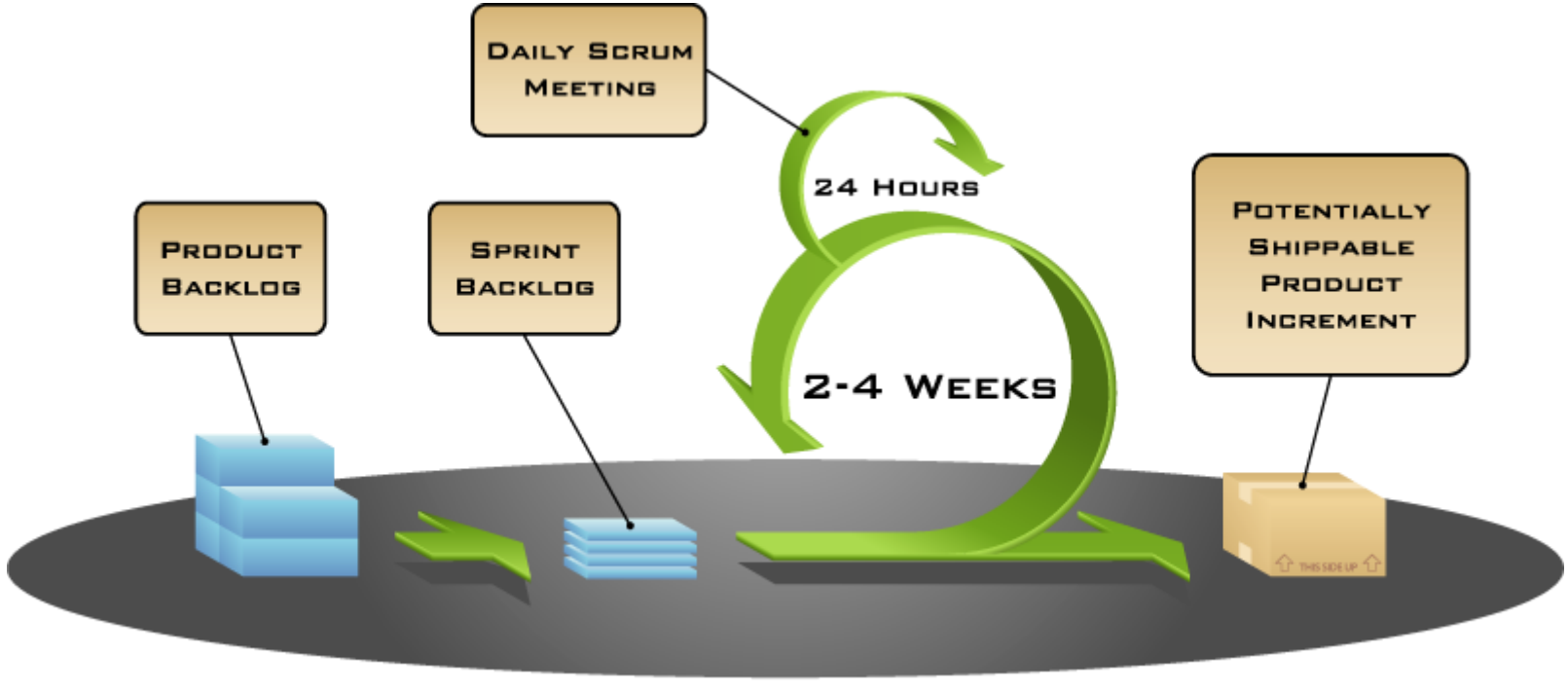
What happens if the implementation is finished?


EXPECTATIONS
VS.
REALITY


A CHANGE MAY BE JUST AROUND THE CORNER

The waterfall methodology has quite some disadvantages

Working Agile

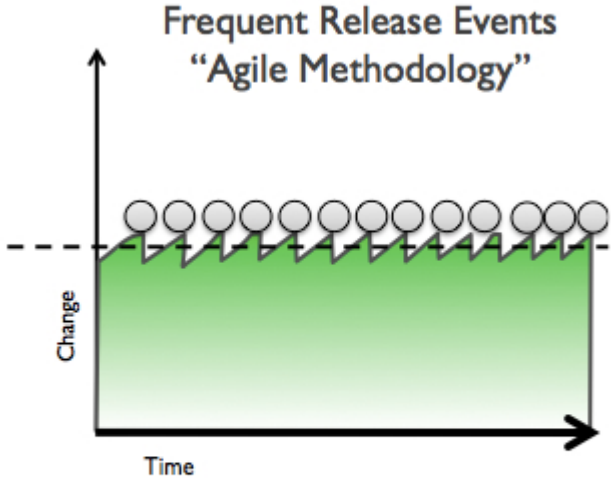

Daily Scrum Meeting
24 Hours
Product Backlog
Sprint Backlog
2-4 Weeks
Potentially Shippable Product Increment

Some Agile methodologies

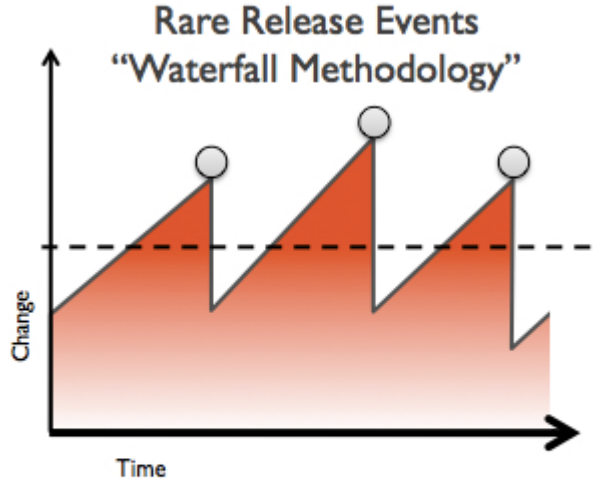
- Scrum
- Kanban

Agile helps us to

- Respond quickly to changing requirements
- Receive faster customer feedback
- Incrementally add small features to our application



Smoother Effort
Less Risk



Effort Peaks
High Risk

What happens with the new features?

Conclusion

- Features are build incrementally
- Features are stored on a shelf
- Twice a year the collection of features on the shelf is deployed to production
- Is that what we and our customers want?

Continuous Delivery

What is continuous delivery?

> *Continuous Delivery is the ability to get changes of all types—including new features, configuration changes, bug fixes and experiments—into production, or into the hands of users, safely and quickly in a sustainable way.*
>
> *https://continuousdelivery.com/*

Continuous delivery versus continuous deployment

- With continuous delivery changes are deployed to production **after pressing a button**
- With continuous deployment changes are deployed to production **automatically**

Continuous delivery versus continuous deployment

- Continuous delivery
  - If a change is committed to Git some processing takes place and then the change waits for manual approval before it's deployed to production
- Continuous deployment
  - If a change is committed to Git some processing takes place and in the end the change is deployed to production

Why?

- To immediately deploy new features to production
- To make sure the exact same application and configuration is deployed on all environments
- Deploying smaller changes to production
- Delivering value for end users

What to deliver?

What to deliver?

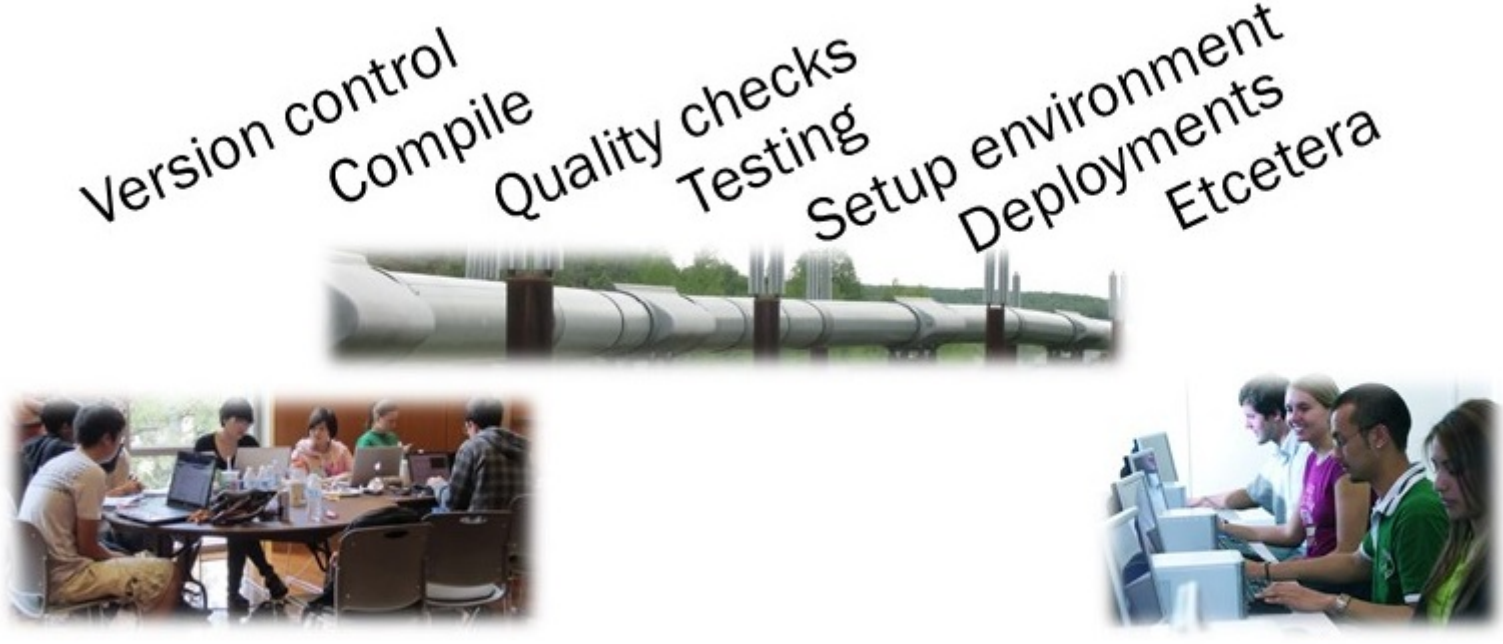
- Most organizations start with just the application (jar/war...)
- But what about Java, the application server etcetera?
- Deliver as much as possible, for instance by using Docker
- Ideally it should be possible to automatically deploy your application to a new machine/environment

InfoQ maturity model

The Continuous Delivery Maturity Model

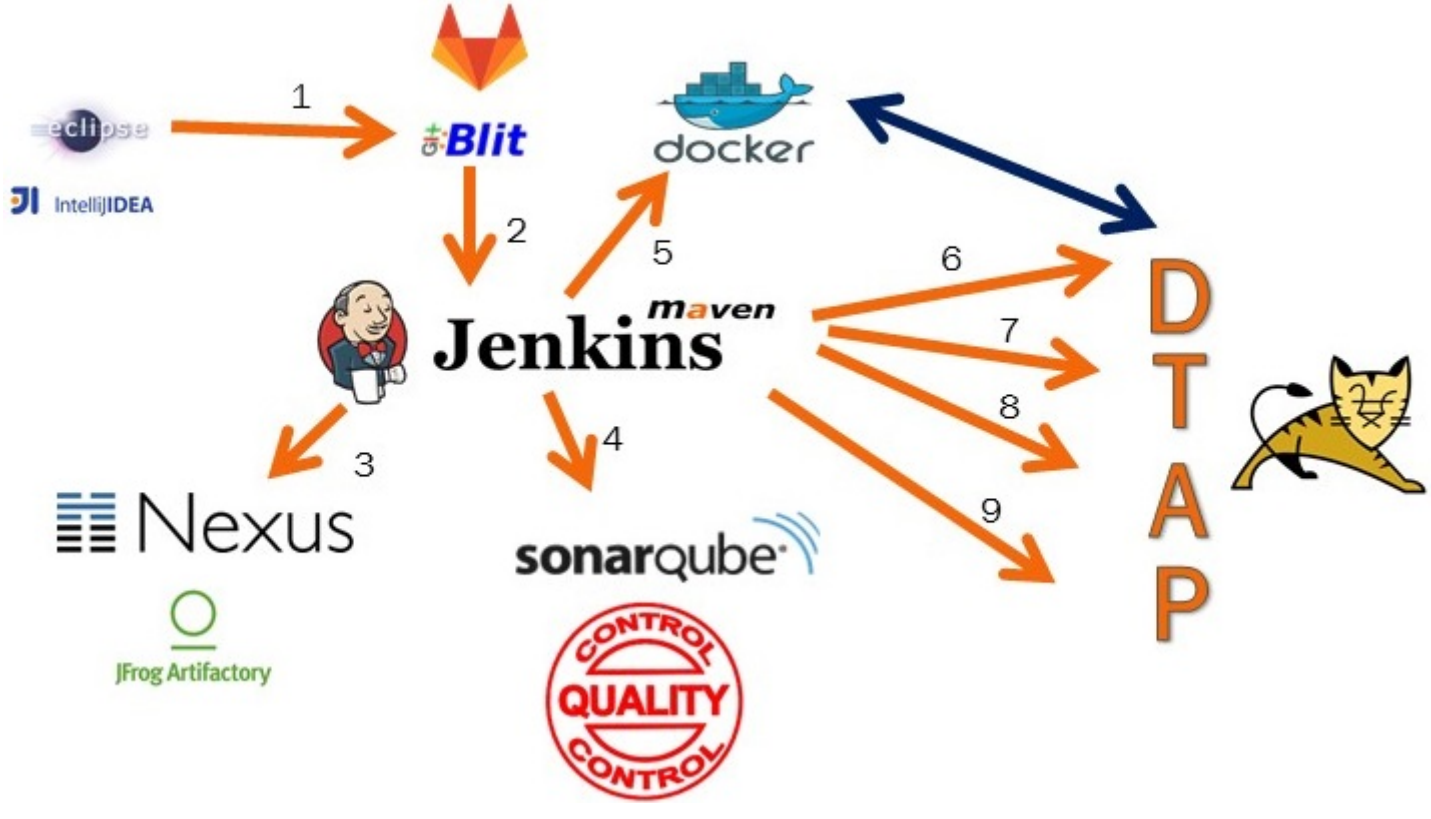
	Base	Beginner	Intermediate	Advanced	Expert
Culture & Organization	• Prioritized work • Defined and documented process • Frequent commits	• One backlog per team • Share the pain • Stable teams • Adopt basic Agile methods • Remove boundary dev & test	• Extended team collaboration • Component ownership • Act on metrics • Remove boundary dev & ops • Common process for all changes • Decentralize decissions	• Dedicated tools team • Team responsible all the way to prod • Deploy disconnected from Release • Continuous improvement (Kaizen)	• Cross functional teams • No rollbacks (always roll forward)
Design & Architecture	• Consolidated platform & technology	• Organize system into modules • API management • Library management • Version control DB changes	• No (or minimal) branching • Branch by abstraction • Configuration as code • Feature hiding • Making components out of modules	• Full component based architecture • Push business metrics	• Infrastructure as code
Build & Deploy	• Versioned code base • Scripted builds • Basic scheduled builds (CI) • Dedicated build server • Documented manual deploy • Some deployment scripts exsists	• Polling builds • Builds are stored • Manual tag & versioning • First step towards standardized deploys	• Auto triggered build (commit hooks) • Automated tag & versioning • Build once deploy anywhere • Automated bulk of DB changes • Basic pipeline with deploy to prod • Scripted config changes (e.g. app server) • Standard process for all environments	• Zero downtime deploys • Multiple build machines • Full automatic DB deploys	• Build bakery • Zero touch continuous deployments
Test & Verification	• Automatic unit tests • Separate test environment	• Automatic integration tests	• Automatic component tests (isolated) • Some automatic acceptance tests	• Full automatic acceptance tests • Automatic performance tests • Automatic security tests • Risk based manual testing	• Verify expected business value
Information & Reporting	• Baseline process metrics • Manual reporting	• Measure the process • Static code analysis • Scheduled quality reports	• Common information model • Traceability built into pipeline • Report history is available	• Graphing as a service • Dynamic test coverage analysis • Report trend analysis	• Dynamic graphing and dashboards • Cross silo analysis

Deployment pipeline



Deployment pipeline

Deployment pipeline



The example pipeline

- Commit and push changes to Git
- Git hook triggers Jenkins
- Build code
- SonarQube incl PiTest
- OWASP dependency check
- Store artifact(s) in Nexus
- Create a Docker image
- Store the Docker image in the registry
- Stop the old Docker container
- Start the new Docker container
- Gatling performance test

Versioning

- Used to link artifacts and sourcecode
- Maven release plugin is often used
- Triggers Git build -> endless loop
- 'Bug' in Jenkins Pipeline

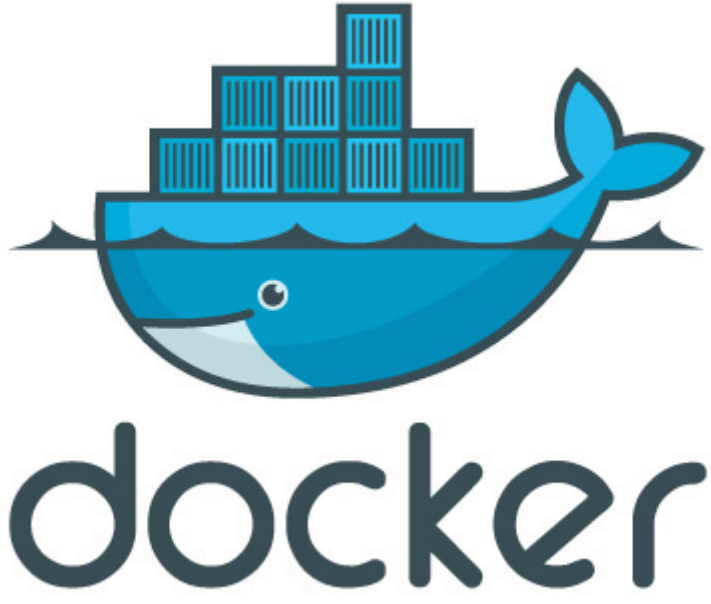
Forward versioning

- Don't change the version in the sourcecode
- Use the commit hash in the artifacts
  - JAR/WAR/EAR
  - Docker images
- Enables link between artifacts and sourcecode

How to implement continuous delivery successful

- Ensure that the code quality is high enough
- Run automated tests/verifications when deploying to production
- Use ATDD or create another form of functional tests
- Create an automated deployment pipeline
- Create the right culture and mindset
- Done means that the feature can be used on production by customers
- Continuously improve!
- Implement DevOps

Docker


docker

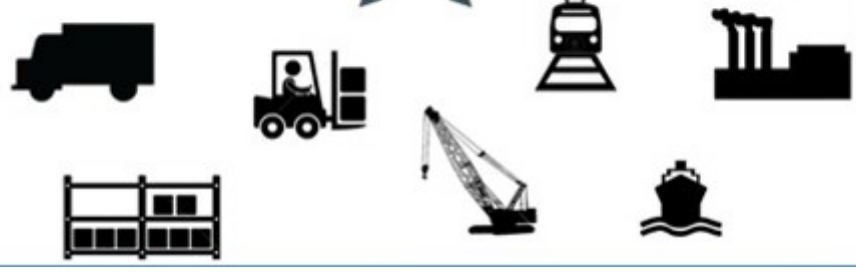
Why Docker?

Cargo Transport Pre-1960

Multiplicity of Goods

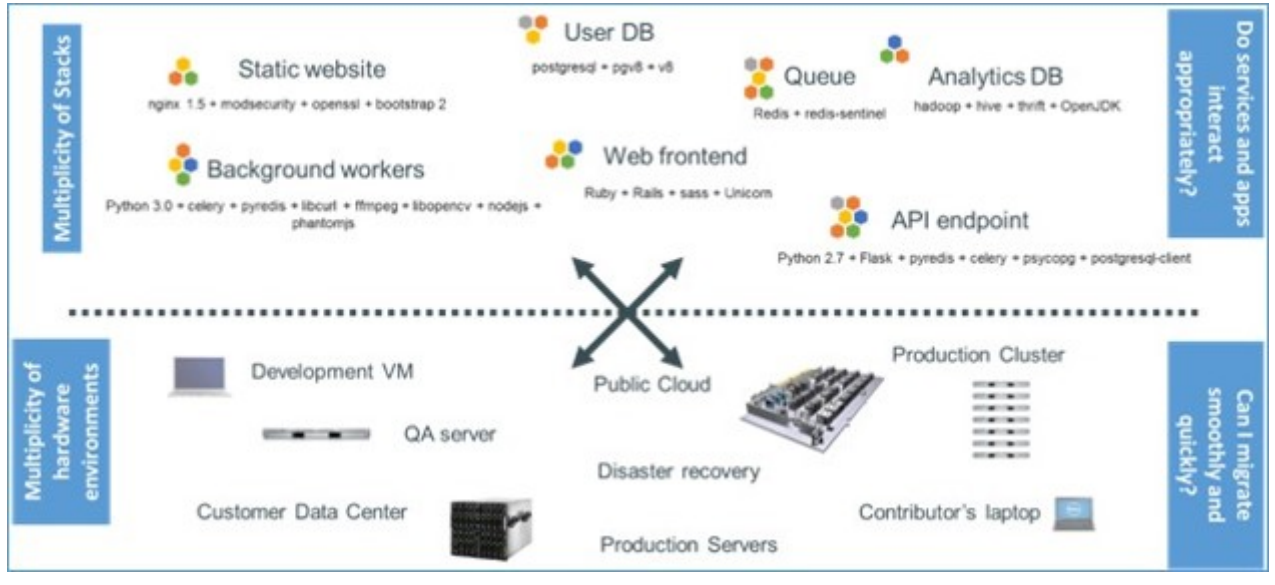
Do I worry about how goods interact (e.g. coffee beans next to spices)

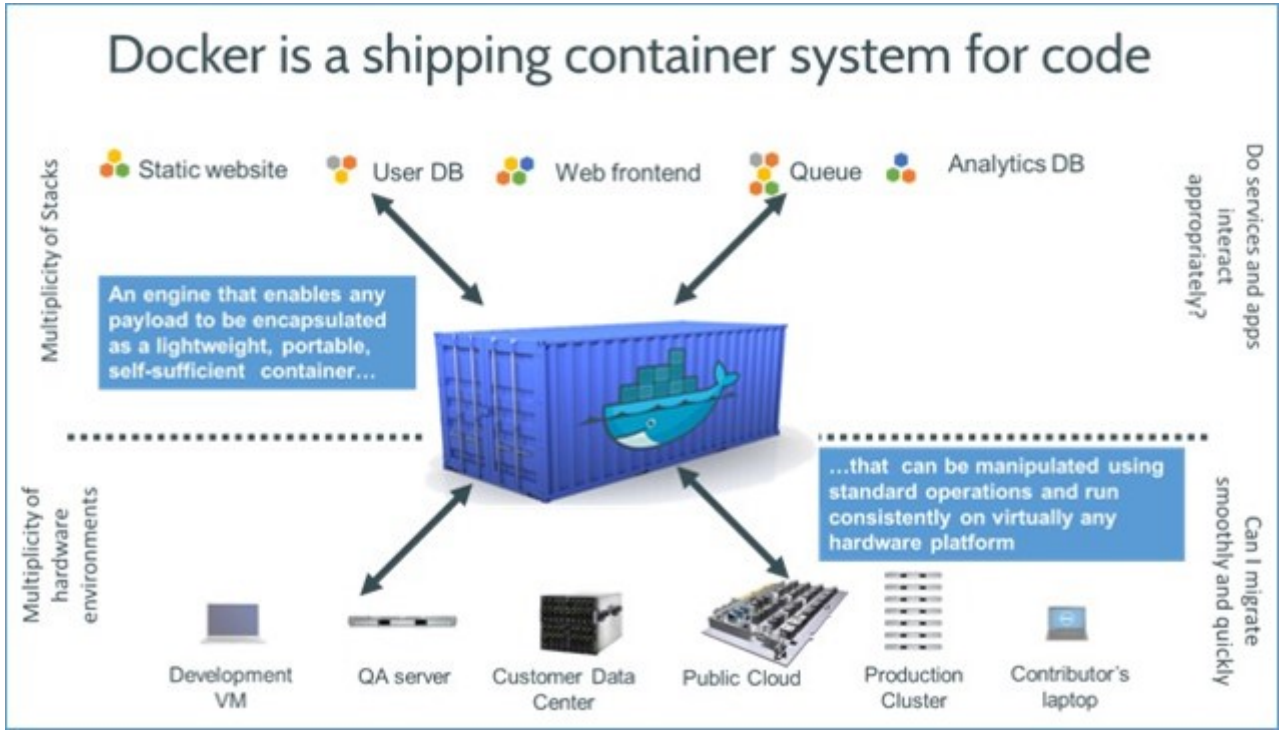
Multipillicity of methods for transporting/storing



Can I transport quickly and smoothly (e.g. from boat to train to truck)


Multiplicity of Goods
A standard container that is loaded with virtually any goods, and stays sealed until it reaches final delivery.
Do I worry about how goods interact (e.g. coffee beans next to spices)
...in between, can be loaded and unloaded, stacked, transported efficiently over long distances, and transferred from one mode of transport to another
Multiplicity of methods for transporting/storing
Can I transport quickly and smoothly (e.g. from boat to train to truck)


Multiplicity of Stacks
Static website
nginx 1.5 + modsecurity + openssl + bootstrap 2
User DB
postgresql + pgv8 + v8
Queue
Redis + redis-sentinel
Analytics DB
hadoop + hive + thrift + OpenJDK
Background workers
Python 3.0 + celery + pyredis + libcurl + ffmpeg + libopencv + nodejs + phantomjs
Web frontend
Ruby + Rails + sass + Unicorn
API endpoint
Python 2.7 + Flask + pyredis + celery + psycopg + postgresql-client
Do services and apps interact appropriately?
Multiplicity of hardware environments
Development VM
QA server
Customer Data Center
Public Cloud
Disaster recovery
Production Servers
Production Cluster
Contributor's laptop
Can I migrate smoothly and quickly?


Docker is a shipping container system for code
Multiplicity of Stacks
Static website
User DB
Web frontend
Queue
Analytics DB
Do services and apps interact appropriately?
An engine that enables any payload to be encapsulated as a lightweight, portable, self-sufficient container…
…that can be manipulated using standard operations and run consistently on virtually any hardware platform
Multiplicity of hardware environments
Can I migrate smoothly and quickly
Development VM
QA server
Customer Data Center
Public Cloud
Production Cluster
Contributor's laptop

Why Docker?

- To enable continuous delivery
- Quickly provision environments
- Run the same software local and in the cloud

Compatibility

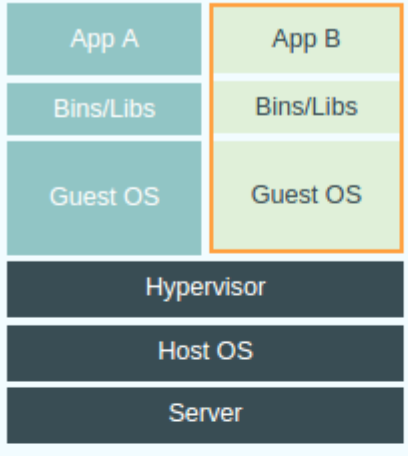
Run the same container everywhere

- Laptop
- Server
- Cloud

Except on other hardware architectures such as ARM

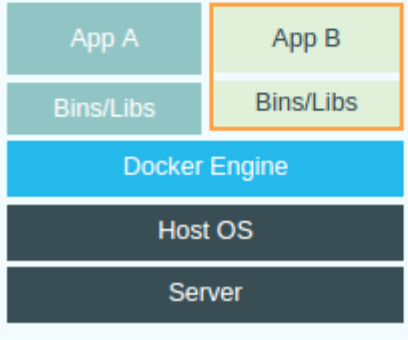
- Raspberry Pi

Docker versus virtual machines



Virtual Machines

Each virtualized application includes not only the application - which may be only 10s of MB - and the necessary binaries and libraries, but also an entire guest operating system - which may weigh 10s of GB.



Docker

The Docker Engine container comprises just the application and its dependencies. It runs as an isolated process in userspace on the host operating system, sharing the kernel with other containers. Thus, it enjoys the resource isolation and allocation benefits of VMs but is much more portable and efficient.

Docker versus virtual machines

- Disk space efficiency
- Memory efficiency
- Speed
- Compatibility (run anywhere)
- Isolation
- Versioning
- Internet of Things (Raspberry Pi etc.)

Terminology

- Image
  - Not running
  - Comparable to classes in Java
- Container
  - Running instance of an image
  - Comparable to objects in Java

My first Docker container

```
docker run -i -t ubuntu /bin/bash
```

Docker usecases

- DTAP environments
- Build environment
- Development including IDE
- Tests and other parts of continuous delivery
- Run the same container locally and on production
- Run applications secure and throw the container away after your done
- Also for GUI applications
- Basically for any abstraction

Dockerfiles directory structure

- Main directory
  - BuildAndRunScript.sh
  - GeneralBase
    - Dockerfile
  - SonarQube
    - Dockerfile

Dockerfile GeneralBase

```
FROM ubuntu:saucy

RUN apt-get -y install software-properties-common
RUN add-apt-repository ppa:webupd8team/java
RUN apt-get update && apt-get -y upgrade
RUN echo "oracle-java7-installer shared/accepted-oracle-license-v1-1
     boolean true" | debconf-set-selections
RUN apt-get -y install oracle-java7-installer
ENV JAVA_HOME /usr/lib/jvm/java-7-oracle
```

Build

- Create the Dockerfile
- Build the images:

```
cd GeneralBase
docker.io build -t GeneralBase .
cd ..
```


INHERIT
AND

Dockerfile SonarQube

```
FROM GeneralBase

RUN apt-get install -y wget unzip
RUN wget http://dist.sonar.codehaus.org/sonarqube-4.2.zip
RUN unzip sonarqube-4.2.zip -d /opt
RUN rm sonarqube-4.2.zip

EXPOSE 9000
EXPOSE 9092
CMD ["/opt/sonarqube-4.2/bin/linux-x86-64/sonar.sh",
    "console", "/bin/bash"]
```

Build

- Create the Dockerfile
- Build the images:

```
cd SonarQube
docker.io build -t SonarQube .
```

Building base

- Building base image is optional
- Only build it if something in base changed

Run

- Start the container

```
docker.io run -p 9000:9000 –p 9092:9092 -d SonarQube
```

List all (in)active containers

```
# docker.io ps –a
CONTAINER ID: ecbecf77461b
CREATED: 32 minutes ago
STATUS: Up 32 minutes
PORTS: 0.0.0.0:9000->9000/tcp, 0.0.0.0:9092->9092/tcp
```

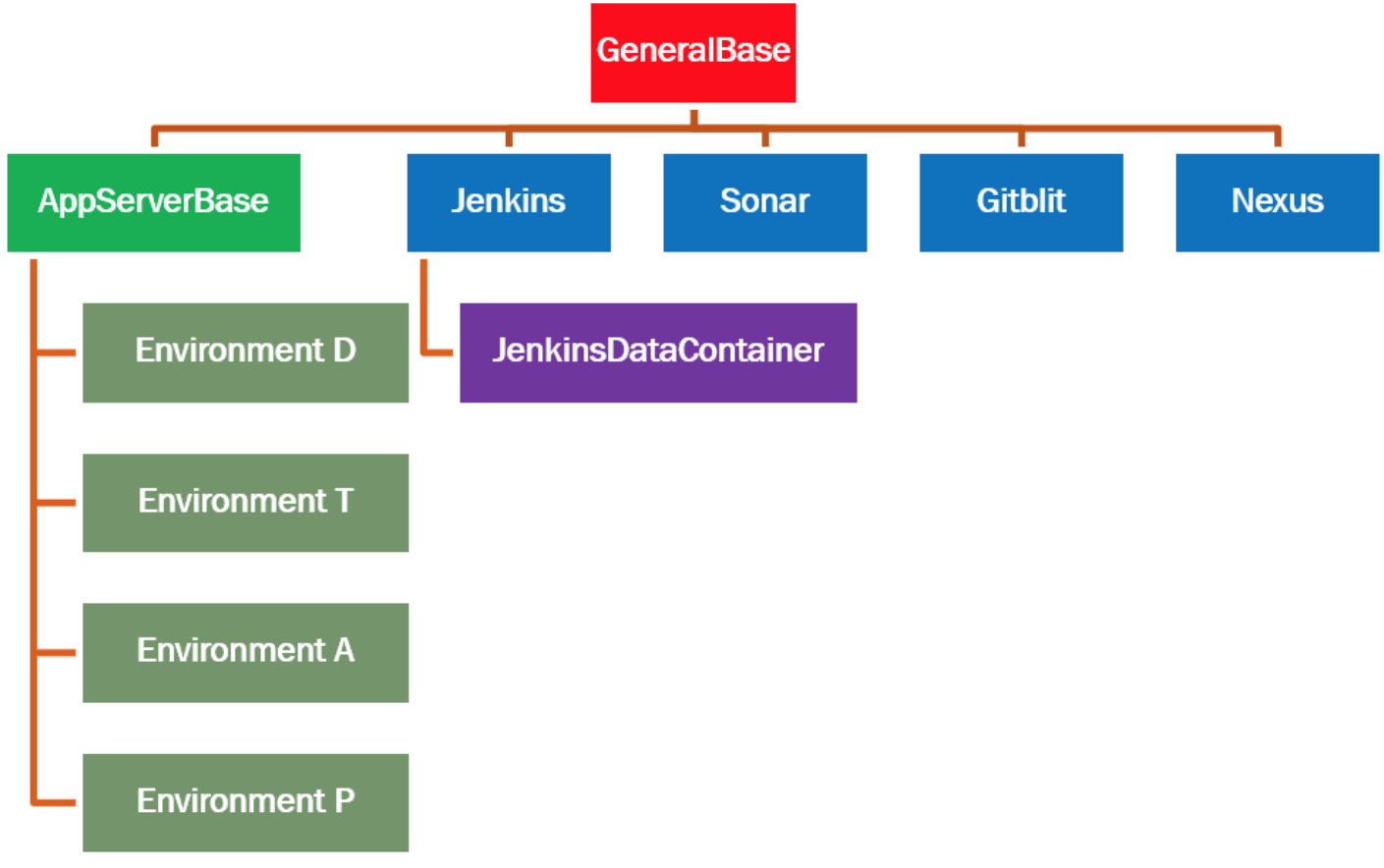
Controlling containers

- Start / stop / restart

```
docker [start/stop/restart] containerid
```

Follow SystemOut and SystemErr

```
docker logs -f containerid
```


GeneralBase
AppServerBase
Jenkins
Sonar
Gitblit
Nexus
Environment D
JenkinsDataContainer
Environment T
Environment A
Environment P


reception
Plastic Bottles Only
Please remove the lids
NO
cans & foil
BROWN
GLASS
BOTTLES
AND JARS
ONLY
Newspapers, Magazines
& Junk Mail
NO
Cardboard
Yellow Pages
Thomson Directory
Plastic Bags
Packing
Wrappers
Other Bits & Bobs
Clear Glass
Bottles & Jars
NO
Crockery / Plate Glass
Pyrex / Pots
Plastic Bags
Cardboard
Packaging
Refuse
recycle
Any Other Bits & Bobs
Green Glass
Bottles & Jars
NO
Crockery / Plate Glass
Pyrex / Pots
Plastic Bags
Cardboard
Packaging
Refuse
recycle
Any Other Bits & Bobs

Data storage

- In the same container as the application
- In a data container / data volume
- On the host

Data volumes

- Dockerfile

```
ENV JENKINS_HOME /var/JenkinsData
```

- Docker commands

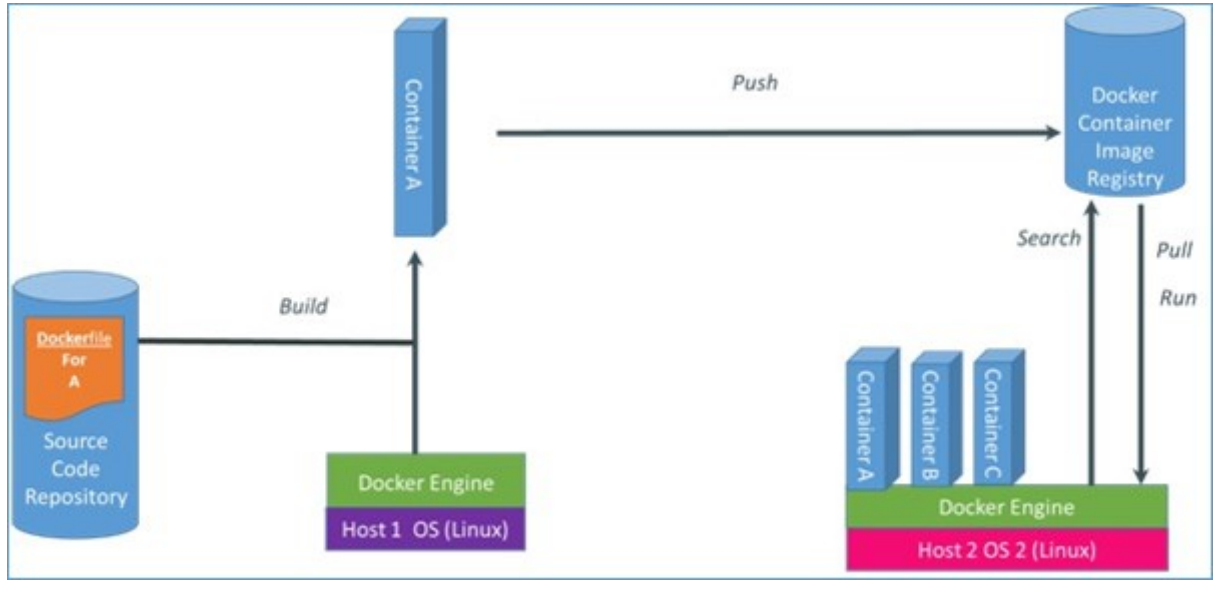
```
docker.io run -v /var/JenkinsData –name
   JenkinsDataContainer ubuntu:saucy true

docker.io run -p 8080:8080 --volumes-from
   JenkinsDataContainer -d Jenkins
```

Diskspace

```
# docker.io images --tree
└─ 179.9 MB Tags: ubuntu:saucy
          └─253.6 MB
             └─741.8 MB Tags: GeneralBase:latest
                    └─763.6 MB Tags: AppServerBase:latest
              …
                     ├─763.6 MB Tags: EnvironmentP:latest
              └─865.6 MB Tags: Nexus:latest
                    └─808.3 MB Tags: Gitblit:latest
                    └─901.5 MB Tags: Sonar:latest
                    └─805.4 MB Tags: Jenkins:latest
```

Execution time (download, build, start)

```
real    4m11.729s
user    0m3.329s
 sys    0m10.054s
```


Container A
Push
Docker
Container
Image
Registry
Search
Pull
Run
Build
Dockerfile
For
A
Source
Code
Repository
Container A
Container B
Container C
Docker Engine
Host 1 OS (Linux)
Docker Engine
Host 2 OS 2 (Linux)

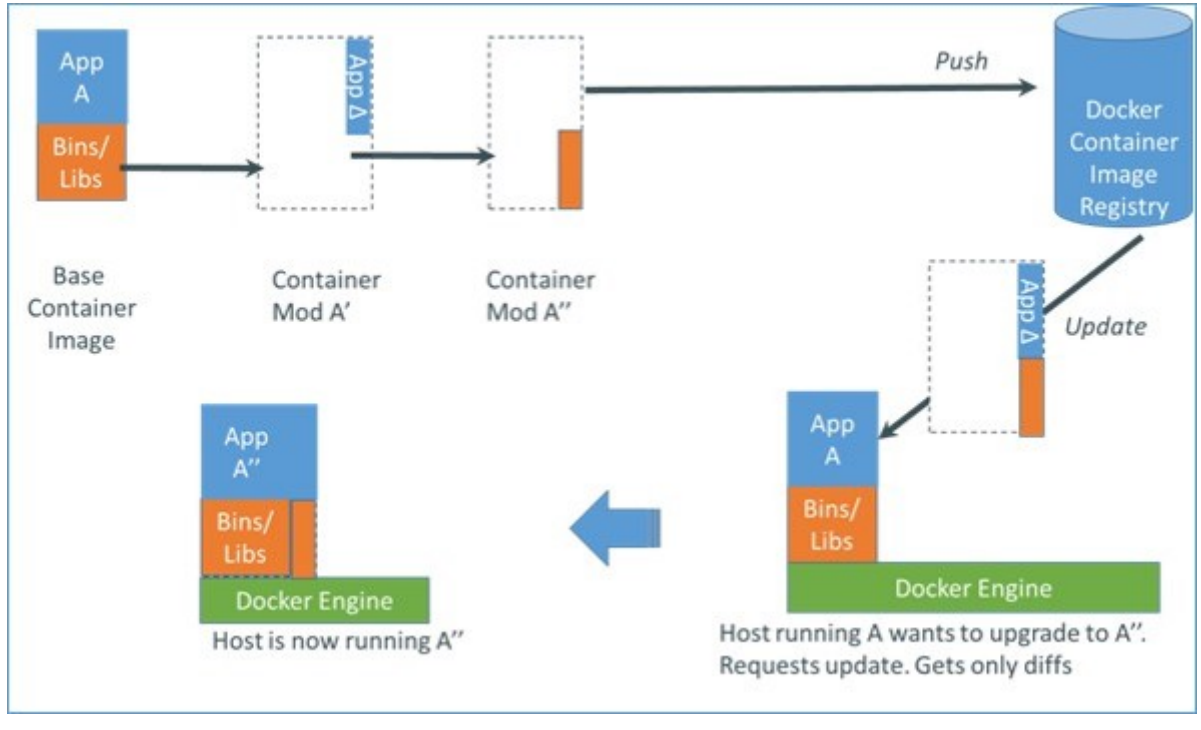
Public Docker registry

- Contains lots of images
- Possibility to store images
- Companies don't want to store their images in a public repository

Local Docker registry

- Creating a local Docker registry

```
docker run -p 5000:5000 registry
```


App A
Bins/ Libs
Base Container Image
App Δ
Container Mod A'
Container Mod A''
Push
Docker Container Image Registry
Update
App A''
Bins/ Libs
Docker Engine
Host is now running A''
App A
Bins/ Libs
Docker Engine
Host running A wants to upgrade to A''. Requests update. Gets only diffs

Docker client 1 (push)

- Modify image
- Commit

```
docker.io commit 064f
    192.168.56.31:5000/test-version-0.2
```

- New id -> **ff7e**
- Push

```
docker.io push
    192.168.56.31:5000/test-version-0.2
```

Docker client 2 (pull)

- Pull

```
docker.io pull 192.168.56.31:5000/
    test-version-0.2
```

- Run

```
docker.io run -i -t ff7e /bin/bash
```

Pull update only

```
docker images -tree
└─153b 194.2 MB test-version-0.1:latest

docker pull 192.168.56.31:5000/test-version-0.2
ff7e: Download complete
153b: Download complete

docker images -tree
└─153b 194.2 MB test-version-0.1:latest
  └─ff7e 194.2 MB test-version-0.2:latest
```

Sharing images

- Base image
  - Team Frodo image
  - Team Bilbo image

Application/environments per team

- Team Frodo
  - Application Gimli
  - Application **Elrond**
- Team Bilbo
  - Application **Elrond**
- Team Galadriel
  - Application Radagast

Where to store the Elrond application?

- Base image pollutes the Galadriel image
- In the Frodo and Bilbo images results in duplication

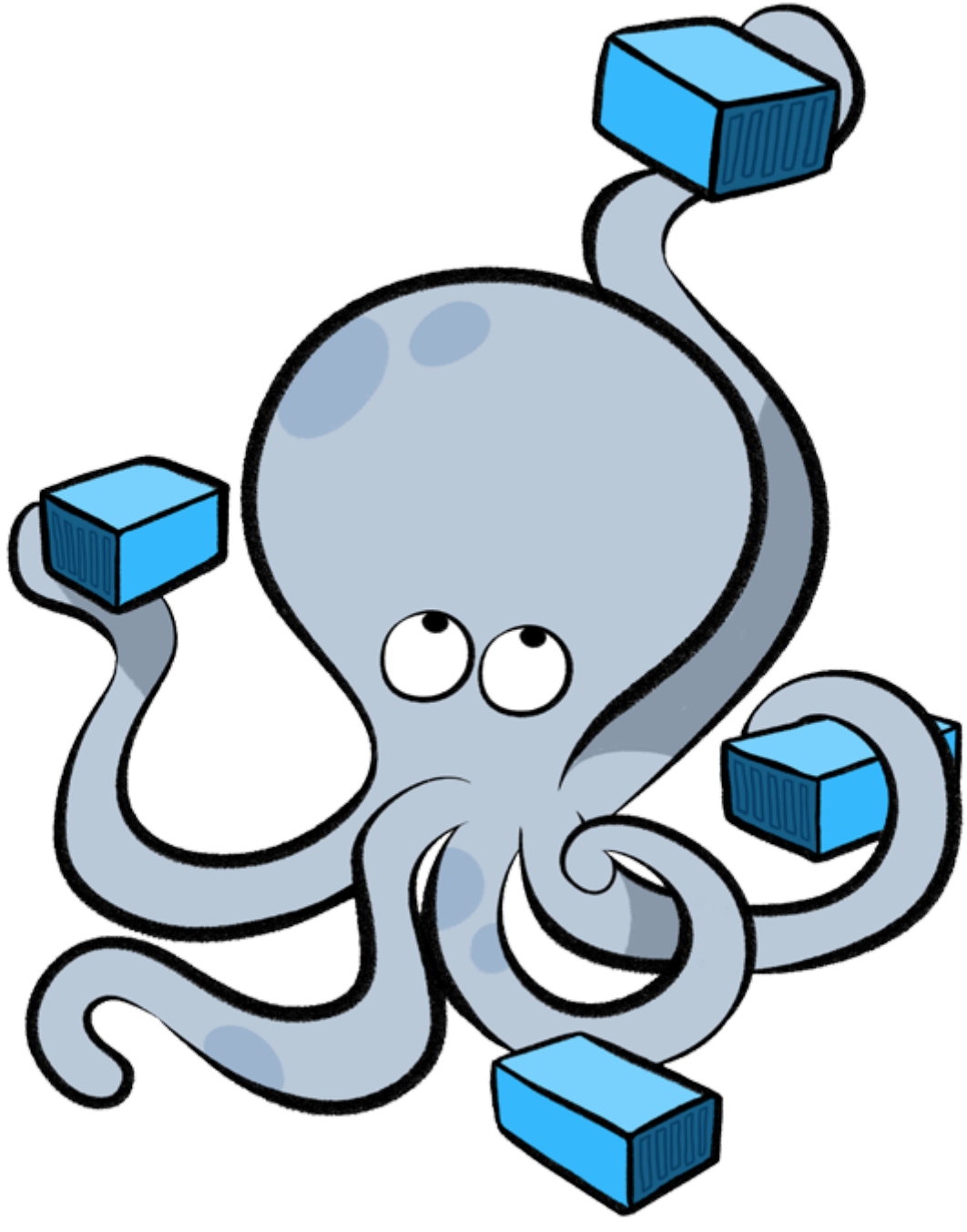
Composition

- Create one image for every application
- At runtime it looks like one environment

Images

- Tomcat image
  - Gimli image
  - Elrond image
  - Radagast image
- Fedora image
  - Development environment

Docker compose



Docker compose

- Define and run multi container Docker applications
- Using a Compose file
- Link containers
- ‘docker-compose up’ to start the containers

Directory structure

- TomcatGimli
  - DockerFile
- TomcatElrond
  - DockerFile
- TomcatRadagast
  - Dockerfile
- DevEnv
  - Dockerfile
- docker-compose.yml

```
tomcatgimli:
 build: TomcatGimli

tomcatelrond:
 build: TomcatElrond

tomcatradagast:
 build: TomcatRadagast

developmentenvironment:
 build:  DevEnv
 ports:
  - "3389:3389"
 links:
  - tomcatgimli:gimli # Makes gimli available on http://gimli:8080
  - tomcatelrond:elrond
  - tomcatradagast:radagast
```

Environment specific configuration

- Use docker-compose.override.yml
- Put all the configuration in one container
- Create small containers with configuration per environment that inherit the application container
- Commandline arguments

Orchestration tools

- Kubernetes
- Docker Swarm
- Mesos
- …

Orchestration tools

- Scalability
- Failover
- Rollouts and rollbacks
- Self healing
- Service discovery
- Load balancing
- …

Conclusion

- Use a (private) Docker registry
- Separate environment settings
- Use a tool like Jenkins to manage containers and images
- Do not add things like OpenSSH

Conclusion

- Think about topics like security, monitoring and logging
- Inherit and/or compose
- Separate concerns in separate containers

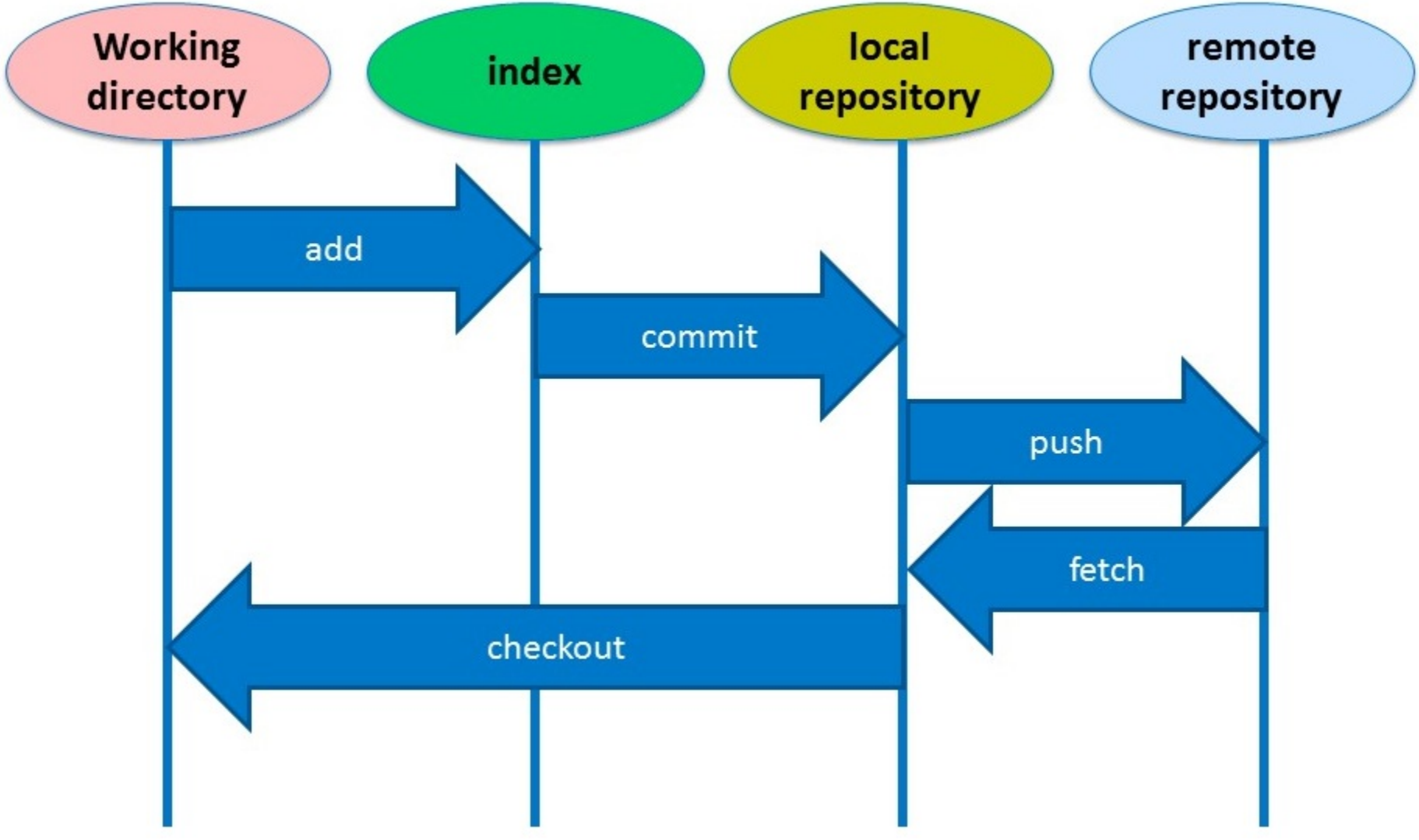
Git

Why use version control?

- History of changes
- Possibility to go back
- Single place for all relevant data
- Ability to work with multiple people on the same codebase
- Create different versions of the application

Git benefits

- Everything is local
- Cheap local branching
- Git is fast
- Git is small



Git terminology

- Pull
  - Update contents of a working copy
  - Fetch (all branches) and merge (current branch)
- Commit
  - Storing a set of changes from the index into the local repository
- Push
  - Storing the local repository in the remote

Git best practices

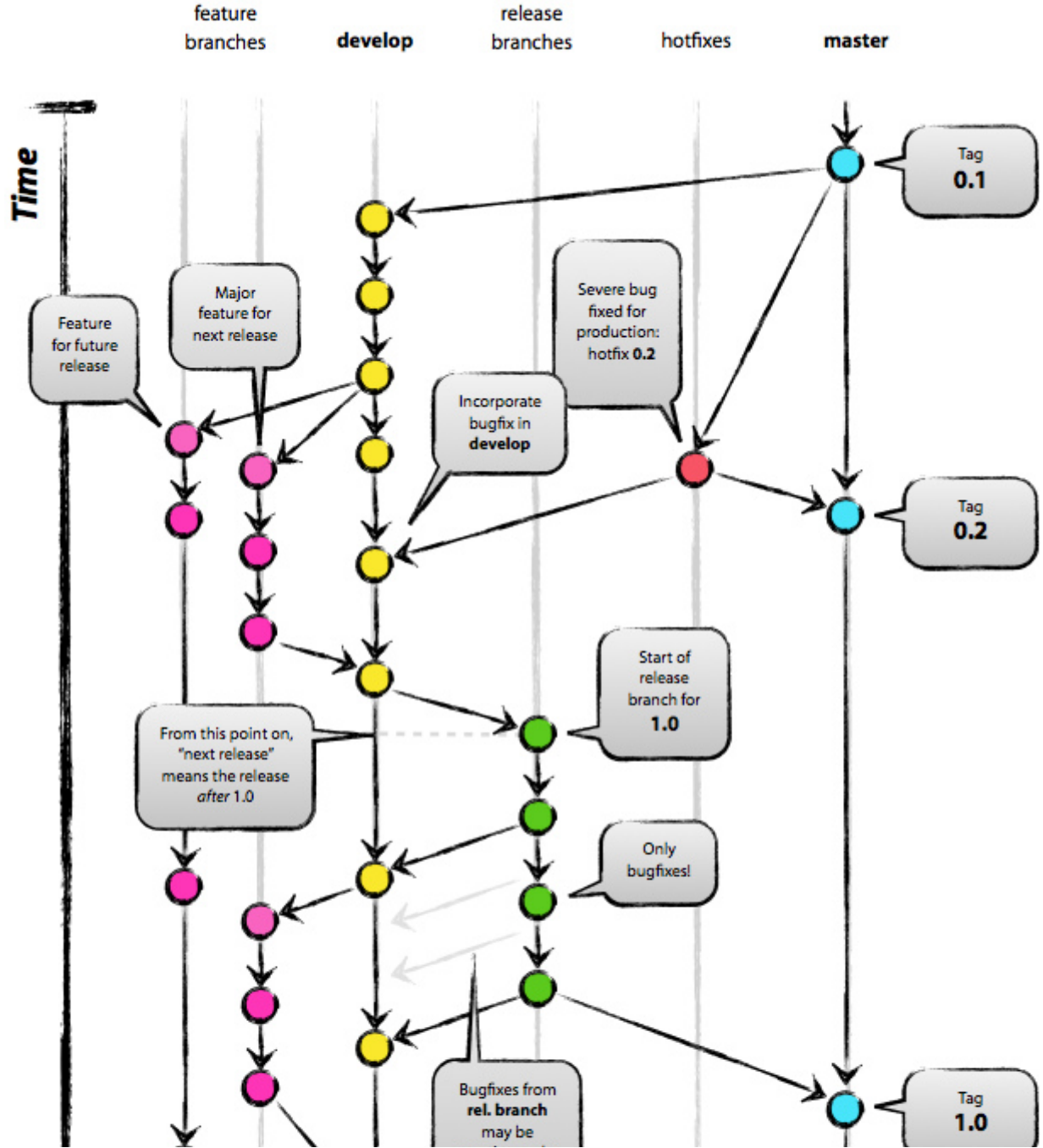
- Work at one change at a time
- Commit regularly
  - Try to break up large changes into smaller pieces
- Add the issue code to the comment
  - This allows Jira to track files that are changed in a issue
  - E.g. “PROJECT-1234 - Fixed some minor bug”

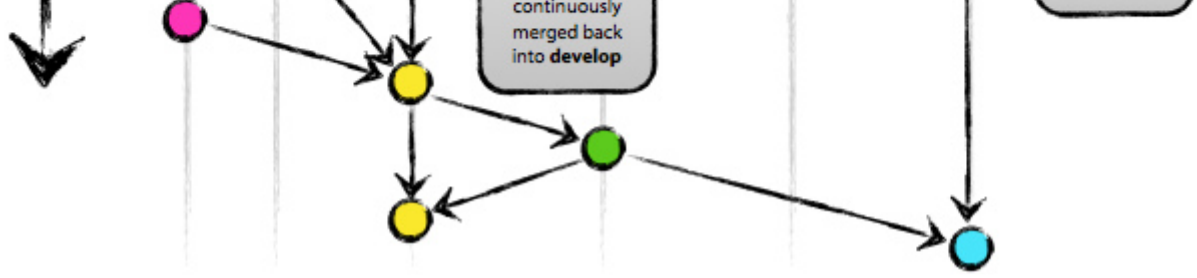
Git server

- There are many options
  - GitHub
  - GitLab
  - BitBucket Server
- Features are comparable
  - Web frontend
  - Support for pull requests
- Difference mainly lies in integration with tools like Jira

Client tooling

- Command line
- Atlassian Sourcetree
- Eclipse/IntelliJ integration


feature branches
develop
release branches
hotfixes
master
Time
Tag 0.1
Feature for future release
Major feature for next release
Severe bug fixed for production: hotfix 0.2
Incorporate bugfix in develop
Tag 0.2
Start of release branch for 1.0
From this point on, "next release" means the release after 1.0
Only bugfixes!
Bugfixes from rel. branch may be
Tag 1.0


continuously
merged back
into **develop**

Using Git with continuous delivery

- Git encourages to use branches
- Git flow requires quite some maintenance
  - Especially if you work with multiple people on the same codebase
  - Branches live for a while making it harder to merge them back

Using Git with continuous delivery

####Using branches is a bad practice in continuous delivery

- Tests / quality control is (mostly) only performed on one branch
- Deployment to environments is (mostly) only performed on one branch
- Code only 'works' after it's successfully merged back to master and deployed

Git tips

####Try to avoid merge issues

- Structure the codebase properly
- Put different features in different files
- Don't work on the same part of the codebase with two or more developers

What you should store

- Source code
- Configuration

What you shouldn't store

- Binaries
  - Artifacts such as JAR's/WAR's
- Log files
- target directory
- Configuration specific for your machine

Build tools

Why?

Makes it easy to automate tasks such as:

- Dependency management (logging, security etc.)
- Compile the application
- Test the application
- Verify the code quality
- Create one or more artifacts

Some options

- Ant
- Maven
- Gradle

Quick comparison

- Ant and Maven use a XML syntax
- Gradle uses a Groovy DSL
- Maven forces you to work in a certain way
- Ant and Gradle give you more freedom

Maven basic example

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/xsd/maven-4.0.0.xsd">

    <modelVersion>4.0.0</modelVersion>
    <groupId>com.example</groupId>
    <artifactId>TestApp</artifactId>
    <version>1.1</version>
    <packaging>jar</packaging>
</project>
```

Maven dependencies

```
<project ...>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>[VERSION]</version>
        <scope>test</scope>
    </dependency>
</project>
```

Maven dependency scope

- Compile (default): packaged in the artifact
- Test: only used for tests, not packaged in the artifact
- Provided: expected to be provided by for instance the application server, not packaged in the artifact

More Maven

- The above is enough to get started
- But you can use Maven for various other things
- Some are shown in the following slides

Maven parent

- Create a hierarchy of projects
- Specify configuration such as dependencies and their versions in one place

```
<project ...>
  <parent>
      <groupId>com.example</groupId>
      <artifactId>ParentExample</artifactId>
      <version>1.0</version>
      <relativePath>../pom.xml</relativePath>
  </parent>
</project>
```

Configure artifact repositories

```
<project ...>
    <distributionManagement>
        <repository>
            <id>nexus</id>
            <url>http://[IP]:[PORT]/repository/maven-releases/</url>
        </repository>
    </distributionManagement>
</project>
```

Configure Maven settings.xml

```
<servers>
    <server>
        <id>nexus</id>
        <username>admin</username>
        <password>admin123</password>
    </server>
</servers>
```

Configure Git

```
<project ...>
    <scm>
        <connection>
            scm:git:http://root:wachtwoord@gitlab:80/root/testapp
        </connection>
        <url>http://gitlab:80/root/testapp</url>
    </scm>
</project>
```

Plugins

- To extend the Maven functionality
- Example to create a FAT Jar with Spring Boot

```
<project ...>
  <build>
      <plugins>
          <plugin>
              <groupId>
                  org.springframework.boot
              </groupId>
              <artifactId>
                  spring-boot-maven-plugin
              </artifactId>
          </plugin>
      </plugins>
  </build>
</project>
```

Artifact repositories

Examples

- Nexus
- Artifactory

Still necessary with a Docker registry?

- Not really

Jenkins

Code Quality

Topics

- Security
- Performance
- Automated SonarQube
- Manual codereviews

Testing

Lots of possibilities

- Unit tests
- Integration tests
- Functional (ATDD) tests
- Manual tests
- Security tests
- Performance tests

What about test quality?

- PiTest

Testing on production

Why?

- Other environments, even acceptance/disaster recovery, (might) vary from production
- Software only works when deployed in production
- Done means customers can use it in a production environment

--- How?

- Release new feature to subset of customers
- Executing test scenario's with test customers
- Don't forget to monitor what's going on

Chaos Engineering

- Made popular by Netflix
- Controlled experiments on production
- Shutting down systems, increasing load etc.
- All about improving the systems, NOT about breaking them

Conclusion

Monitoring

Security

Issue tracking

Documentation

Conclusion

Interesting books

- Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation
- Chaos Engineering: http://www.oreilly.com/webops-perf/free/chaos-engineering.csp
- The Art of Application Performance Testing

Interesting websites

- https://martinfowler.com/delivery.html

Conclusion

Questions

You can always contact me or Info Support.