

Software Engineering and Architecture

Lecture 3: Continuous Delivery (1)

Olivier Liechti

olivier.liechti@heig-vd.ch



MASTER OF SCIENCE
IN ENGINEERING

Today.

Continuous Delivery (1)

15h00 - 15h45

OO Reengineering Patterns (7 x 10')

16h00 - 17h10

First Jenkins demo

17h10 - 17h20

Week	Theory	OO Reengineering	Practice
#1	Agile, Scrum		Intro to Docker
#2	Software evolution	Introduction	Specify and implement a micro-service
#3	Continuous delivery (1)	<i>Setting Directions</i>	Intro to Jenkins & Travis
#4	Continuous delivery (2)	<i>First Contact</i>	Our first build pipeline
#5	Agile testing (1)	<i>Initial Understanding</i>	Intro to Cucumber
#6	Agile testing (2)	<i>Detailed Model Capture</i>	Add tests to pipeline
#7	Agile metrics	<i>Tests: Your Life Insurance!</i>	Add Sonar to pipeline
#8	Continuous improvement	<i>Migration Strategies I</i>	Add non-functional tests
#9	Wrap-up	<i>Migration Strategies II</i>	

The Business Need

INTERNATIONAL BESTSELLING SENSATION

THE LEAN STARTUP



How Constant **Innovation**
Creates Radically
Successful Businesses

ERIC RIES

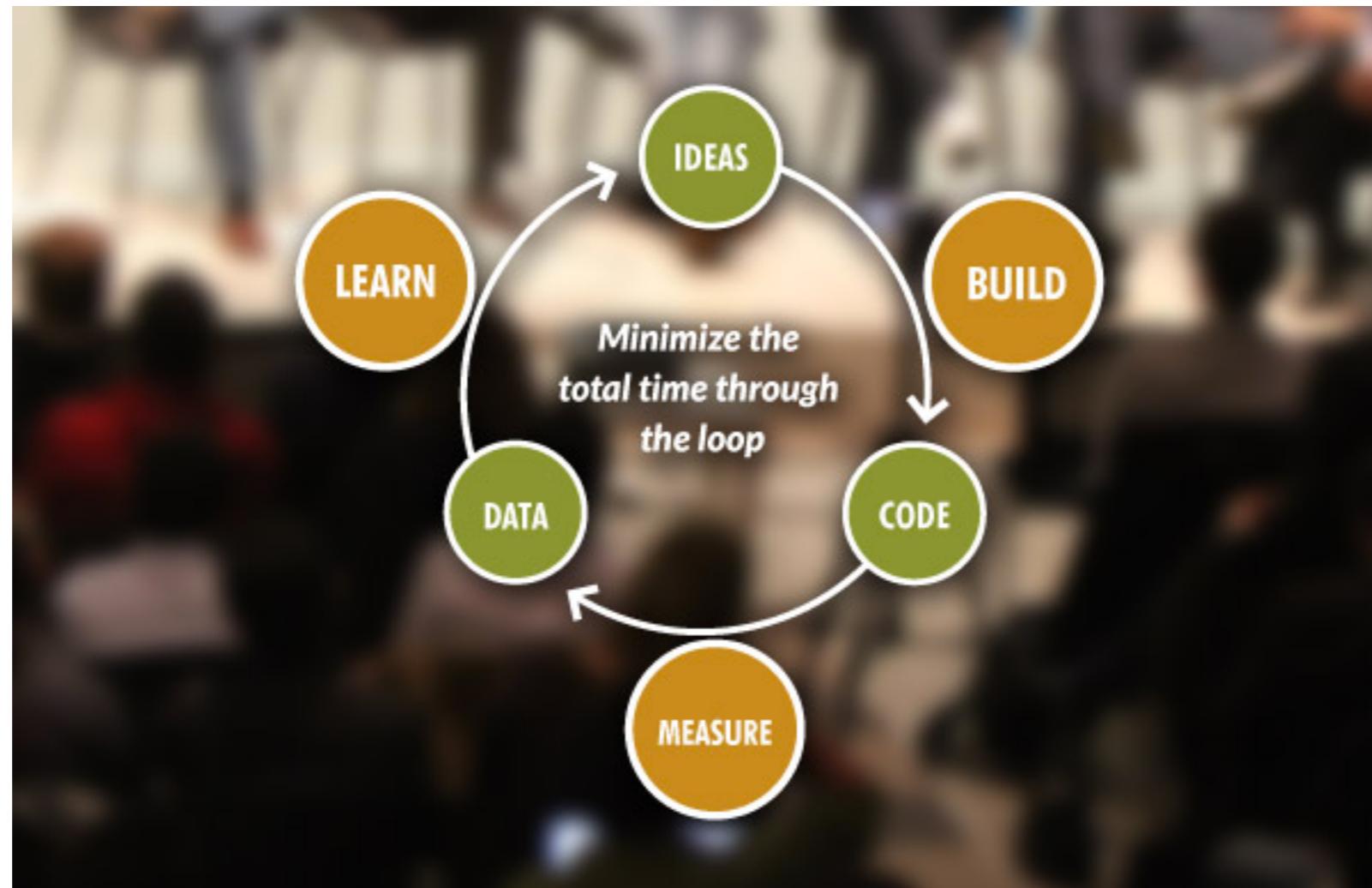
'Mandatory reading for entrepreneurs' **Dan Heath**



MASTER OF SCIENCE
IN ENGINEERING

How to build the right product?

Apply the scientific method to business ideas



<http://theleanstartup.com/principles>

Problem statement

How do we ensure that the business can quickly run a lot of experiments, to assess the value of features or to compare alternatives?

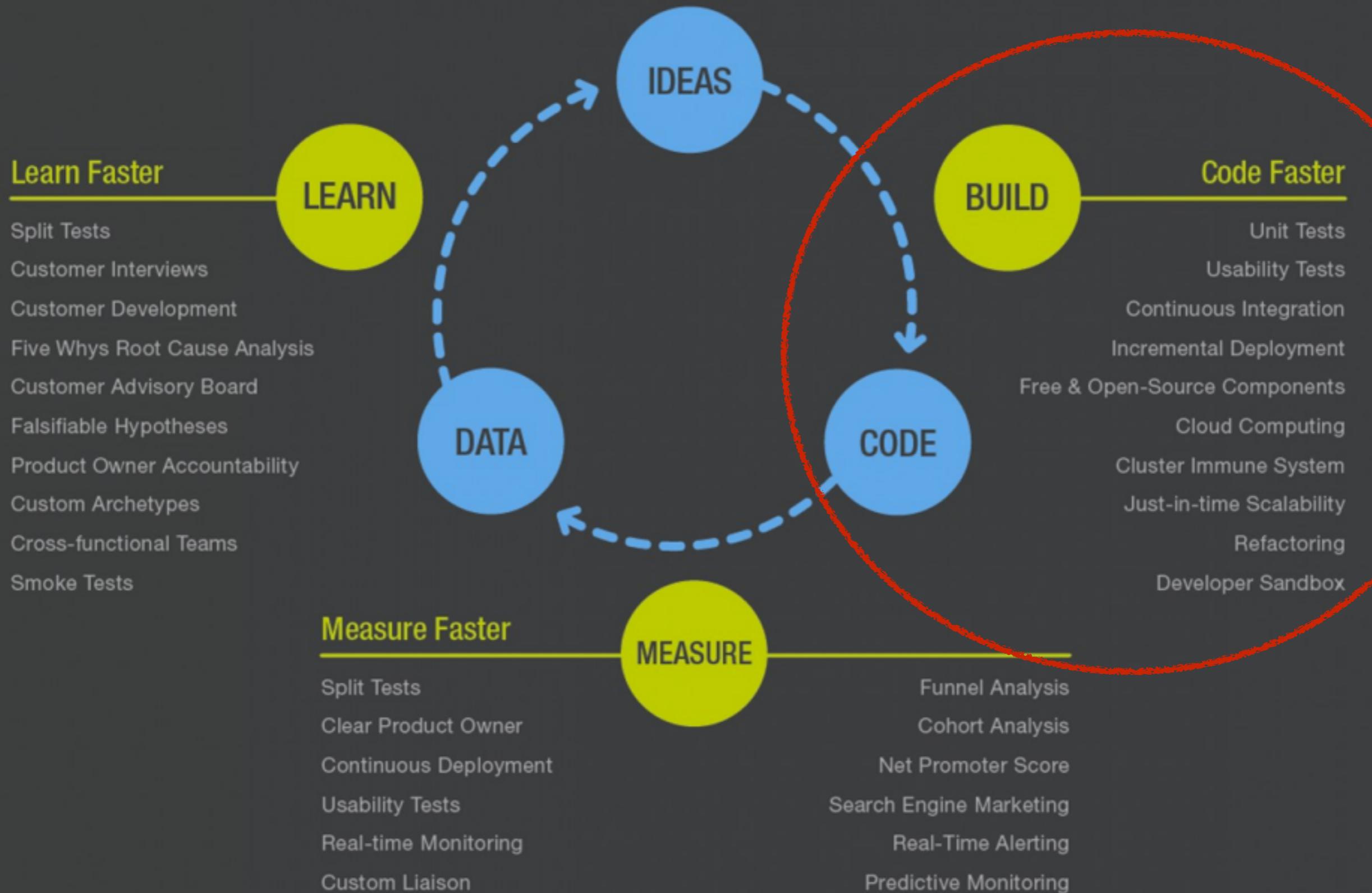
“If we added a feature to receive Telegram notifications from our online shop, how many customers would use it?”

How much time do we need to 1) implement a minimal version of this feature, 2) to release it to a portion of the audience and 3) to collect data?

THE LEAN STARTUP

Created by Eric Ries - startuplessonslearned.blogspot.com

Designed by  KISSmetrics



The Problem with Silos



“Here is a USB key with the .war file that you need to deploy on Tomcat. I have written down the configuration steps on this sheet of paper...”



“WTF... why do I get a database connection error when I deploy this damn application in the Glassfish cluster...???”

Worried
Ops guy



Worried
Dev guy

*"We have deployed the new release... are we **sure** that we are ok?
By the way, anyone has a clue about what is in the release?"*

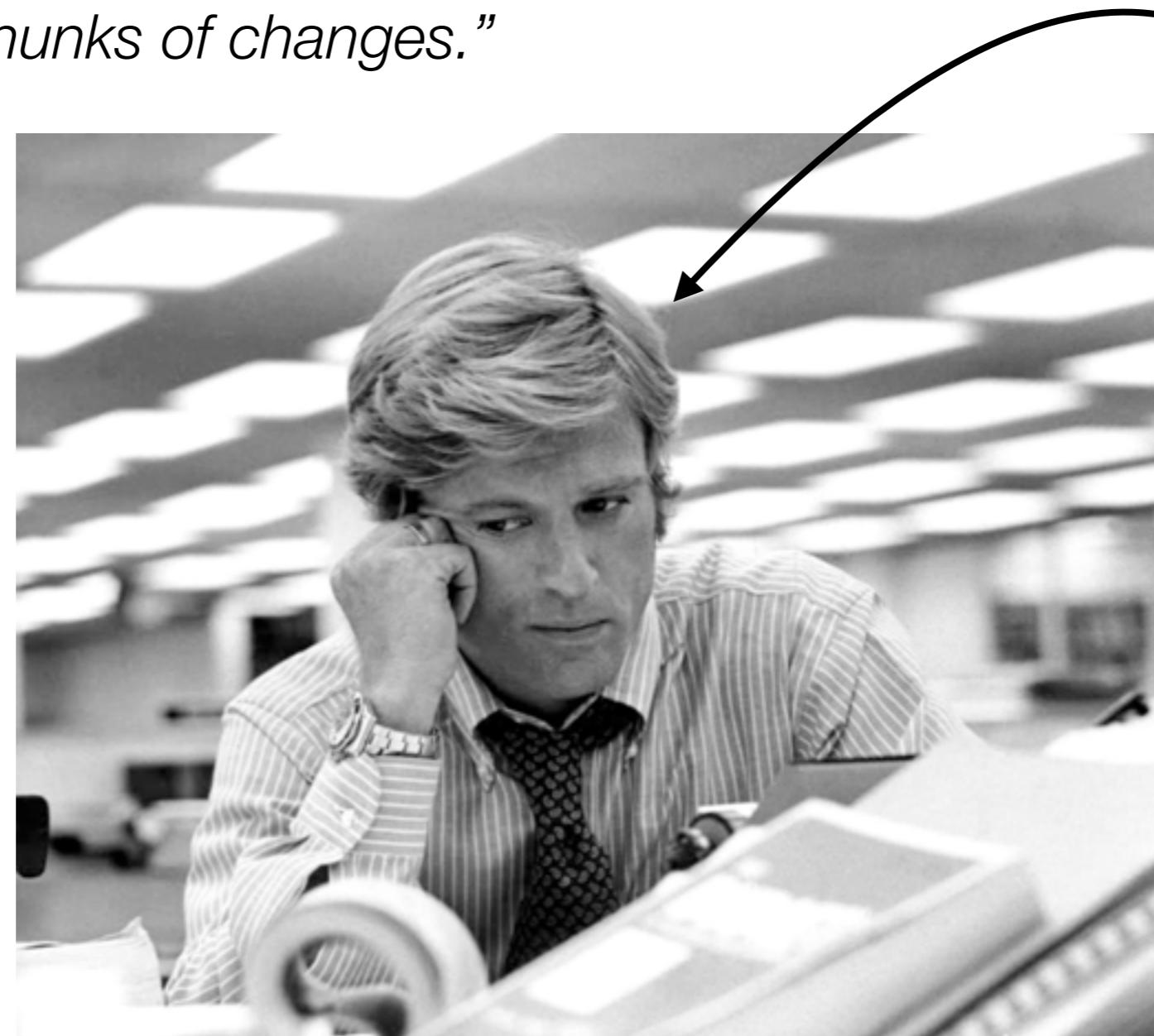


Robert Redford

*"I hate these production rollouts... it's always so **stressful**.
What could we do to avoid bad surprises and not be so
afraid to touch the prod...?"*

“Releasing **small chunks** of changes is less scary than releasing big chunks of changes.”

“Something that you do **often** is less scary than something you do rarely.”



Agile
Robert Redford

“If something is **automated**, there is now worry that I can make a wrong manipulation.”

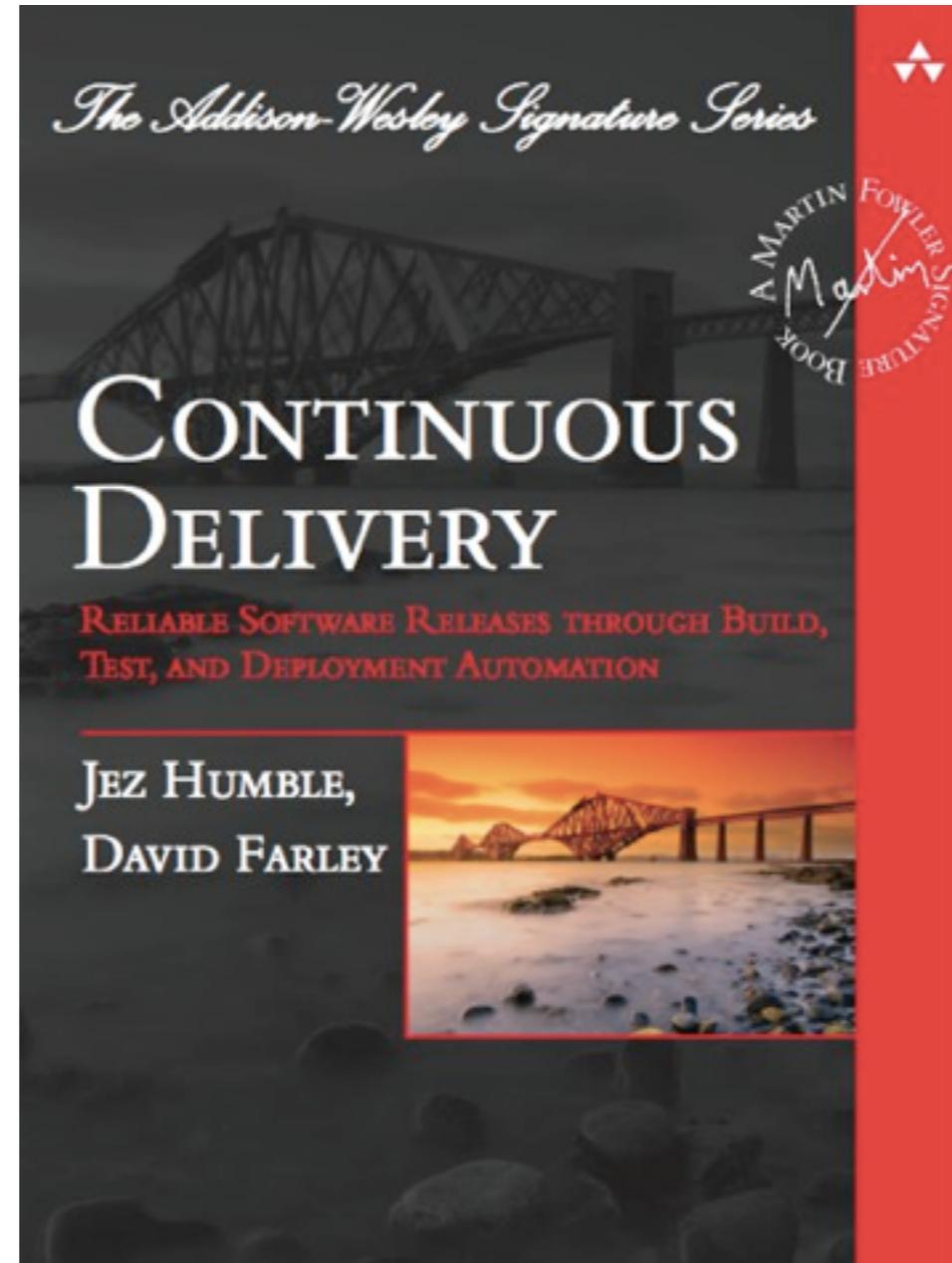
“If I have **evidence** that the system has been **fully tested**, at different levels, I feel secure.”



MASTER OF SCIENCE
IN ENGINEERING

The Solution

Continuous Delivery



The slides are based on this book (ISBN:)
Free chapter: <http://www.informit.com/articles/article.aspx?p=1621865>

Foreword, by Martin Fowler (1)

“Continuous Integration is just the first step.

Software that's been successfully integrated into a mainline code stream still isn't software that's out in production doing its job.

*Dave and Jez's book pick up the story from CI to **deal with that “last mile”** describing **how to build the deployment pipeline that turns integrated code into production software.**”*

Foreword, by Martin Fowler (2)

*“This kind of delivery thinking has long been a forgotten corner of software development, falling into **a hole between developers and operations teams.***

*So it’s no surprise that the techniques in this book rest upon bringing these teams together—a harbinger of the nascent but growing **DevOps** movement.*

*This process also involves **testers**, as testing is a key element of ensuring error-free releases.”*

The Problem of Delivering Software

- How do we transform an **idea** into **something usable by end-users** as **quickly** as possible?
- We need an efficient **software development process**, but that is not enough.
- We need to cover the **complete delivery life-cycle**, until the release into the production environment.



Release Anti-Patterns

- **Anti-Pattern:** Deploying Software Manually
- **Anti-Pattern:** Deploying to a Production-like Environment Only after Development Is Complete
- **Anti-Pattern:** Manual Configuration Management of Production Environments

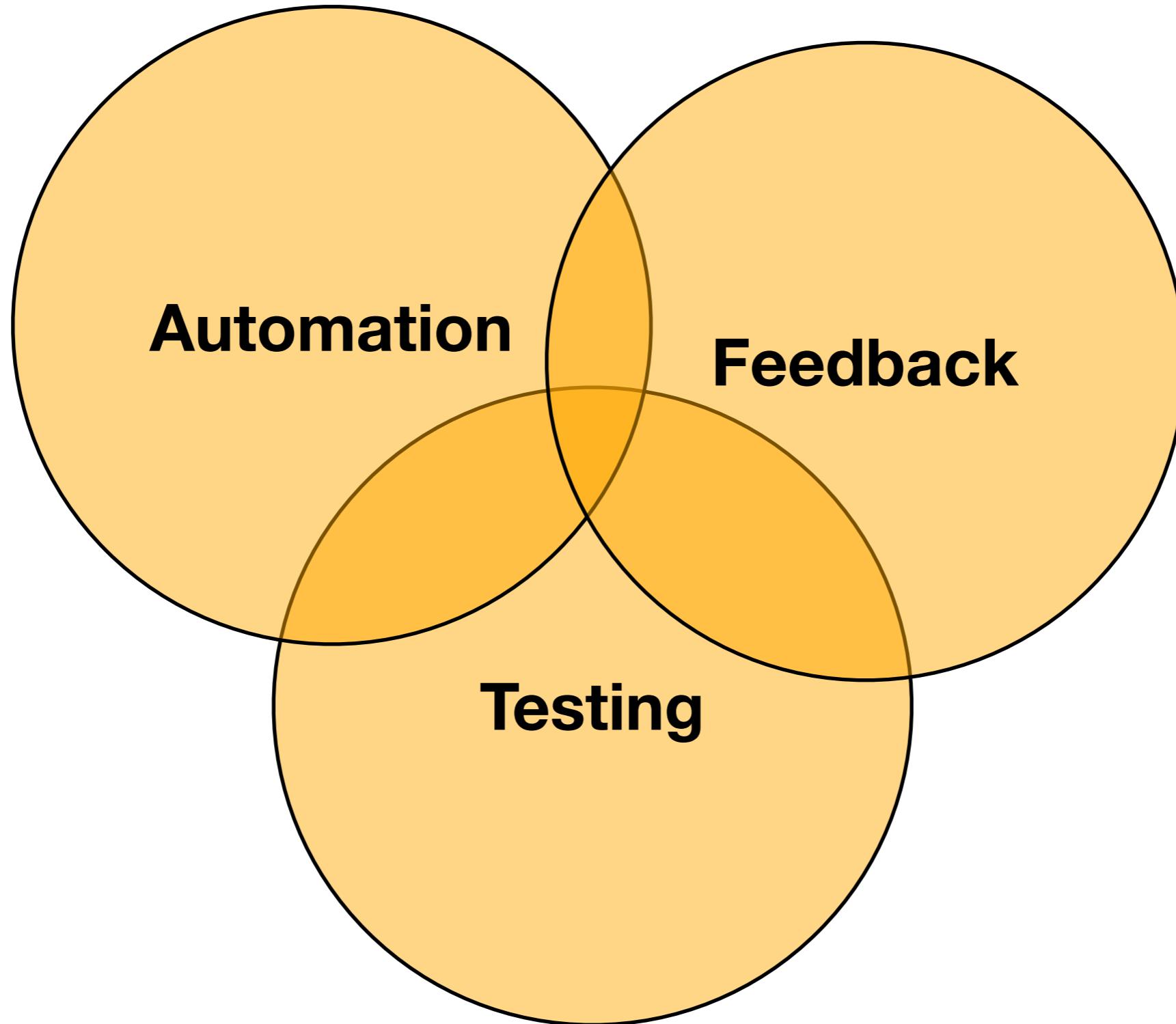
Principles of Software Delivery

1. Create a **Repeatable, Reliable Process** for Releasing Software
2. **Automate Almost Everything**
3. Keep Everything in **Version Control**
4. **If It Hurts, Do It More Frequently**, and Bring the Pain Forward
5. Build **Quality In**
6. “**Done**” Means Released
7. **Everybody Is Responsible for the Delivery Process**
8. **Continuous Improvement**

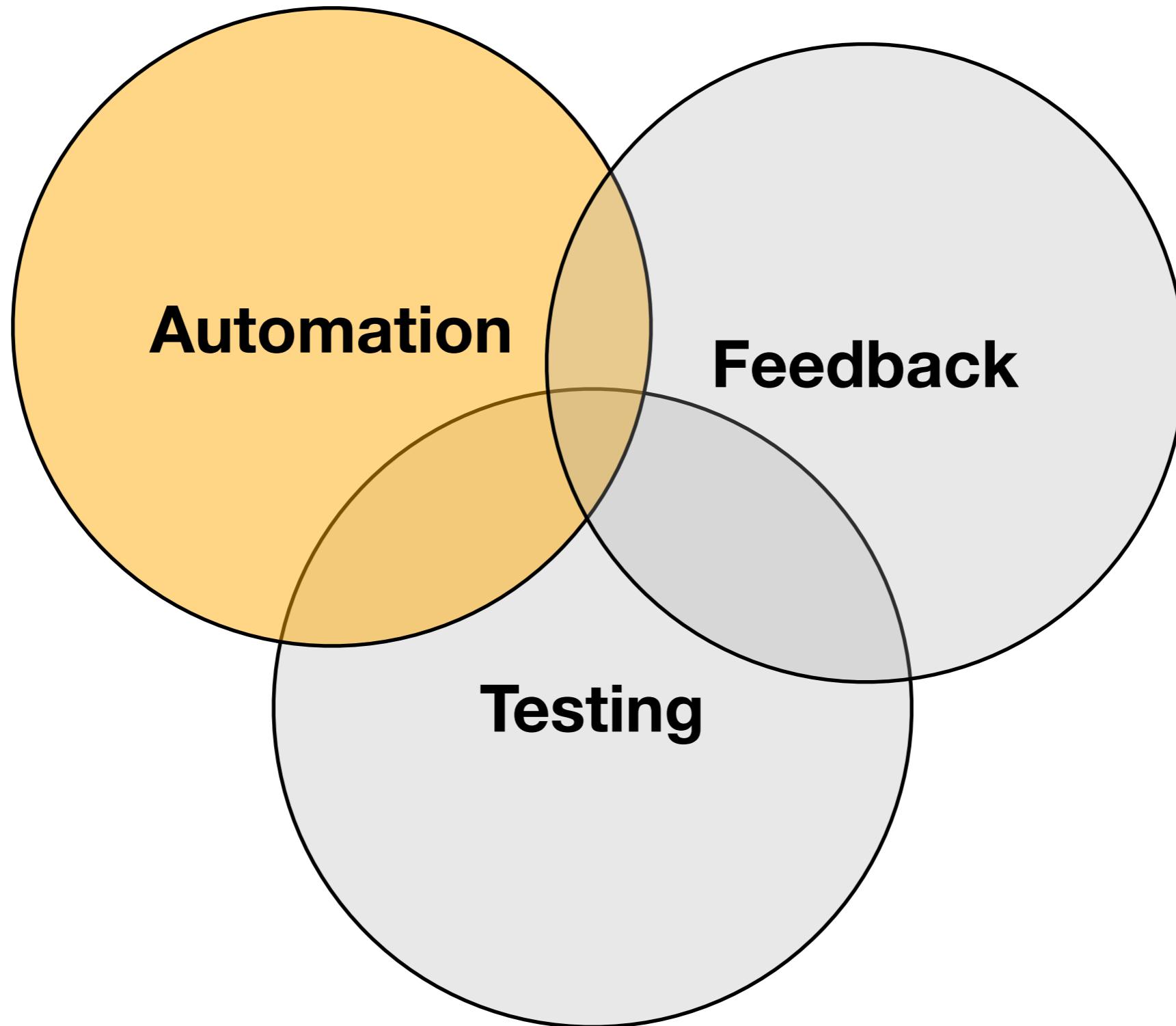
Frequent and Automated Releases

- We, and our fellow practitioners, have discovered that in order to achieve these goals—low cycle time and high quality—**we need to make frequent, automated releases of our software**. Why is this?
- **Automated.** If the build, deploy, test, and release process is not automated, it is not repeatable. Every time it is done, it will be different, because of changes in the software, the configuration of the system, the environments, and the release process. Since the steps are manual, they are error-prone, and there is no way to review exactly what was done. This means there is no way to gain control over the release process, and hence to ensure high quality. **Releasing software is too often an art; it should be an engineering discipline.**
- **Frequent.** If releases are frequent, the delta between releases will be small. This significantly reduces the risk associated with releasing and makes it much easier to roll back. Frequent releases also lead to faster feedback—indeed, they require it. Much of this book concentrates on getting feedback on changes to your application and its associated configuration (including its environment, deployment process, and data) as quickly as possible.

The Pillars of Continuous Delivery



The Pillars of Continuous Delivery



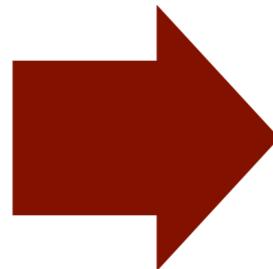
The Deployment Pipeline

*“The deployment pipeline has its foundations in the process of continuous integration and is in essence **the principle of continuous integration taken to its logical conclusion.**”*





Continuous Integration



Kanpai-Japan.com

Continuous Delivery

The Deployment Pipeline



Figure 1.1 *The deployment pipeline*

The Deployment Pipeline

*“Every change that is made to an application’s configuration, source code, environment, or data, **triggers the creation of a new instance of the pipeline.***

*One of the first steps in the pipeline is to **create binaries and installers.***

*The rest of the pipeline **runs a series of tests** on the binaries to prove that they can be released. Each test that the release candidate passes gives us more confidence that this particular combination of binary code, configuration information, environment, and data will work.*

*If the **release candidate** passes all the tests, it can be **released.**”*

The Deployment Pipeline

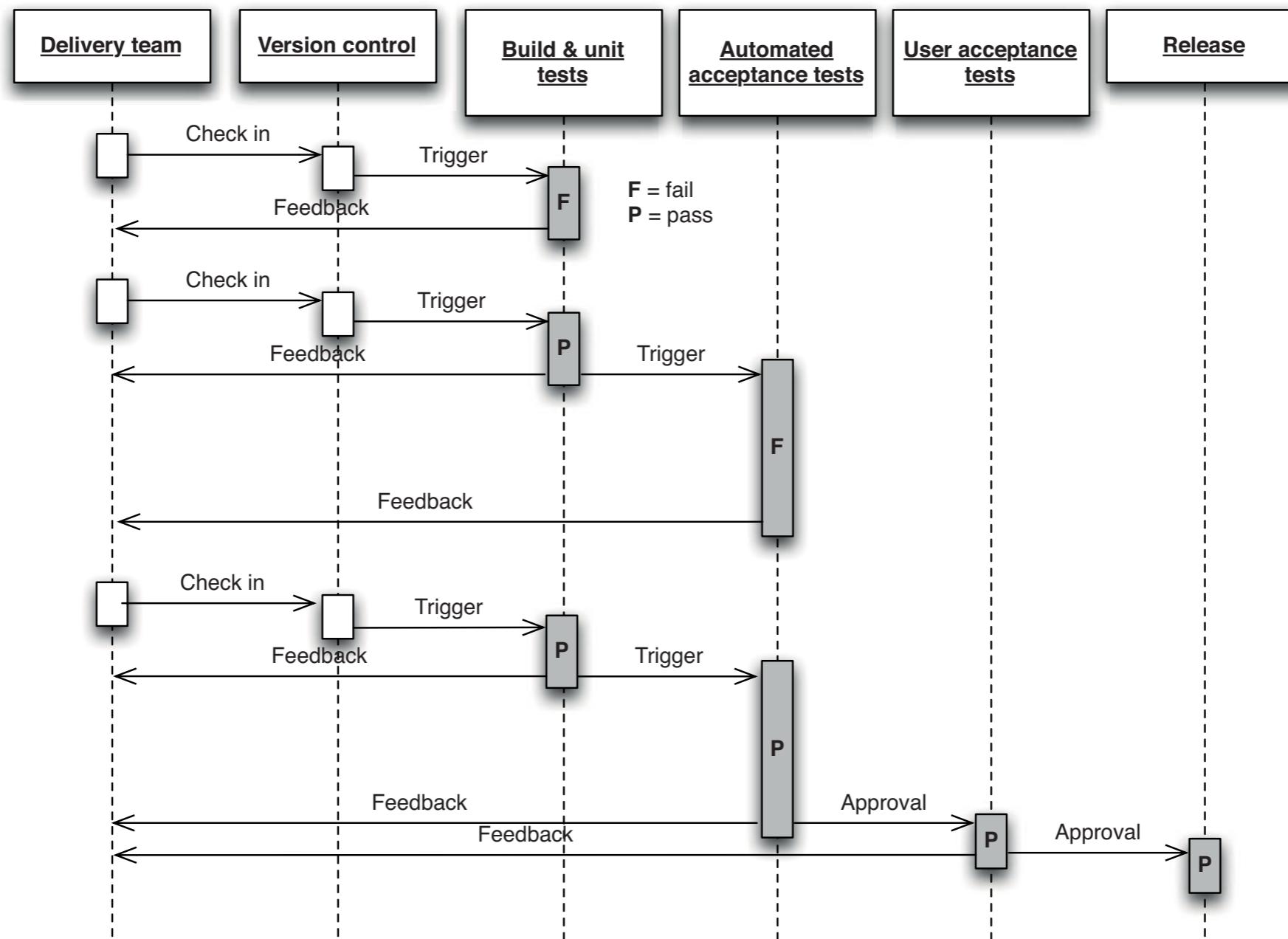


Figure 5.2 *Changes moving through the deployment pipeline*

The Deployment Pipeline

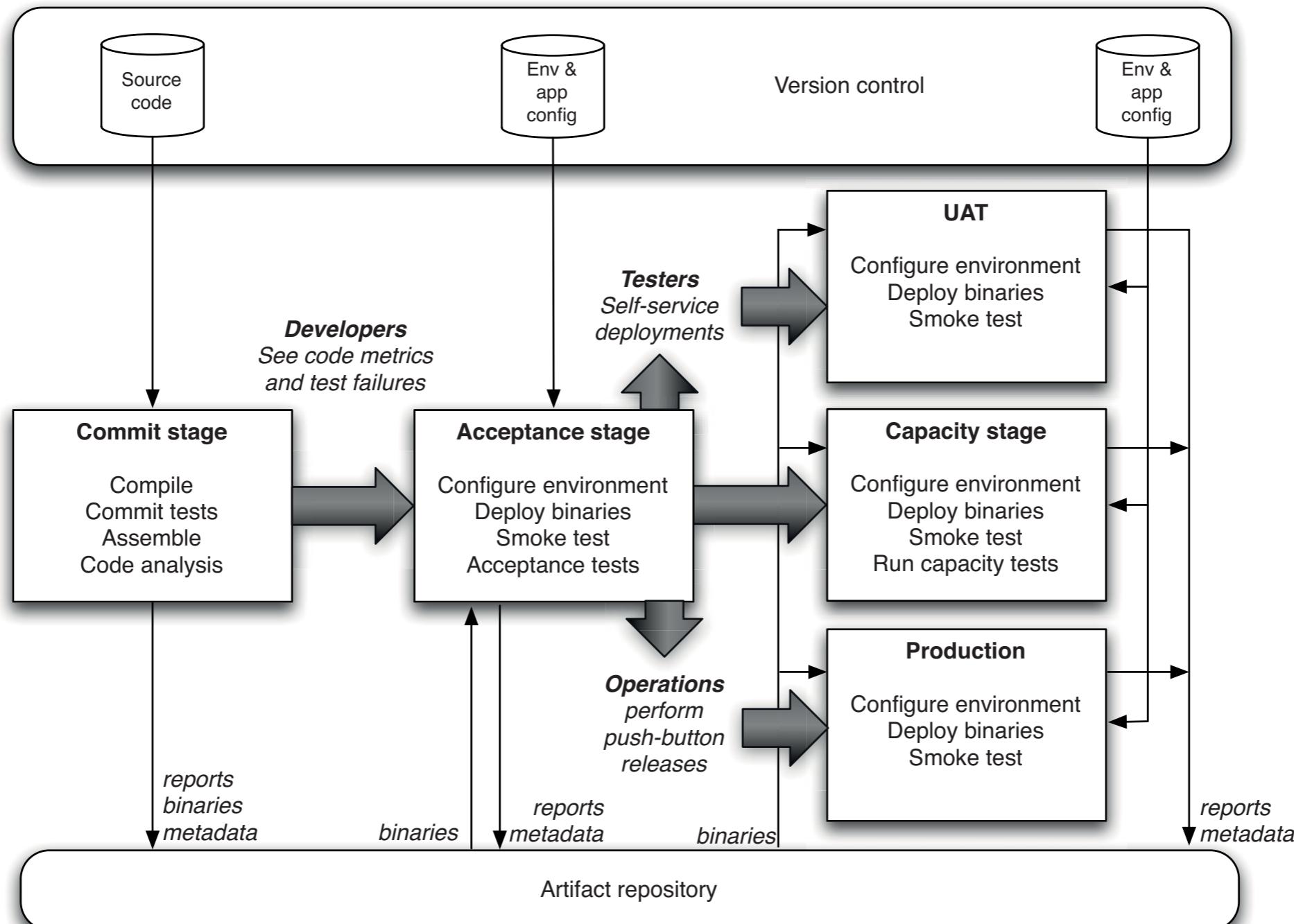


Figure 5.4 Basic deployment pipeline

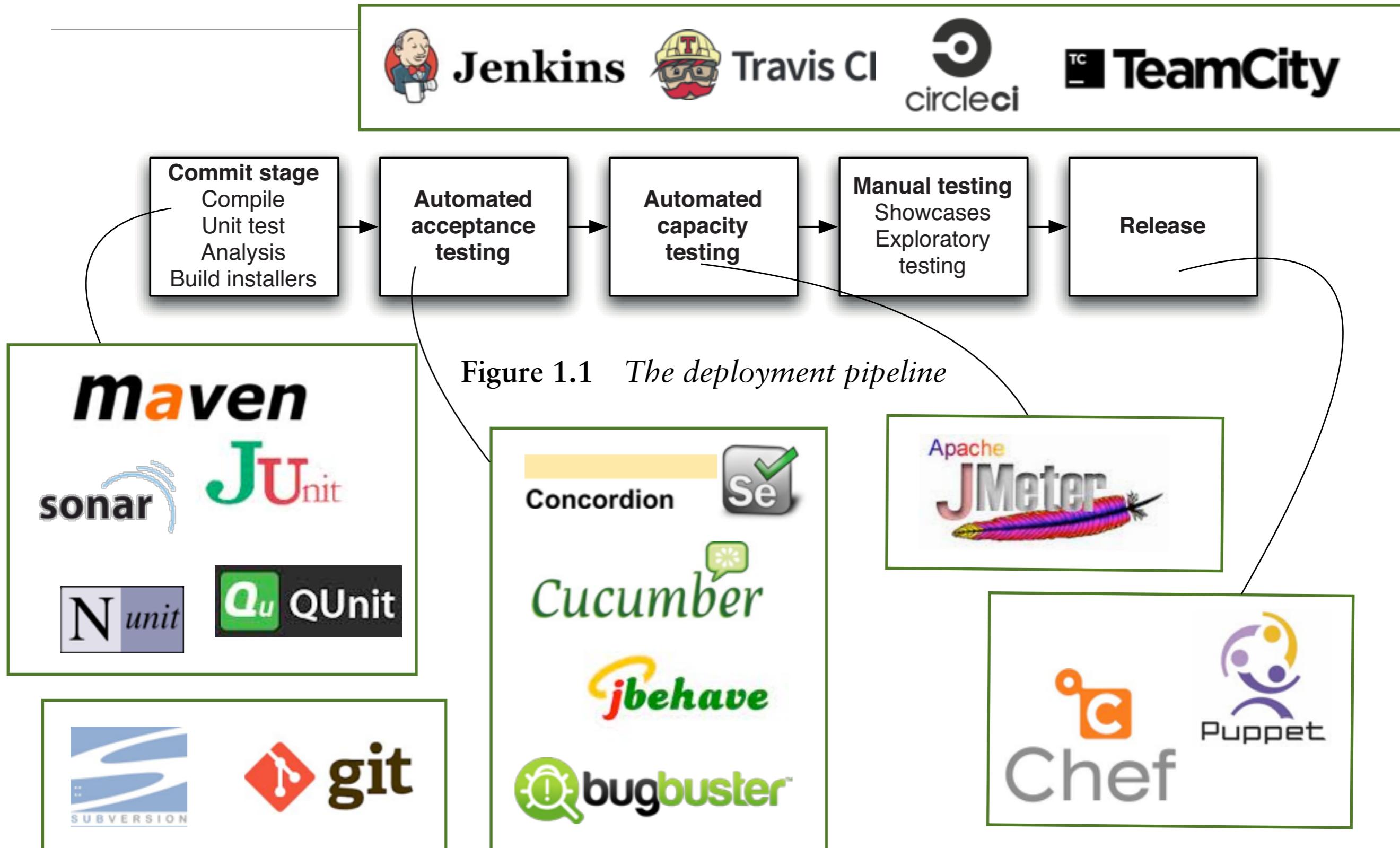
The Commit Stage

- The commit stage begins with a change to the state of the project—that is, a **commit to the version control system**.
- It **ends with either a report of failure or**, if successful, **a collection of binary artifacts and deployable assemblies** to be used in subsequent test and release stages, as well as reports on the state of the application.
- Ideally, a commit stage should **take less than five minutes to run**, and certainly no more than ten.
- The commit stage represents, in more ways than one, **the entrance into the deployment pipeline**. Not only is it the point at which a new release candidate is created; it is also where many teams start when they begin to implement a deployment pipeline. **When a team implements the practice of continuous integration, it creates a commit stage in the process of doing so.**

Implementing a Deployment Pipeline

- **Take an agile approach.** Start simple, iterate and improve on a continuous basis.
- Consider the following elements:
 - Automate the **build** and **deployment** process.
 - Automate **unit tests** and **code analysis**.
 - Automate **acceptance tests**.
 - Automate **releases**.

Tools



Summary

*“There is **no one-size-fits-all solution** to the complex problem of implementing a deployment pipeline.*

*The crucial point is to create a system of record that **manages each change from check-in to release**, providing the **information** you need to discover problems as early as possible in the process.”*

Summary

*“Having an implementation of the deployment pipeline can then be used to **drive out inefficiencies** in your process so you can make your feedback cycle faster and more powerful, perhaps by adding more automated acceptance tests and parallelizing them more aggressively, or by making your testing environments more production-like, or by implementing better configuration management processes.”*

Summary

*“A deployment pipeline, in turn, depends on having some **foundations** in place: **good configuration management, automated scripts for building and deploying your application, and automated tests** to prove that your application will deliver value to its users.*

*It also requires **discipline**, such as ensuring that only changes that have passed through the automated build, test, and deployment system get released.”*

Resources

- <http://continuousdelivery.com>



The screenshot shows the homepage of the Continuous Delivery website. At the top, there's a banner featuring a red bridge against a sunset sky with the text "CONTINUOUS DELIVERY". Below the banner, there are three orange buttons: "BLOG", "PUBLICATIONS" (which is highlighted), and "TALKS ABOUT". The main content area has two columns. The left column is titled "GET THE BOOK" and contains links to purchase the book "Continuous Delivery" on Amazon (hardback, Kindle) and InformIT (pdf, epub, mobi). It also lists translations in Chinese and Japanese. The right column is titled "Publications" and lists various publications and conference talks by Jez Humble and David Farley, including a link to download the Agile Release Management white paper. Further down, there are sections for "Papers", "Articles and Interviews", "Recordings", and "RECENT POSTS" which lists recent blog posts. On the far right, there's a sidebar with a "FOLLOW ME ON twitter" button, a "SIGN UP FOR THE NEWSLETTER" form, and a "RECENT POSTS" sidebar.

CONTINUOUS DELIVERY

BLOG PUBLICATIONS TALKS ABOUT

GET THE BOOK

 Amazon (hardback, Kindle)
InformIT (pdf, epub, mobi)
中文 (in Chinese)
日本語 (in Japanese)

You can also see a list of all my publications and talks, including slides, on the Publications page.

GET THE SOFTWARE

go

Publications

Here is a list of all my publications and conference talks, by category, with newest first.

Book

Jez Humble and David Farley, [Continuous Delivery](#), Addison-Wesley, 2010

Papers

["Why Enterprises Must Adopt Devops to Enable Continuous Delivery"](#), Cutter IT Journal Vol. 24 No. 8 (August 2011, registration required)
["Agile Release Management"](#) white paper, ThoughtWorks, 2010 (registration required)
Jez Humble, Chris Read, Dan North, ["The Deployment Production Line"](#), Proceedings of Agile 2006, IEEE Computer Society ([Download](#))

Articles and Interviews

[Is the Enterprise Ready for DevOps?](#), InfoQ, 1 August 2012
[Interview with Chris Little for BMC DevOps Leadership Series](#), 28 March 2012
[Four Principles of Low-Risk Software Releases](#), InformIT, 16 February 2012
[Analysis for Continuous Delivery: Five Core Practices](#), InformIT, 25 January 2012
[Simple Talk's Geek of the Week](#), <http://simple-talk.com/>, 28 September 2011
[Interviewed by Forrester's Jeffrey Hammond on Devops and Continuous Delivery](#), <http://forrester.com/>, 9 September 2011
[Interview with me and Dave Farley on Continuous Delivery](#), <http://infoq.com/>, 6 September 2011
["Tired of Playing Ping-Pong with Dev, QA and Ops?"](#), <http://ciupdate.com/>, 10 May 2011
["Five predictions for 2011"](#), <http://cmcrossroads.com/>, 4 February 2011
["What DevOps Means for Enterprises"](#), <http://agileweboperations.com/>, 18 January 2011
["Continuous Delivery: The Value Proposition"](#), <http://informit.com/>, 26 October 2010
["Continuous Delivery: Anatomy of the Deployment Pipeline"](#), <http://informit.com/>, 7 September 2010 (Chapter 5 of the book)

Recordings

[Hanselminutes: Me and Martin Fowler interviewed by Scott Hanselman](#), 2 October 2012
[Devops Cafe: Interviewed by Damon Edwards and John Willis on continuous delivery](#), 26 September 2012
[Devnology interview Dave Farley and me about continuous delivery](#), 27 November 2011
[30m interview on Continuous Delivery, the Lean Startup, and Devops](#) during Agile 2011
[Interviewed with Martin Fowler](#) during QCon San Francisco, 2010

FOLLOW ME ON twitter

SIGN UP FOR THE NEWSLETTER

email address

[SIGN UP >>](#)

RECENT POSTS

There's No Such Thing as a "Devops Team"
Elisabeth Hendrickson Discusses Agile Testing
Continuous Delivery: The Case of Apple
John Allspaw Discusses Devops and Continuous Delivery
Why Software Development Methodologies Suck

ARCHIVES

October 2012 (3)
September 2012 (1)
August 2012 (1)
July 2012 (2)
February 2012 (1)
January 2012 (1)
December 2011 (1)
July 2011 (2)
May 2011 (1)
January 2011 (1)
November 2010 (1)
October 2010 (1)
September 2010 (1)
August 2010 (2)
July 2010 (1)
June 2010 (1)
February 2010 (1)



First experiments with Jenkins

- **Last week**
 - When you have made the 2 Spring Boot tutorials, you have run “**mvn clean install**” on your local machine. As a result, you have generated a .jar file in your “target” folder.
- **Today**
 - **Let’s configure Jenkins to do the exact same procedure.** This would allow us to do a daily build of the micro-service (get the last version of the code and build it with maven).
 - This is going to be a very, **very basic build pipeline with only the commit stage.**

Overview

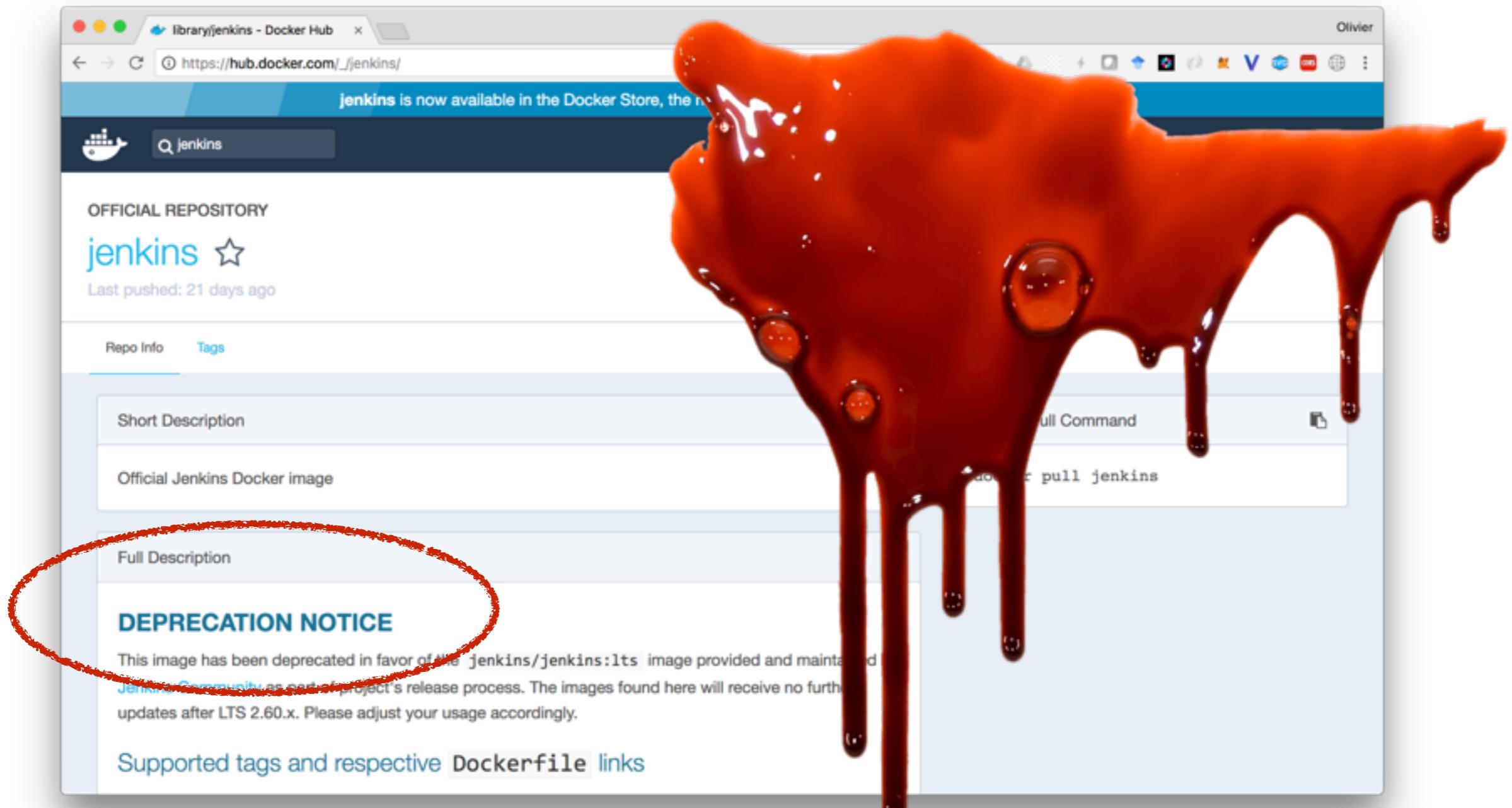
- **Jenkins** is one of the most popular continuous integration / continuous delivery servers.
- **There are many others.** Some are “cloud-first” services (e.g. Travis CI, CircleCI, etc.), others are “on-premise” products (e.g. GoCD, TeamCity, Bamboo, etc.).
- It is an **open source project** with a large community (lots of plugins) and the support of a commercial company (Cloudbees).
- It was initially created with a Java / maven focus but has a wider scope today.
- A **big transition** is underway, from being a continuous integration to becoming a full-fledged continuous delivery infrastructure. See the “pipeline” and “stage view” plugins released by Cloudbees.

First steps

- **The first thing that we want to do is to get a Jenkins server running on our machine and have a look at its web UI.**
- How do we do that?
 - We could make sure that we have a JVM on our machine, download the Jenkins binaries, follow the instructions, install it on our machine. We could then play with it. If we don't like it, there are certainly instructions for uninstalling it.
 - Or we can take advantage of Docker!

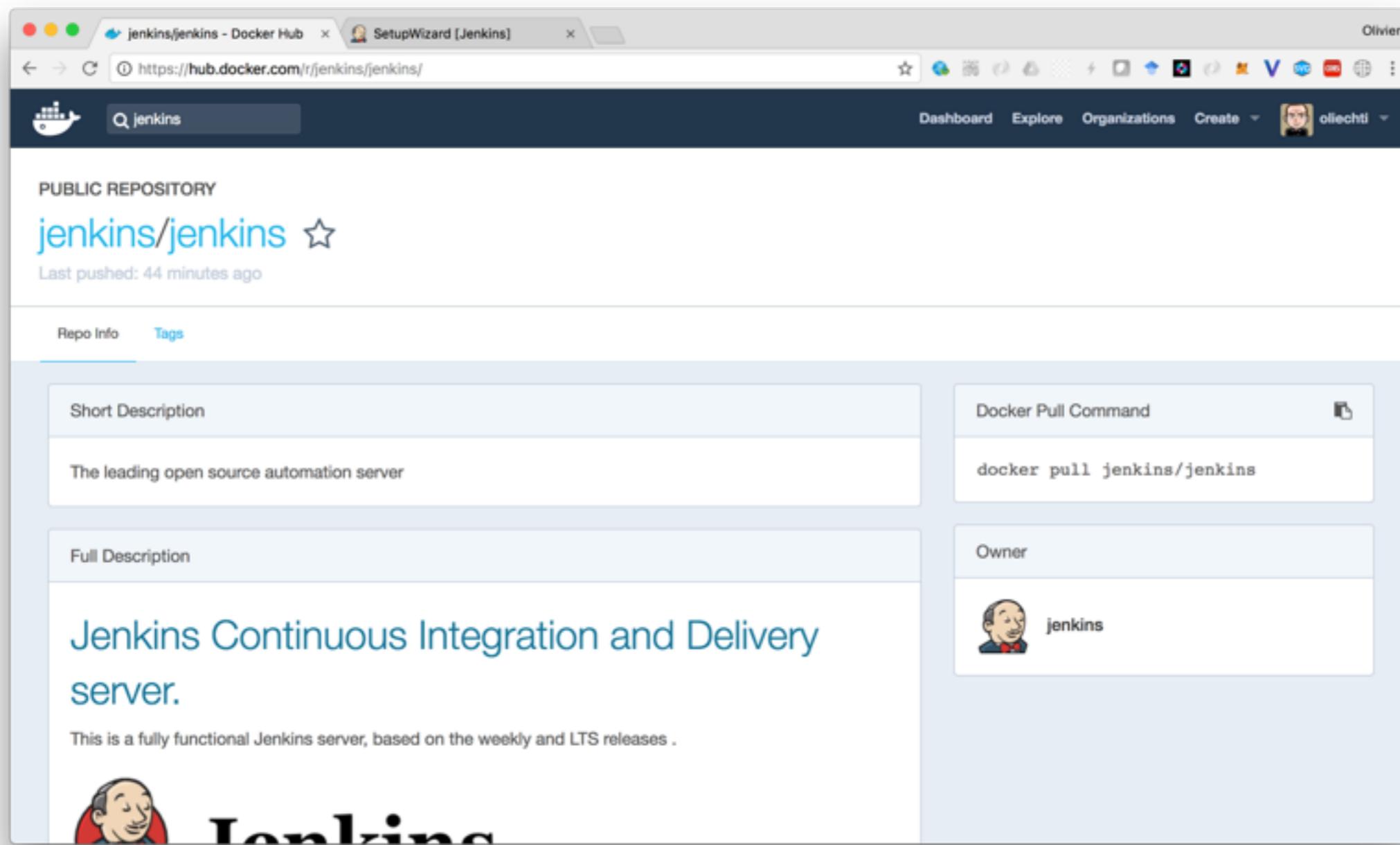
Ready to use Jenkins images

- Warning: there are many Jenkins images on Dockerhub...



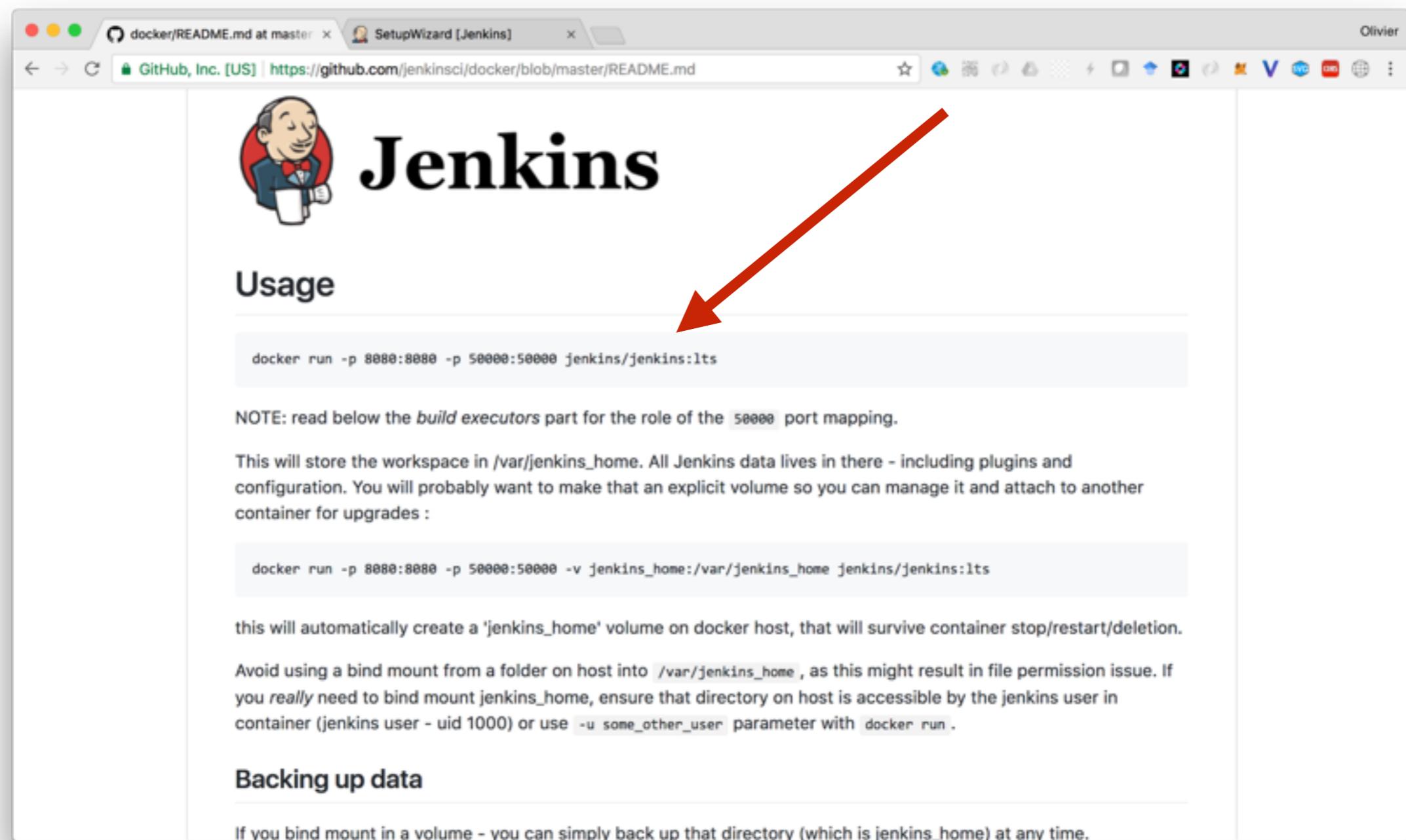
Ready to use Jenkins images

- Use the jenkins/jenkins image



The official Jenkins Docker image

- It only takes one command to run it (notice the port mapping)!



The official Jenkins Docker image

- It only takes one command to run it (notice the port mapping)!

```
$ docker run -p 9080:8080 -p 50000:50000 jenkins
```

host port : container port

host

vm (192.168.42.42)

9080 50000

8080 50000

jenkins container

The official Jenkins Docker image

- **Read carefully what appears in the docker log. You will find the default admin password (for THIS container).**
- **If you are new to Docker, try to ‘docker run -it XXX /bin/sh’ (where XXX is the name of your container)**

```
*****  
*****  
*****
```

Jenkins initial setup is required. An admin user has been created and a password generated.
Please use the following password to proceed to installation:

9f87ada80d464782b2249cad4fd1b688

This may also be found at: /var/jenkins_home/secrets/initialAdminPassword

```
*****  
*****  
*****
```

Jenkins [Jenkins]

Not Secure | 192.168.99.100:8888/login?from=%2F Olivier

Getting Started

Unlock Jenkins

To ensure Jenkins is securely set up by the administrator, a password has been written to the log ([not sure where to find it?](#)) and this file on the server:

`/var/jenkins_home/secrets/initialAdminPassword`

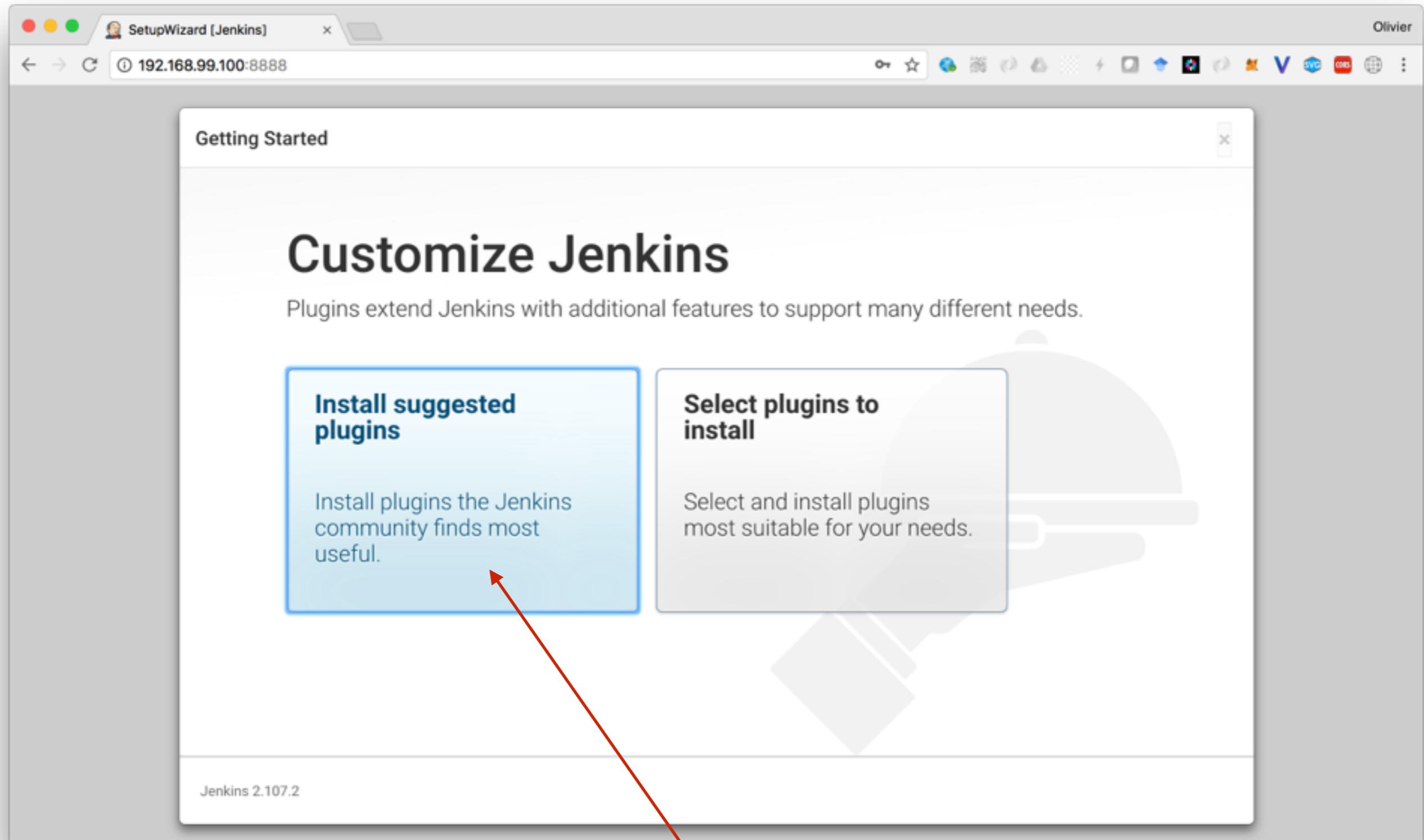
Please copy the password from either location and paste it below.

Administrator password

.....

the password from previous screen

Continue



Doing that in Provence... can be a challenge
(check the UI and the log on your Docker console)

SetupWizard [Jenkins] Not Secure | 192.168.99.100:8888 Olivier

Getting Started

Create First Admin User

Username:

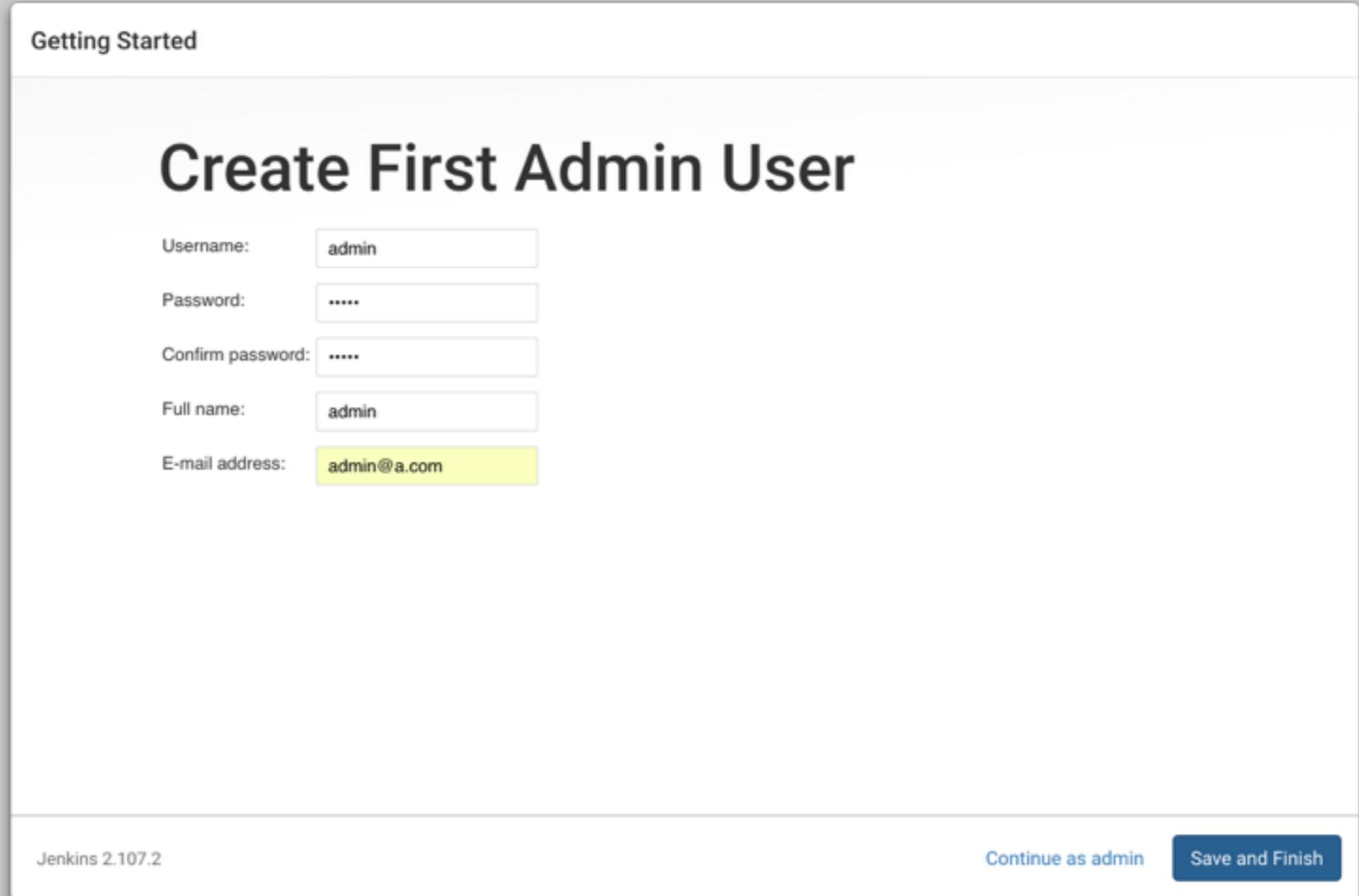
Password:

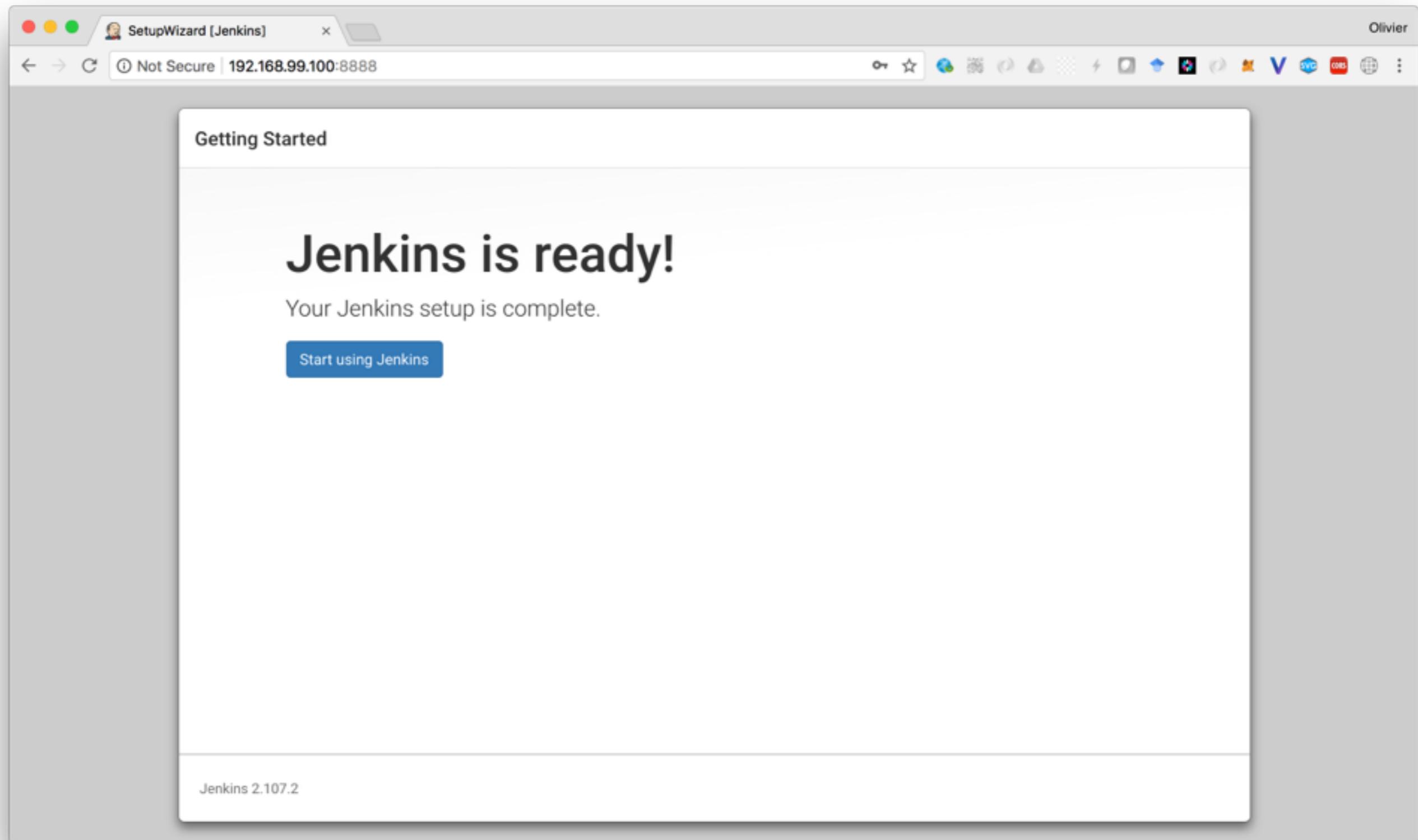
Confirm password:

Full name:

E-mail address:

Jenkins 2.107.2 Continue as admin Save and Finish





The screenshot shows the Jenkins dashboard at 192.168.99.100:8888. The left sidebar includes links for New Item, People, Build History, Manage Jenkins, My Views, Credentials, and New View. The main area features a 'Welcome to Jenkins!' message with a call to action: 'Please [create new jobs](#) to get started.' A red arrow points from this message to the 'Manage Jenkins' link in the sidebar. Another red arrow points from the 'Manage Jenkins' link in the sidebar to the 'Jobs' section of the 'Manage Jenkins' page, which is not fully visible in the screenshot.

Welcome to Jenkins!

Please [create new jobs](#) to get started.

Jenkins jobs are the pipelines that we have talked about in the theory. Often, you configure multiple jobs on a single Jenkins server. There are different types of jobs - here we use a maven job.

Jenkins is extremely extensible and configurable. The community is providing a lot of plugins for all sorts of things. Installing a plugin usually adds elements to the UI.

Page generated: Apr 19, 2018 12:01:02 PM UTC [REST API](#) [Jenkins ver. 2.107.2](#)

The screenshot shows the Jenkins Manage Jenkins interface. On the left, there's a sidebar with links like 'New Item', 'People', 'Build History', 'Manage Jenkins' (which is highlighted in purple), 'My Views', 'Credentials', and 'New View'. Below that are sections for 'Build Queue' (empty) and 'Build Executor Status' (2 Idle). The main content area is titled 'Manage Jenkins' and lists several configuration options: 'Configure System', 'Configure Global Security', 'Configure Credentials', 'Global Tool Configuration' (which has a red arrow pointing to it from the left), 'Reload Configuration from Disk', 'Manage Plugins', 'System Information', and 'System Log'. The URL at the bottom is 192.168.99.100:8888/configureTools.

Jenkins will need to delegate (most of the) work to maven. So, we need to install maven in our Docker container. We could do that in the Dockerfile, but we can ask Jenkins to do that for us (via the Global Tool Configuration)

The screenshot shows the Jenkins Global Tool Configuration interface. Under the 'Maven' section, there is a 'Name' field containing 'mymaven'. Below it, the 'Install automatically' checkbox is checked. A red arrow points from the text 'Jenkins will need to delegate (most of the) work to maven.' to the 'Install automatically' checkbox.

Global Tool Configuration [Jen... X] Not Secure | 192.168.99.100:8888/configureTools/ Olivier

Jenkins > Global Tool Configuration

Install automatically

Gradle

Gradle installations

List of Gradle installations on this system

Ant

Ant installations

List of Ant installations on this system

Maven

Maven installations

Name	mymaven
------	---------

Install automatically

Version 3.5.3

Jenkins will need to delegate (most of the) work to maven. So, we need to install maven in our Docker container. We could do that in the Dockerfile, but we can ask Jenkins to do that for us (via the Global Tool Configuration)

S  Manage Jenkins [Jenkins] X 

← → ⌛ ⓘ 192.168.99.100:8888/manage              

Jenkins  admin | log out [DISABLE AUTO REFRESH](#)

New Item 
People 
Build History 
Manage Jenkins 
My Views 
Credentials 
New View 

Build Queue  -
No builds in the queue.

Build Executor Status  -
1 Idle
2 Idle

Manage Jenkins

 **Configure System**
Configure global settings and paths.

 **Configure Global Security**
Secure Jenkins; define who is allowed to access/use the system.

 **Configure Credentials**
Configure the credential providers and types

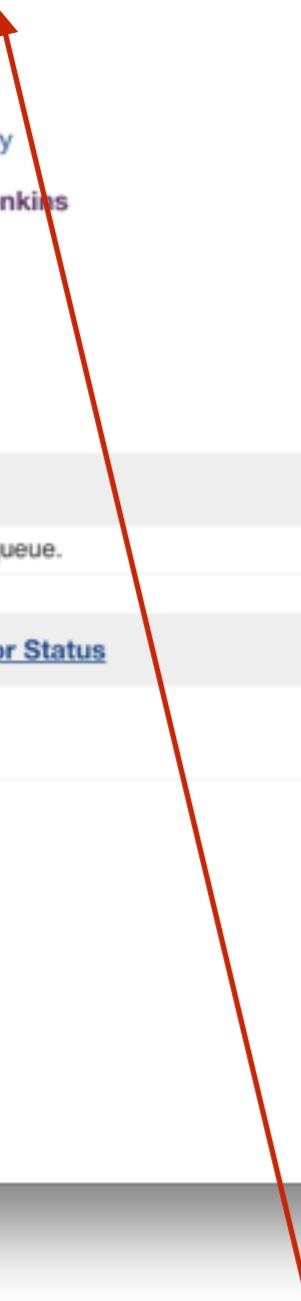
 **Global Tool Configuration**
Configure tools, their locations and automatic installers.

 **Reload Configuration from Disk**
Discard all the loaded data in memory and reload everything from file system. Useful when you modified config files directly on disk.

 **Manage Plugins**
Add, remove, disable or enable plugins that can extend the functionality of Jenkins.

 **System Information**
Displays various environmental information to assist trouble-shooting.

 **System Log**
System log captures output from `java.util.logging` related to Jenkins.



Let's create a new job (pipeline).

The screenshot shows the Jenkins web interface for creating a new item. In the top left, it says "New Item [Jenkins]". The address bar shows "Not Secure | 192.168.99.100:8888/view/all/newJob". The top right shows the user "Olivier" and navigation icons. The main title is "Jenkins". Below it, "Jenkins > All >" is visible. A search bar and "admin" link are on the right.

Enter an item name

build microservice with maven

» Required field

Freestyle project

This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

Pipeline

Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.

Multi-configuration project

Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.

Folder

Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders.

GitHub Organization

Scans a GitHub organization (or user account) for all repositories matching some defined markers.

Multibranch Pipeline

Creates a set of Pipeline projects according to detected branches in one SCM repository.

OK

A red arrow points from the text "For simple maven builds, we use the ‘freestyle’ job type" at the bottom to the "Freestyle project" section in the screenshot.

**For simple maven builds, we use
the “freestyle” job type**

The screenshot shows the Jenkins configuration interface for a job named "build microservice with maven". The "Source Code Management" tab is selected. Under the "Repositories" section, "Git" is chosen as the provider. The "Repository URL" is set to <https://github.com/openaffect/openaffect-server.git>. The "Branch Specifier (blank for 'any')" is set to */master. A red arrow points from the question "Where is the git repo where you have the code of your micro-service?" to the "Repository URL" field.

Where is the git repo where you have the code of your micro-service?

The screenshot shows the Jenkins job configuration interface for a job named "build microservice with maven". The "Build Environment" tab is selected. Under "Build", a dropdown menu is open, showing various build steps. The "Invoke top-level Maven targets" option is highlighted with a blue selection bar and has a red arrow pointing down to the explanatory text below.

General Source Code Management Build Triggers **Build Environment** Build Post-build Actions

Poll SCM

Build Environment

Delete workspace before build starts ?
 Use secret text(s) or file(s) ?
 Abort the build if it's stuck ?
 Add timestamps to the Console Output ?
 With Ant ?

Build

Add build step ▾

- Execute Windows batch command
- Execute shell
- Invoke Ant
- Invoke Gradle script
- Invoke top-level Maven targets**
- Run with timeout
- Set build status to "pending" on GitHub commit

Save **Apply**

Page generated: Apr 19, 2018 12:13:54 PM UTC [REST API](#) [Jenkins ver. 2.107.2](#)

We have only one thing to do: launch maven (with a proper reference to the pom.xml file)

The screenshot shows the Jenkins job configuration for a Maven build. The job is named "build microservice with maven". The configuration page has tabs for General, Source Code Management, Build Triggers, Build Environment, Build, and Post-build Actions. The Build tab is selected. Under the "Invoke top-level Maven targets" section, there are fields for Maven Version (set to "mymaven"), Goals (set to "clean install"), POM (set to "microservices/oa-server/pom.xml"), and Properties. Below these are sections for JVM Options, Inject build variables, Use private Maven repository, Settings file (set to "Use default maven settings"), and Global Settings file (set to "Use default maven global settings"). At the bottom are Save and Apply buttons.

The tool we previously installed

we want a mvn clean install

where in the repo is your pom.xml

build microservice with maven x openaffect-server/microservice x Olivier

192.168.99.100:8888/job/build%20microservice%20with%20maven/

Jenkins build microservice with maven DISABLE AUTO REFRESH

Back to Dashboard

Status

Changes

Workspace

Build Now

Delete Project

Configure

Build History trend — find RSS for all RSS for failures

Workspace

Recent Changes

Permalinks

Page generated: Apr 19, 2018 12:19:12 PM UTC REST API Jenkins ver. 2.107.2

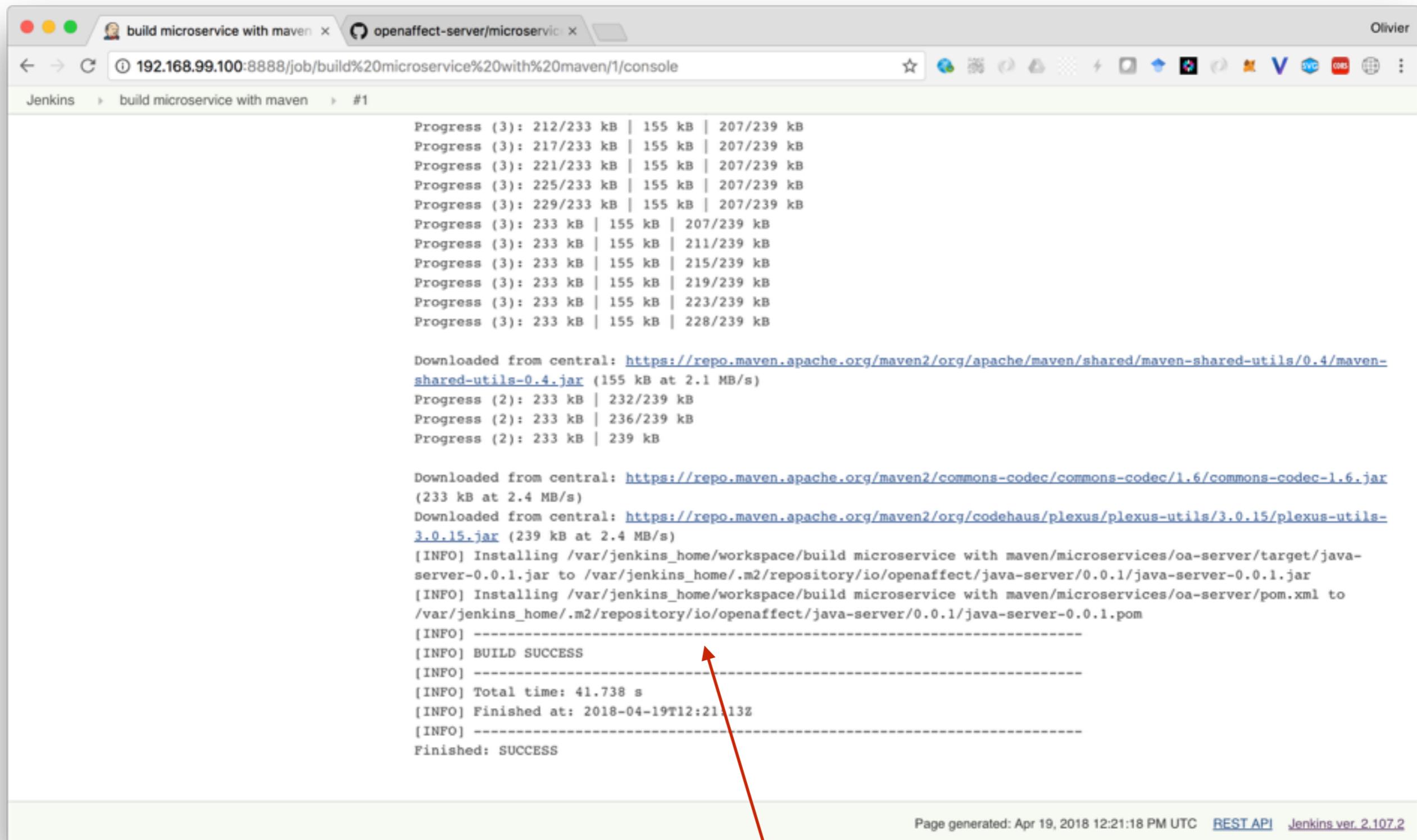
In this first example, we trigger the pipeline execution manually

The screenshot shows a Jenkins project named "build microservice with maven". The main content area is titled "Project build microservice with maven". On the left, there's a sidebar with links: "Back to Dashboard", "Status", "Changes", "Workspace", "Build Now", "Delete Project", and "Configure". Below this is a "Build History" section with a search bar and a table. The table has one row: a radio button labeled "#1", the date "Apr 19, 2018 12:20 PM", and a progress bar. At the bottom of this section are two RSS feed links: "RSS for all" and "RSS for failures". A red arrow points from the text below to the "RSS for all" link. To the right of the build history are links for "Workspace" and "Recent Changes". Under "Permalinks", there's a list with one item: "Last build (#1), 6.2 sec ago". The bottom right corner of the page shows the footer: "Page generated: Apr 19, 2018 12:20:25 PM UTC REST API Jenkins ver. 2.107.2".

The pipeline execution has started. We can click on that link to see the details (what is displayed in the console)

The screenshot shows a Jenkins build console output page. At the top, there are two tabs: "build microservice with maven" and "openaffect-server/microservice". The URL in the address bar is "192.168.99.100:8888/job/build%20microservice%20with%20maven/1/console". The main content area is titled "Console Output". On the left, there is a sidebar with the following links: "Back to Project", "Status", "Changes", "Console Output" (which is highlighted with a red arrow pointing to it), "View as plain text", "Edit Build Information", "Delete Build", "Git Build Data", and "No Tags". The "Console Output" section displays the Maven build logs. It starts with "Skipping 861 KB.. [Full Log](#)". Below this, it shows the download of several JAR files from the central Maven repository, including "jcl-over-slf4j-1.5.6.jar", "commons-lang3-3.1.jar", "maven-reporting-api-3.0.jar", "surefire-api-2.18.1.jar", "maven-metadata-2.2.1.jar", "maven-toolchain-2.2.1.jar", "maven-error-diagnostics-2.2.1.jar", and "maven-plugin-annotations-3.3.jar". Progress bars are shown for each download, such as "Progress (2): 13 kB | 2.2/10 kB".

The pipeline execution has started. We can click on that link to see the details (what is displayed in the console)



build microservice with maven × openaffect-server/microservice × Olivier

← → ⌂ 192.168.99.100:8888/job/build%20microservice%20with%20maven/1/console

Jenkins > build microservice with maven > #1

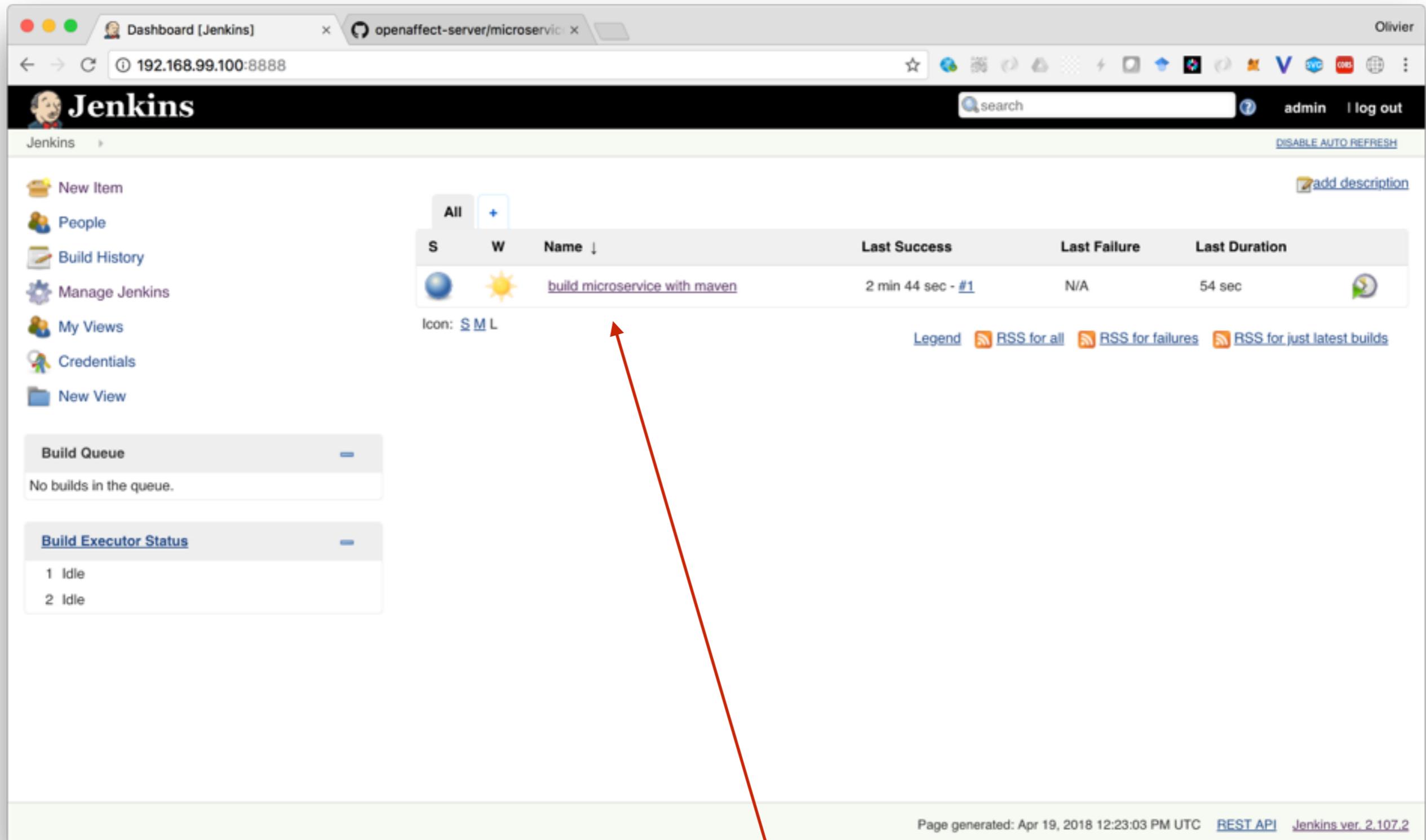
```
Progress (3): 212/233 kB | 155 kB | 207/239 kB
Progress (3): 217/233 kB | 155 kB | 207/239 kB
Progress (3): 221/233 kB | 155 kB | 207/239 kB
Progress (3): 225/233 kB | 155 kB | 207/239 kB
Progress (3): 229/233 kB | 155 kB | 207/239 kB
Progress (3): 233 kB | 155 kB | 207/239 kB
Progress (3): 233 kB | 155 kB | 211/239 kB
Progress (3): 233 kB | 155 kB | 215/239 kB
Progress (3): 233 kB | 155 kB | 219/239 kB
Progress (3): 233 kB | 155 kB | 223/239 kB
Progress (3): 233 kB | 155 kB | 228/239 kB

Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/shared/maven-shared-utils/0.4/maven-shared-utils-0.4.jar (155 kB at 2.1 MB/s)
Progress (2): 233 kB | 232/239 kB
Progress (2): 233 kB | 236/239 kB
Progress (2): 233 kB | 239 kB

Downloaded from central: https://repo.maven.apache.org/maven2/commons-codec/commons-codec/1.6/commons-codec-1.6.jar (233 kB at 2.4 MB/s)
Downloaded from central: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-utils/3.0.15/plexus-utils-3.0.15.jar (239 kB at 2.4 MB/s)
[INFO] Installing /var/jenkins_home/workspace/build microservice with maven/microservices/oa-server/target/java-server-0.0.1.jar to /var/jenkins_home/.m2/repository/io/openaffect/java-server/0.0.1/java-server-0.0.1.jar
[INFO] Installing /var/jenkins_home/workspace/build microservice with maven/microservices/oa-server/pom.xml to /var/jenkins_home/.m2/repository/io/openaffect/java-server/0.0.1/java-server-0.0.1.pom
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 41.738 s
[INFO] Finished at: 2018-04-19T12:21:13Z
[INFO] -----
Finished: SUCCESS
```

Page generated: Apr 19, 2018 12:21:18 PM UTC [REST API](#) [Jenkins ver. 2.107.2](#)

You have seen last week on your terminal, when doing the Spring Boot tutorials



The screenshot shows the Jenkins dashboard at 192.168.99.100:8888. The left sidebar contains links for New Item, People, Build History, Manage Jenkins, My Views, Credentials, and New View. The main area displays a table of builds with columns: S (Status), W (Last Result), Name (build microservice with maven), Last Success (2 min 44 sec - #1), Last Failure (N/A), and Last Duration (54 sec). A red arrow points from the text below to the 'W' column of the first build row.

S	W	Name	Last Success	Last Failure	Last Duration
		build microservice with maven	2 min 44 sec - #1	N/A	54 sec

Icon: S M L

Legend: [RSS for all](#) [RSS for failures](#) [RSS for just latest builds](#)

Page generated: Apr 19, 2018 12:23:03 PM UTC [REST API](#) [Jenkins ver. 2.107.2](#)

Back to the main page, we see that our last build was successful. Try to introduce a syntax error in your code. Commit and Push. Rebuild and you will see that Jenkins is unhappy.