

# CONTINUOUS DELIVERY

Hands on session

# Agenda

Introduction

- Purpose
- Topics covered
- Scheduling
- Environment
- Sample application

Use cases description per level (description, tooltips, details)

Debriefing and conclusions

# INTRODUCTION

# Introduction : purpose

The hands-on session is aimed to deliver a **global, practical knowledge of continuous delivery processes in a realistic context**:

- Self-organized teams (DEV/OPS, optionally BA/manager if any)
- Non static, changing needs and priorities (unexpected reviews)
- A toolset supporting the processes to masterize and set up
- An application to understand, extend and deliver through continuous delivery practices and tools

# Introduction : topics covered

PILLAR	PROCESS/PRACTICE	MANDATORY TOOLSET
AGILITY	KANBAN-LIKE MANAGEMENT	ZENHUB.IO
AGILITY	INSTANT FEEDBACK	SLACK
AGILITY	SOCIAL CODING	GITHUB
CRAFTSMANSHIP	CONTINUOUS BUILD/INTEGRATION	MAVEN / JENKINS / NEXUS
CRAFTSMANSHIP	UNIT/INTEGRATION TESTS BDD/TDD CODE COVERAGE/MUTATION TESTS PERFORMANCE TESTS	JUNIT (SUREFIRE/FAILSAFE) JGIVEN JACOCO/PITEST GATLING
DEVOPS	SINGLE VERSION PACKAGING AUTODEPLOYMENT : LOCAL AUTODEPLOYMENT : CLOUD PIPELINE ORCHESTRATION	JENKINS / DOCKER DOCKER DOCKER / TUTUM JENKINS
MEASUREMENT	CODE QUALITY ANALYSIS PROCESS METRICS	SONARQUBE ELASTICSEARCH/KIBANA

# Introduction : environment

A workstation per trainee is recommended, at least one per team :

- Recent Chrome/Firefox version
- Enough RAM to run Java IDE (IntelliJ / Eclipse)

At least two linux servers/VMs per team are required :

- 50 Gb HD, 8 Gb RAM, CentOS Core 7.2+ : build platform
- 16 Gb HD, 2 Gb RAM, Ubuntu 14.04+ : "cloud" deployment server

Full internet access from the servers/workstations is mandatory

Linux, maven and java skills in each team are highly recommended

# Introduction : scheduling

The toolset previously mentioned is mandatory, except IDE

The use cases can be processed in any order ... when possible

Track your progression on the kanban board

If stuck:

- Help is available on lower slides, from expert to beginner level
- Feel free to ask questions ... through kanban/instant messaging tool

KEEP IT SIMPLE : we do not matter about security aspects here ...

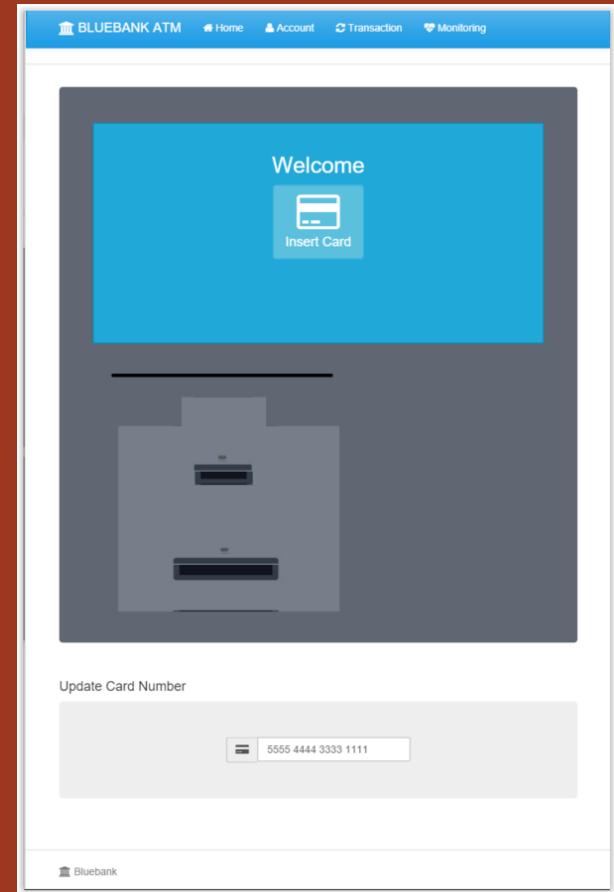
# Introduction : sample application

Bluebank : ATM application  
(thanks to Riad)

3 main operations :

- cash deposit
- cash withdrawal
- account inquiry

Server : java standalone (port 8180)  
UI : web site (requires HTTP server)



<https://github.com/Finaxys/bluebank-atm-server>

# USE CASES DESCRIPTION

# Use case #1 : Setup collaboration tools

To manage the team's follow-up during the hands-on session, set up the following items:

- ATM repository forked in this organization
- Collaborators included in this repository, including staff
- Assignment of issues in Zenhub (keep To Do/Progress/Done only)
- Set the start/due dates of the DEADLINE milestone
- Slack integration with the repository in a dedicated channel

# Use case #1 : Tooltips

For Zenhub/Github, register your personal account using Finaxys email

Use cases already exist in the ATM repository as issues

For slack, we have a Finaxys domain : <https://fnaxys.slack.com>

# Use case #1 : Details 1/4

Github : source code management tool including social coding features

- Worldwide standard
- Git model (distributed versioning control system)
- Wiki/markdown online documentation
- Collaboration features : issues, forks, pull requests
- "Forking" the repository allows to work on a personal/organization copy with ability to merge (or not) changes later

Github account signup

Create organization and invite members

Fork the repository into your organization =>

Assign the team to repository (Settings -> Collaborators and Teams)



# Use case #1 : details 2/4

Zenhub : Chrome/Firefox extension handling Github issues "a la agile"  
[Install Zenhub on Firefox/Chrome](#) (tour included)

On "Boards" screen keep only To Do/In Progress/Done/Closed stages  
ASK FOR STAFF TO COPY ISSUES ON YOUR FORKED PROJECT

Each issue can be

- assigned
- slided between stages
- prioritized (milestone)
- estimated (workload)

The screenshot shows the Zenhub interface with four columns of issues:

- To Do (3):**
  - vm-formation #10 revoir la partie pratique (enhancement)
  - vm-formation #2 scripter le setup initial (enhancement)
  - vm-formation #1 installer UCP (wontfix)
- In Progress (2):**
  - vm-formation #15 demander 1 VM dev + 1 VM run a LG (12 si 5 groupes + formateur) (enhancement, help wanted)
  - vm-formation #13 slider les TPs (enhancement)
- Done (4):**
  - vm-formation #3 voir les soucis de test/packaging avec Riad (bug, help wanted)
  - vm-formation #14 deploier avec Tutum via l'API (enhancement)
  - vm-formation #8 setupera la partie measurement (enhancement)
  - vm-formation #7 rediger la partie measurement (enhancement)
- Closed (6):**
  - vm-formation #6 fusionner les slides de Guillaume (devops)
  - vm-formation #9 revoir la partie theorique
  - vm-formation #5 configurer le conteneur jenkins pour build/docker
  - vm-formation #11 configurer le conteneur avec git + nodejs pour lancer le client
  - vm-formation #4 creer la SF avec docker
  - vm-formation #12 configurer le conteneur avec git + nodejs pour lancer le client

# Use case #1 : details 3/4

Assign, estimate and define start and due dates of DEADLINE milestone  
(beginning-end of session)

The screenshot shows a GitHub repository page for 'Finaxys / bluebank-atm-server'. The user is creating a new milestone. The 'Milestones' tab is selected. The 'Title' field contains 'DEADLINE'. The 'Due date (optional)' section shows a calendar for January 2016, with the 29th selected. Below the calendar, there is a large empty 'Description' text area. At the bottom, there are 'Cancel', 'Close milestone', and 'Save changes' buttons.



We are almost ready to go ...

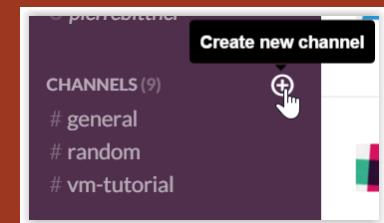
# Use case #1 : Details 3/3

Slack : instant messaging for enterprises solution

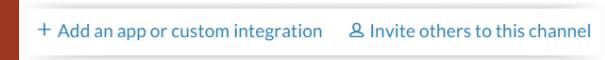
- allows public/shared/private channels to communicate
- integrates with Github and other tools to notify changes

If not already done yet, create an account : <https://finaxys.slack.com>

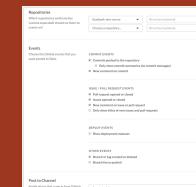
Create a channel for your team =>



Include github application =>



Map your repository to your channel =>



# Use case #2 : Build locally

This part must be performed on a local workstation IDE

- Get the source code and make the application compile and package (without tests for the moment)

## Use case #2 : tooltips

The compilation requires specific tools and configurations to find

The server part is handled by Maven

## Use case #2 : Details 1/6

ATM server build requires on the workstation:

- A Java development environment ([Eclipse/IntelliJ](#))
- A java 8 JDK compiler (current link [here](#))
- [Maven](#) (included in Eclipse/IntelliJ) for build/test/package/deploy
- [Protocol buffers](#) compiler (message serialization)

The IDE configuration must be this one:

- JDK 8 / Maven 3.x
- Environment variable PIPELINE\_VERSION enforced in maven build
- path to PROTOC executable available in the PATH

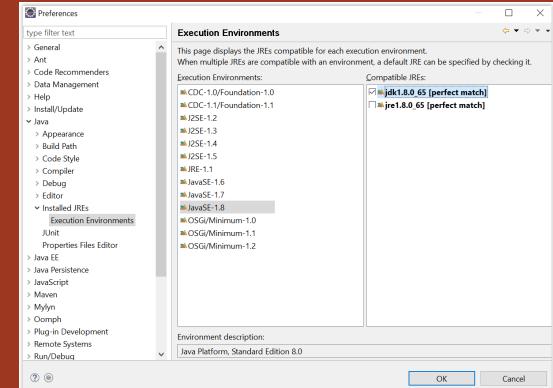
# Use case #2 : Details 2/6

Build environment sample (Eclipse / Windows)

=> use JDK not JRE

Build is managed by [maven](#)

- #1 open-source build, test, package, deployment tool
- Command-line tools with many ways to operate (IDEs, CI tools...)
- Descriptive configuration : pom.xml (c.f. [descriptor](#) and [XSD](#))
- Plenty of existing plugins and reusable components
- Handles dependencies between components
- Default lifecycles provided for components see [here](#) for details)



Each component has a unique GAV : groupId - artifactId - version

# Use case #2 : Details 3/6

Protocol buffers : data serialization and interface framework (Google)

- generic, technology agnostic description of the data structure

```
<plugin>
  <groupId>com.google.protobuf.tools</groupId>
  <artifactId>maven-protoc-plugin</artifactId>
  <version>0.3.2</version>
  <executions>
    <execution>
      <id>generate-sources</id>
      <goals>
        <goal>compile</goal>
      </goals>
      <phase>generate-sources</phase>
      <configuration>
        <protoSourceRoot>${basedir}/src/main/resources/proto</protoSour
        <includes>
          <param>**/*.proto</param>
        </includes>
      </configuration>
    </execution>
  </executions>
  <configuration>
    <protocExecutable>${PROTOC_HOME}</protocExecutable>
  </configuration>
</plugin>
```

## Use case #2 : Details 4/6

- interface classes generation for each language with protoc compiler
- sources in src/main/resources/proto/messages.proto
- interface classes compiled in target/generated-sources/protobuf/java

```
package org.bluebank.contract;

message ValidateCardRequest {
    optional string cardNumber = 1;
    optional string transactionId =
    optional string id = 3;
    optional string location = 4;
    optional string bankName = 5;
}
...
```

=>

```
// Generated by the protocol buffer compiler
// source: messages.proto

package org.bluebank.contract;

public final class Messages {
    private Messages() {}
    public static void registerAllExtensions(
        com.google.protobuf.ExtensionRegistry
    }
    public interface ValidateCardRequestOrBuilder
        // @@protoc_insertion_point(interface_
        com.google.protobuf.MessageOrBuilder {

        /**
         * <code>optional string cardNumber = 1;
         */
        boolean hasCardNumber();
    ...
}
```

COMPILER MUST BE INSTALLED &  
AVAILABLE IN THE PATH !

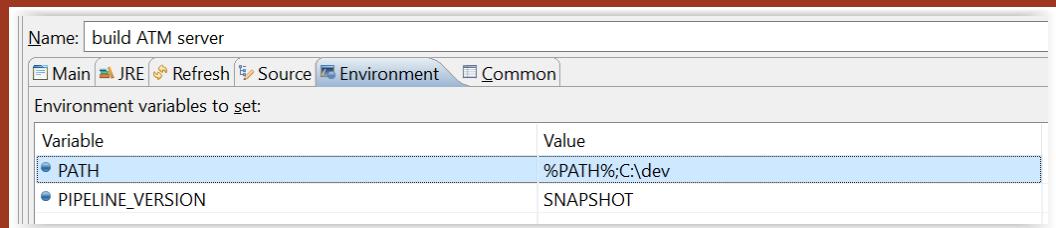
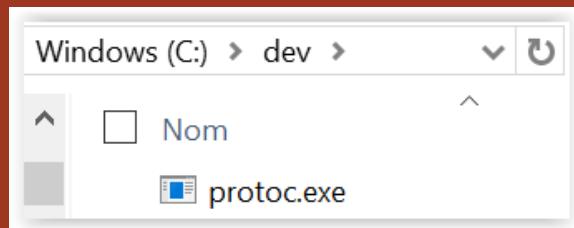
# Use case #2 : Details 5/6

Here, an enforcer requires an environment variable to build version  
(unclean but useful for next parts)

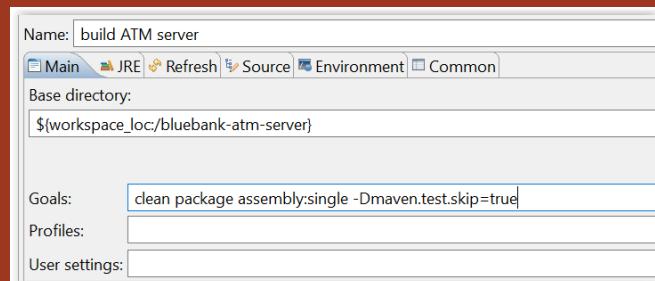
```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-enforcer-plugin</artifactId>
  <version>1.4.1</version>
  <executions>
    <execution>
      <id>enforce-property</id>
      <goals>
        <goal>enforce</goal>
      </goals>
      <configuration>
        <rules>
          <requireProperty>
            <property>project.version</property>
            <message>"Project version must be specified."</message>
            <regex>.*(\d|-SNAPSHOT)$</regex>
            <regexMessage>
              "Please set the PIPELINE_VERSION environment variable (number or -SNAPS
            </regexMessage>
          </requireProperty>
        </rules>
        <fail>true</fail>
      </configuration>
    </execution>
  </executions>
</plugin>
```

# Use case #2 : Details 6/6

Here is a sample configuration to **build** and package the server/jar (Eclipse / Windows)



Maven goals : clean package assembly:single -Dmaven.test.skip=true



```
:terminated> build ATM server [Maven Build] C:\Program Files\Java\jdk1.8.0_65\bin\javaw.exe (23 janv. 2016 23:39:01)
[WARNING] /C:/Users/sguclu/git/sguclu/bluebank-atm-server/src/main/java/org/bluebank/api/domain/AggregateRoot.java:
[INFO]
[INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @ bluebank-atm ---
[INFO] Not copying test resources
[INFO]
[INFO] --- maven-compiler-plugin:3.1:testCompile (default-testCompile) @ bluebank-atm ---
[INFO] Not compiling test sources
[INFO]
[INFO] --- maven-surefire-plugin:2.19:test (default-test) @ bluebank-atm ---
[INFO] Tests are skipped.
[INFO]
[INFO] --- jacoco-maven-plugin:0.7.4.201502261218:report (post-unit-test) @ bluebank-atm ---
[INFO] Skipping JaCoCo execution due to missing execution data file:C:\Users\sguclu\git\sguclu\bluebank-atm-server\
[INFO]
[INFO] --- maven-jar-plugin:2.4:jar (default-jar) @ bluebank-atm ---
[INFO] Building jar: C:\Users\sguclu\git\sguclu\bluebank-atm-server\target\bluebank-atm.jar
[INFO]
[INFO] --- maven-assembly-plugin:2.2-beta-5:single (default-cli) @ bluebank-atm ---
[INFO] Reading assembly descriptor: src/assembly/jar-with-dependencies.xml
[INFO] Building zip: C:\Users\sguclu\git\sguclu\bluebank-atm-server\target\bluebank-atm.zip
[INFO]
[INFO] -----
[INFO] BUILD SUCCESS
[INFO]
[INFO] -----
[INFO] Total time: 31.041 s
[INFO] Finished at: 2016-01-23T23:39:33+01:00
[INFO] Final Memory: 35M/389M
[INFO] -----
```

# Use case #3 : Analyze locally

This part must be performed on a local workstation IDE

- Run unit tests (JUnit)
- Run integration tests (failsafe)
- Run code coverage tests (jacoco)
- Run BDD scenarios (jgiven)
- Run mutation tests (pitest)
- Generate all reports for review

# Use case #3 : tooltips

All test frameworks are available as maven dependencies

Unit tests call BDD/TDD scenarios ([Jgiven](#)/[Junit](#) + [mockito](#))

Integration tests call a [jetty](#) server ([failsafe](#))

A profile called "all\_tests" activates unit and integration tests

Code coverage must be called explicitly ([jacoco](#))

Mutation testing must be called explicitly ([pitest](#))

# Use case #3 : Details 1/2

Using same running config as for build, full analysis can be run:

```
clean verify -Pall-tests jacoco:report org.pitest:pitest-maven:mutationCoverage
```

Warning : as you will see, all tests cannot apply to all the code

Unit tests results (raw) : target/surefire-tests

```
1-----
2Test set: org.bluebank.atm.AtmHealthCheckTest
3-----
4Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.103 sec - in org.bluebank.atm.AtmHealthCheckTest
5
```

Integration tests (raw) : target/failsafe-tests

```
1-----
2Test set: org.bluebank.atm.AtmResourceIT
3-----
4Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 6.883 sec - in org.bluebank.atm.AtmResourceIT
5
```

# Use case #3 : Details 2/2

BDD HTML report in target/jgiven-reports/html

Coverage HTML reports in target/site/jacoco

Mutation testing reports in target/pit-reports

JGiven Report

## All Scenarios

5 Successful, 0 Failed, 0 Pending, 5 Total (1.159s)

Group By Sort By Status Tags Classes

Request Authorization Customer enters a wrong pin number (73ms)  
Given a card with a pin number of "0000" (16ms)  
And the card is inserted (10ms)  
When the wrong pin number "1234" is entered (4ms)  
Then the atm should displayed the message "Wrong PIN" (3ms)

Request Authorization Customer enters wrong pins numbers three times in a row (65ms)

Transaction Balance inquiry (3ms)

Deposit Deposit cash into account (67ms)

Transaction Withdraw cash from account (260ms)

Bluebank ATM Server

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cov.	Missed	Lines	Missed
org.bluebank.contract	23 %	78 %	13 %	87 %	701	638	1 415	1 882	195
org.bluebank	0 %	100 %	0 %	100 %	360	360	564	564	195
org.bluebank.resource	20 %	80 %	40 %	60 %	49	58	82	108	26
org.bluebank.atm	89 %	11 %	33 %	66 %	30	35	95	99	99
org.bluebank.atm.authorization.outbound	87 %	13 %	40 %	60 %	41	61	25	12	12
org.bluebank.api	0 %	100 %	0 %	100 %	17	17	29	29	12
org.bluebank.api.domain	27 %	72 %	0 %	24 %	31	31	49	49	21
org.bluebank.banking.transaction.outbound	92 %	6 %	46 %	42 %	12	13	103	103	0
org.bluebank.banking.command.service	0 %	100 %	0 %	100 %	12	12	17	17	6
org.bluebank.alm.transaction.inbound	87 %	13 %	50 %	30 %	66	0	100	0	0
org.bluebank.alm.transaction.command.factory	86 %	14 %	50 %	30 %	66	0	84	0	0
org.bluebank.alm.authorization.outbound	48 %	51 %	12 %	88 %	8	12	24	24	0
org.bluebank.alm.authorization.inbound	86 %	14 %	50 %	20 %	44	0	56	0	0
org.bluebank.alm.authorization.command.factory	87 %	13 %	50 %	15 %	33	0	55	0	0
org.bluebank.banking.transaction.outbound	87 %	13 %	50 %	15 %	33	0	48	0	0
org.bluebank.banking.transaction.command.factory	89 %	10 %	50 %	16 %	33	0	42	0	0
org.bluebank.api.configuration	0 %	100 %	n/a	2	2	10	10	2	2
org.bluebank.banking.account.model	93 %	7 %	71 %	7	44	7	101	3	3
org.bluebank.banking.authorization.outbound	50 %	50 %	13 %	23	0	46	0	0	0
org.bluebank.banking.authorization.command.factory	50 %	50 %	9 %	19	0	29	0	0	0
org.bluebank.banking.authorization.inbound	86 %	14 %	50 %	5	11	0	14	0	0
org.bluebank.banking.authorization.command	86 %	14 %	50 %	5	11	0	14	0	0
org.bluebank.banking.transaction.model	91 %	9 %	n/a	2	9	0	21	2	2
org.bluebank.banking.transaction.outbound	89 %	11 %	50 %	5	9	0	14	0	0
org.bluebank.banking.persistence	85 %	15 %	n/a	1	4	1	6	1	1
org.bluebank.atm.transaction.command	100 %	n/a	n/a	12	0	34	0	0	0
org.bluebank.api.command	100 %	n/a	100 %	0	6	0	29	0	0
org.bluebank.authorization.transaction.command	100 %	n/a	n/a	0	0	21	0	0	0
org.bluebank.banking.transaction.command	100 %	n/a	n/a	0	6	0	17	0	0
org.bluebank.authorization.command	100 %	n/a	n/a	0	2	0	6	0	0
org.bluebank.banking	100 %	n/a	0	2	0	6	0	0	0

Pit Test Coverage Report

### Project Summary

Number of Classes	Line Coverage	Mutation Coverage
148	38% [36/96] 20%	31% [15/48]

### Breakdown by Package

Name	Number of Classes	Line Coverage	Mutation Coverage
org.bluebank	32	0% [0/32]	0% [0/19]
org.bluebank.api	2	0% [0/2]	0% [0/6]
org.bluebank.api.bus	2	100% [14/14]	100% [4/4]
org.bluebank.api.command	2	100% [29/29]	100% [8/8]
org.bluebank.api.command.processor	2	50% [12/24]	40% [2/5]
org.bluebank.api.command.service	2	0% [0/17]	0% [0/5]
org.bluebank.api.configuration	1	0% [0/10]	0% [0/3]
org.bluebank.api.domain	5	31% [14/45]	15% [3/20]
org.bluebank.api.persistence	1	83% [5/6]	50% [1/2]
org.bluebank.api.atm	5	68% [64/94]	51% [19/37]
org.bluebank.atm.authorization.command	4	100% [23/23]	100% [4/4]
org.bluebank.atm.authorization.command.factory	8	100% [56/56]	100% [12/12]
org.bluebank.atm.authorization.inbound	3	100% [27/27]	100% [6/6]
org.bluebank.atm.authorization.outbound	10	75% [77/102]	52% [11/21]
org.bluebank.atm.transaction.command	6	100% [34/34]	100% [6/6]
org.bluebank.atm.transaction.command.factory	12	100% [84/84]	100% [18/18]
org.bluebank.atm.transaction.inbound	6	100% [54/54]	100% [12/12]
org.bluebank.atm.transaction.outbound	8	90% [115/128]	74% [17/23]
org.bluebank.banking	1	100% [17/17]	100% [2/2]
org.bluebank.banking.account.command	1	100% [6/6]	100% [1/1]

# Use case #4 : Run ATM locally

This part must be performed on a local workstation IDE

Make the application run locally for review

- ATM server
- HTTP client

## Use case #4 : Tooltips

The server part can be run as a standalone application locally by :

- running the main class
- running the jar (requires classpath setup)

The UI part (find out where it is located) requires an HTTP server

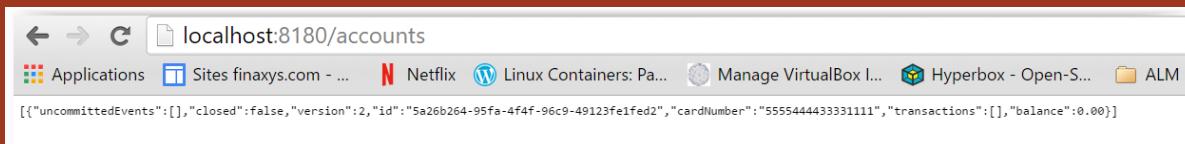
# Use case #4 : Details 1/2

Here let's just run the main class as-is, jar run will be used later ...

The main class to run is org.bluebank.Application

```
run ATM server [Java Application] C:\Program Files\Java\jre1.8.0_65\bin\javaw.exe (24 janv. 2016 21:41:14)
21:41:16.850 [main] DEBUG o.e.j.u.component.AbstractLifeCycle - STARTED @2026ms o.e.j.s.ServletContextHandler@4b5d6a01{/null,AVAILABLE}
21:41:16.850 [main] DEBUG o.e.j.u.component.AbstractLifeCycle - starting ServerConnector@4445629[HTTP/1.1,[http/1.1]{0.0.0.0:8180}
21:41:16.860 [main] DEBUG o.e.j.u.component.ContainerLifeCycle - ServerConnector@4445629[HTTP/1.1,[http/1.1]{0.0.0.0:8180} added (sun.nio.ch.ServerSocketChann
21:41:16.861 [main] DEBUG o.e.j.u.component.AbstractLifeCycle - starting org.eclipse.jetty.util.thread.ScheduledExecutorScheduler@33afa13b
21:41:16.861 [main] DEBUG o.e.j.u.component.AbstractLifeCycle - STARTED @2038ms org.eclipse.jetty.util.thread.ScheduledExecutorScheduler@33afa13b
21:41:16.861 [main] DEBUG o.e.j.u.component.AbstractLifeCycle - starting HttpConnectionFactory@fe20588[HTTP/1.1]
21:41:16.861 [main] DEBUG o.e.j.u.component.AbstractLifeCycle - STARTED @2038ms HttpConnectionFactory@fe20588[HTTP/1.1]
21:41:16.861 [main] DEBUG o.e.j.u.component.AbstractLifeCycle - starting org.eclipse.jetty.server.ServerConnector$ServerConnectorManager@54c562f7
21:41:16.866 [main] DEBUG o.e.j.u.component.AbstractLifeCycle - starting org.eclipse.jetty.io.ManagedSelector@251f7d26 id=0 keys=-1 selected=-1
21:41:16.881 [main] DEBUG o.e.j.u.component.AbstractLifeCycle - STARTED @2057ms org.eclipse.jetty.io.ManagedSelector@251f7d26 id=0 keys=0 selected=0
21:41:16.881 [main] DEBUG o.e.j.u.component.AbstractLifeCycle - queue org.eclipse.jetty.io.ManagedSelector@251f7d26 id=0 keys=0 selected=0
21:41:16.881 [main] DEBUG o.e.j.u.component.AbstractLifeCycle - starting org.eclipse.jetty.io.ManagedSelector@cf9fb3 id=1 keys=-1 selected=-1
21:41:16.882 [qtp750468423-15] DEBUG o.e.j.util.thread.QueuedThreadPool - run org.eclipse.jetty.io.ManagedSelector@251f7d26 id=0 keys=0 selected=0
21:41:16.882 [qtp750468423-15] DEBUG o.e.j.u.util.thread.QueuedThreadPool - run org.eclipse.jetty.io.ManagedSelector@251f7d26 id=0 keys=0 selected=0
21:41:16.882 [qtp750468423-15] DEBUG o.e.j.u.t.s.ExecuteProduceConsume - EPR Idle/org.eclipse.jetty.io.ManagedSelectors$SelectorProducer@7dce6799 execute
21:41:16.882 [qtp750468423-15] DEBUG o.e.j.u.t.s.ExecuteProduceConsume - EPR Prod/org.eclipse.jetty.io.ManagedSelectors$SelectorProducer@7dce6799 produce enter
21:41:16.882 [qtp750468423-15] DEBUG o.e.j.u.t.s.ExecuteProduceConsume - EPR Prod/org.eclipse.jetty.io.ManagedSelectors$SelectorProducer@7dce6799 producing
21:41:16.882 [qtp750468423-15] DEBUG org.eclipse.jetty.io.ManagedSelector - Selector loop waiting on select
21:41:16.882 [main] DEBUG o.e.j.u.component.AbstractLifeCycle - STARTED @2058ms org.eclipse.jetty.io.ManagedSelector@cf9fb3 id=1 keys=0 selected=0
21:41:16.882 [main] DEBUG o.e.j.util.thread.QueuedThreadPool - queue org.eclipse.jetty.io.ManagedSelector@cf9fb3 id=1 keys=0 selected=0
21:41:16.883 [main] DEBUG o.e.j.u.component.AbstractLifeCycle - STARTED @2059ms org.eclipse.jetty.server.ServerConnector$ServerConnectorManager@54c562f7
21:41:16.883 [qtp750468423-14] DEBUG o.e.j.util.thread.QueuedThreadPool - run org.eclipse.jetty.io.ManagedSelector@cf9fb3 id=1 keys=0 selected=0
21:41:16.883 [qtp750468423-14] DEBUG o.e.j.u.t.s.ExecuteProduceConsume - EPR Idle/org.eclipse.jetty.io.ManagedSelectors$SelectorProducer@ffeaade execute
21:41:16.883 [qtp750468423-14] DEBUG o.e.j.u.t.s.ExecuteProduceConsume - EPR Prod/org.eclipse.jetty.io.ManagedSelectors$SelectorProducer@ffeaade produce enter
21:41:16.883 [qtp750468423-14] DEBUG o.e.j.u.t.s.ExecuteProduceConsume - EPR Prod/org.eclipse.jetty.io.ManagedSelectors$SelectorProducer@ffeaade producing
21:41:16.883 [qtp750468423-14] DEBUG org.eclipse.jetty.io.ManagedSelector - Selector loop waiting on select
21:41:16.884 [main] DEBUG o.e.j.u.component.ContainerLifeCycle - ServerConnector@4445629[HTTP/1.1,[http/1.1]{0.0.0.0:8180} added (acceptor-0@4593ff34,POJO)
21:41:16.884 [main] DEBUG o.e.j.util.thread.QueuedThreadPool - queue acceptor-0@4593ff34
21:41:16.884 [qtp750468423-16] DEBUG o.e.j.util.thread.QueuedThreadPool - run acceptor-0@4593ff34
21:41:16.884 [main] INFO o.e.jt.server.ServerConnector - Started ServerConnector@4445629[HTTP/1.1,[http/1.1]{0.0.0.0:8180}
21:41:16.884 [main] DEBUG o.e.j.u.component.AbstractLifeCycle - STARTED @2060ms ServerConnector@4445629[HTTP/1.1,[http/1.1]{0.0.0.0:8180}
21:41:16.885 [main] INFO org.eclipse.jetty.server.Server - Started @2061ms
21:41:16.885 [main] DEBUG o.e.j.u.component.AbstractLifeCycle - STARTED @2061ms org.eclipse.jetty.server.Server@402e37bc
```

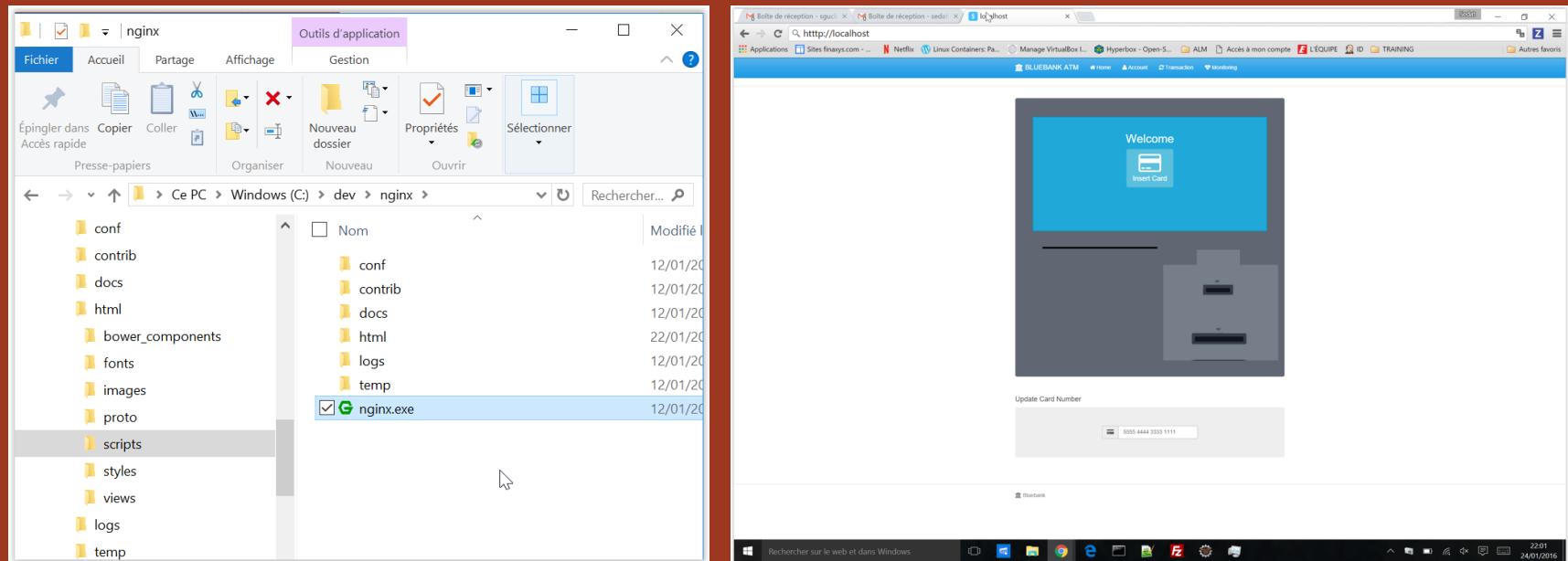
The server listens on port 8180, e.g. <http://localhost:8180/accounts>



# Use case #4 : Details 2/2

For static site, the static pages are located in node-client

A server like [nginx](#) is required to display pages (copy the content of node-client directly in *html* folder)



Play a bit to understand the design and features

# Use case #5 : Install a software

factory must be performed on the virtual machine

Install and run a software factory including:

- [jenkins](#) : continuous integration engine version (jenkins:1.625.3)
- [nexus](#) : binary repository management tool (sonatype/nexus:2.11.2-06)
- [sonarqube](#) : code quality analysis, version (sonarqube:5.2)
- [elasticsearch](#)/[logstash](#)/[kibana](#) : metrology (sebp/elk:E1L1K4)

... using [docker](#) and [docker-compose](#)

# Use case #5 : Tooltips 1/2

Docker is #1 container management solution

- Strong alternative to virtual machines (lighter, faster, better)
- Allows custom image build/commit, or reuse/override existing ones
- Can handle almost any kind of components/applications

First, an account on <https://hub.docker.com/> will greatly help ...

# Use case #5 : Tooltips 2/2

Docker-compose runs on top of docker

- Allows composition of containers (a.k.a. services)
- Manages services start/stop order based on services dependencies
- Handles network link and naming resolution between services
- Supports custom build and out-of-the box images as services

# Use case #5 : Details 1/10

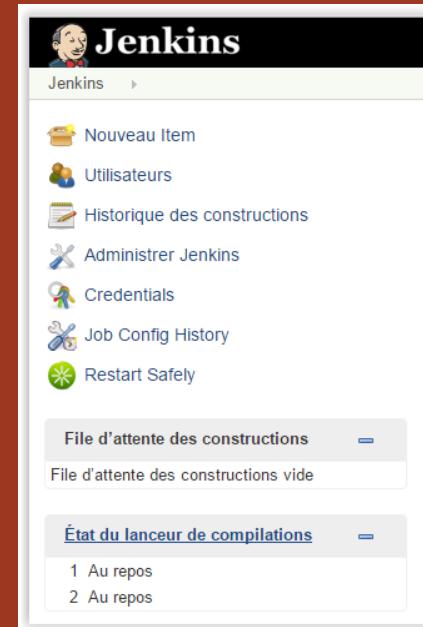
- On the virtual machine, connect to the docker hub

```
docker login
```

- Try to run a Docker image containing Jenkins official image (<https://github.com/jenkinsci/docker>)

```
docker run -p 8080:8080 jenkins:1.625.
```

- After a while, you should have this on [http://<your\\_vm\\_address>:8080](http://<your_vm_address>:8080) :



- What happened there ?

# Use case #5 : Details 2/10

- Docker downloaded the latest Jenkins image from <https://hub.docker.com/>

```
Unable to find image 'jenkins:latest' locally
latest: Pulling from library/jenkins
523ef1d23f22: Extracting 25.17 MB/51.35 MB
140f9bdf97: Download complete
5c63804eac90: Download complete
ce2b29af7753: Download complete
4ee671494b6b: Download complete
```

- Then Docker ran a container based on this image listening on http port 8080 (jenkins port mapped with the 8080 port of your VM)

And voila ... You can check the running container using:

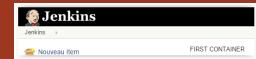
```
docker ps
```

```
[traineegrp@formation ~]$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS
cale9611433c        jenkins            "/bin/tini -- /usr/lo"   6 minutes ago     Up 6 minutes      50000/tcp, 0.0.0.0:5432->8080/tcp
                                                               NAMES
                                                               cranky_brahmagupta
```

WARNING : data belong to the running containers, not the images!

- Add a description (FIRST CONTAINER) on Jenkins:

[Ajouter une description](#)



# Use case #5 : Details 3/10

- Stop the container, (ctrl + c) then launch the Docker run command again : description disappeared from the UI!



We just ran a new container with its own storage...

```
docker ps -a | grep jenki
```

```
xraineegrp@formation ~]$ docker ps -a | grep jenkins
fale9611435c    jenkins      "/bin/tini -- /usr/lo"  About an hour ago   Up About an hour          50000/tcp, 0.0.0.0:5432->8080/tcp
/c3cac9be7d8    jenkins      "/bin/tini -- /usr/lo"  About an hour ago   Exited (130) About an hour ago
                                                               cranky_brahmagupta
                                                               fervent_boyd
```

- Now, stop this new container and run the previous one using ids

```
docker stop <new_container_id> ; docker start <old_container_id>
```

Message is back ... because it was saved in the first container created

# Use case #5 : Details 4/10

Containers persist data in the container by default, but what if we really need to keep the data ?

=> We need to "export" the datas in a volume on the host server

- Stop the container, and run it again with the volume mount option

```
docker stop <old_container_id>
sudo mkdir -p /volumes/jenkins
sudo chown 1000:1000 /volumes/jenkins
docker run -p 8080:8080 -v /volumes/jenkins:/var/jenkins_home jenkins:1.625.3
```

In this new container, var/jenkins\_home folder (jenkins repository ) is now hosted on the /volumes/jenkins folder of the virtual machine

=> persists across container restart/removals

=> take care of concurrent accesses through multiple containers!

# Use case #5 : Details 5/10

- In the same way, start a Sonarqube container with these properties:
    - port 9000 : HTTP port of the user interface
    - port 9092 : JDBC port of the internal database
- /volumes/sonar/data:/opt/sonarqube/data (internal database)  
/volumes/sonar/extensions:/opt/sonarqube/extensions (plugins)

```
sudo mkdir -p /volumes/sonar/data /volumes/sonar/extensions
sudo chown -R 1000:1000 /volumes/sonar
docker run -p 9000:9000 -p 9092:9092 \
-v /volumes/sonar/data:/opt/sonarqube/data \
-v /volumes/sonar/extensions:/opt/sonarqube/extensions \
sonarqube:5.2
```

The screenshot shows the SonarQube web interface. At the top, there's a navigation bar with links for Dashboards, Issues, Measures, Rules, Quality Profiles, Quality Gates, More, Log in, and a search icon. Below the navigation is a "Home" section with a "Welcome to SonarQube Dashboard" message. It says: "Since you are able to read this, it means that you have successfully started your SonarQube server. Well done! If you have not removed this text, it also means that you have not yet played much with SonarQube. So here are a few pointers for your next step:" followed by a list of links: "run analysis on a project?", "customizing dashboards?", "complete documentation?", and "Get Support page.". To the right of this are two tables under the heading "PROJECTS". The first table has columns: QG NAME, VERSION, LOC, TECHNICAL DEBT, and LAST ANALYSIS. It shows "No data". The second table also shows "No data".

# Use case #5 : Details 6/10

- Then, start a Nexus container with these properties:
  - port 8081 : HTTP port of the user interface
  - /volumes/nexus:/sonatype-work (internal repository)

```
sudo mkdir /volumes/nexus
sudo chown 200:200 /volumes/nexus
docker run -p 8081:8081 \
-v /volumes/nexus:/sonatype-work \
sonatype/nexus:2.11.2-06
```

The screenshot shows the SonarQube web interface. At the top, there is a navigation bar with links for Dashboards, Issues, Measures, Rules, Quality Profiles, Quality Gates, More, Log in, and a search bar. Below the navigation bar, the page title is "Home". A prominent message reads: "Welcome to SonarQube Dashboard. Since you are able to read this, it means that you have successfully started your SonarQube server. Well done! If you have not removed this text, it also means that you have not yet played much with SonarQube. So here are a few pointers for your next step: » Do you now want to [run analysis](#) on a project? » Maybe start [customizing dashboards](#)? » Or simply browse the [complete documentation](#)? » If you have a question or an issue, please visit the [Get Support](#) page."

On the right side of the screen, there are two sections titled "PROJECTS". The first section has a header with columns: QG NAME ▲, VERSION, LOC, TECHNICAL DEBT, and LAST ANALYSIS. It contains one row with the status "No data". The second section also has a header with the same columns and contains one row with the status "No data".

# Use case #5 : Details 7/10

- Finally, run an Elastic/Kibana container with these properties:  
port 9200, 9300 : Elasticsearch ports  
port 5601 : Kibana user interface  
/volumes/elk:/var/lib/elasticsearch (internal repository)

```
sudo mkdir /volumes/elk
sudo chown 103:107 /volumes/elk
docker run -p 9200:9200 -p 9300:9300 -p 5601:5601 \
-v /volumes/elk:/var/lib/elasticsearch \
sebp/elk:E1L1K4
```

The screenshot shows the SonarQube dashboard. At the top, there is a navigation bar with links for Dashboards, Issues, Measures, Rules, Quality Profiles, Quality Gates, More, Log in, and a search bar. Below the navigation bar, the page title is "Home". On the left, there is a sidebar with a "Welcome to SonarQube Dashboard" message and a list of initial steps: "Do you now want to run analysis on a project?", "Maybe start customizing dashboards?", "Or simply browse the complete documentation?", and "If you have a question or an issue, please visit the Get Support page.". The main content area has two sections: "PROJECTS" and "PROJECTS". Both sections are currently empty, showing the message "No data".

# Use case #5 : Details 8/10

It works fine, but using docker-compose:

- everything is started in the proper order
- ports/volumes are handled automatically
- using named containers, services can talk to each other
- existing, unchanged service is started only once
- Stop all the containers, then edit a file called [docker-compose.yml](#)

# Use case #5 : details 9/10

- Put the following content in it, then save (scroll!):

```
jenkins:  
  image : "jenkins:1.625.3"  
  ports:  
    - "8080:8080"  
  volumes:  
    - /volumes/jenkins:/var/jenkins_home  
  container_name: traineegrp-jenkins  
  links:  
    - sonar  
    - nexus  
    - elk  
  
nexus:  
  image: "sonatype/nexus:2.11.2-06"  
  ports:  
    - "8081:8081"  
  volumes:  
    - /volumes/nexus:/sonatype-work  
  container_name: traineegrp-nexus  
  links:  
    - elk  
  
sonar:  
  image: "sonarqube:5.2"  
  ports:
```

# Use case #5 : details 10/10

- Run now this command and check what is going on : docker-compose up

Each service is launched as expected in the proper order

```
traineegrp@formation FACTORY_SETUP]$ docker-compose up
Starting traineegrp-elk
Starting traineegrp-sonarqube
Starting traineegrp-nexus
Starting traineegrp-jenkins
Attaching to traineegrp-jenkins, traineegrp-sonarqube, traineegrp-nexus, traineegrp-jenkins
traineegrp-sonarqube 2016-01-26 00:24:04 INFO app[.o.s.m.JavaProcessLauncher] Launch process[search]: /usr/lib/jvm/java-8-openjdk-amd64/jre/bin/java -Djava.awt.headless=true -Xmx1G -Xms256m -Xss256K -Djava.net.preferIPv4Stack=true -XX:+UseParNewGC -XX:+CMSInitiatingOccupancyFraction=75 -XX:+UseCMSInitiatingOccupancyOnly -XX:+HeapDumpOnOutOfMemoryError -Djava.io.tmpdir=/opt/sonarqube/temp -cp ./lib/common/*./lib/search/* org.sonar.search.SearchServer /tmp/sq-process3283018939097733594properties
traineegrp-elk      | Starting Elasticsearch Server
traineegrp-elk      | systemctl start elasticsearch
traineegrp-elk      | logstash started
traineegrp-elk      | waiting for Elasticsearch to be up (1/30)
traineegrp-nexus    | 2016-01-26 00:24:04,290+0000 INFO [main] *SYSTEM org.sonatype.nexus.bootstrap.ConfigurationBuilder - Properties:
traineegrp-nexus    | 2016-01-26 00:24:04,299+0000 INFO [main] *SYSTEM org.sonatype.nexus.bootstrap.ConfigurationBuilder - application-conf='/sonatype-work/conf'
traineegrp-nexus    | 2016-01-26 00:24:04,300+0000 INFO [main] *SYSTEM org.sonatype.nexus.bootstrap.ConfigurationBuilder - application-host='10.0.0.0'
traineegrp-nexus    | 2016-01-26 00:24:04,301+0000 INFO [main] *SYSTEM org.sonatype.nexus.bootstrap.ConfigurationBuilder - application-port='8081'
traineegrp-nexus    | 2016-01-26 00:24:04,301+0000 INFO [main] *SYSTEM org.sonatype.nexus.bootstrap.ConfigurationBuilder - bundleBasedir='/opt/sonatype/nexus'
traineegrp-nexus    | 2016-01-26 00:24:04,301+0000 INFO [main] *SYSTEM org.sonatype.nexus.bootstrap.ConfigurationBuilder - java.awt.headless='true'
traineegrp-nexus    | 2016-01-26 00:24:04,302+0000 INFO [main] *SYSTEM org.sonatype.nexus.bootstrap.ConfigurationBuilder - networkAddress.cache.ttl='3600'
traineegrp-nexus    | 2016-01-26 00:24:04,302+0000 INFO [main] *SYSTEM org.sonatype.nexus.bootstrap.ConfigurationBuilder - nexus-baseapp='/opt/sonatype/nexus/nexus/NEXUS-INF'
traineegrp-nexus    | 2016-01-26 00:24:04,302+0000 INFO [main] *SYSTEM org.sonatype.nexus.bootstrap.ConfigurationBuilder - nexus-webapp='/opt/sonatype/nexus/nexus'
traineegrp-nexus    | 2016-01-26 00:24:04,302+0000 INFO [main] *SYSTEM org.sonatype.nexus.bootstrap.ConfigurationBuilder - nexus-work='/sonatype-work'
traineegrp-nexus    | 2016-01-26 00:24:04,302+0000 INFO [main] *SYSTEM org.sonatype.nexus.bootstrap.ConfigurationBuilder - org.eclipse.ecf.provider.filetransfer.retrieve.rea
dTimeout='30000'
traineegrp-nexus    | 2016-01-26 00:24:04,302+0000 INFO [main] *SYSTEM org.sonatype.nexus.bootstrap.ConfigurationBuilder - runTime='/opt/sonatype/nexus/nexus/NEXUS-INF'
traineegrp-nexus    | 2016-01-26 00:24:04,302+0000 INFO [main] *SYSTEM org.sonatype.nexus.bootstrap.ConfigurationBuilder - security-xml-file='/sonatype-work/conf/security.xml'
1'
traineegrp-nexus    | 2016-01-26 00:24:04,303+0000 INFO [main] *SYSTEM org.sonatype.nexus.bootstrap.Launcher - Java: 1.7.0_76, Java HotSpot(TM) 64-Bit Server VM, Oracle Corpor
ation, 24.76-b04
traineegrp-nexus    | 2016-01-26 00:24:04,303+0000 INFO [main] *SYSTEM org.sonatype.nexus.bootstrap.Launcher - OS: Linux, 3.10.0-229.20.1.el7.x86_64, amd64
traineegrp-nexus    | 2016-01-26 00:24:04,303+0000 INFO [main] *SYSTEM org.sonatype.nexus.bootstrap.Launcher - User: nexus, en, /sonatype-work
```

```
[sguclu@formation:~]$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS
 NAMES
545e6533016d        factorysetup_jenkins   "/bin/tini -- /usr/lo"   About a minute ago   Up About a minute   0.0.0.0:8080->8080/tcp, 50000/tcp
traineegrp-jenkins
ed1d592310d6        sonatype/nexus:2.11.2-06  "/bin/sh -c 'java -"   2 minutes ago       Up About a minute   0.0.0.0:8081->8081/tcp
traineegrp-nexus
ffbaba2c7ea5        sonarqube:5.2          "./bin/run.sh"        2 minutes ago       Up 2 minutes       0.0.0.0:9000->9000/tcp, 0.0.0.0:9092->9092/tcp
traineegrp-sonarqube
969481ed9711        sebp/elk:E1L1K4        "/usr/local/bin/start" 2 minutes ago       Up 2 minutes       0.0.0.0:5601->5601/tcp, 5000/tcp, 9300/tcp, 0.0.0.0:9200->9200/t
cp traineegrp-elk
[sguclu@formation:~]$ ~
```

This software factory will be improved later ...

# Use case #6 : Prepare the factory

Jenkins :

- Install tools : [Gatling](#), [Java 8](#), [Maven 3](#), [git client](#), [Protocol Buffers](#)
- install plugins : git, sonar, htmlpublisher, jobConfigHistory, saferestart, pitmutation, envinject, parameterized-trigger, build-pipeline-plugin, rebuild, mask-passwords, logstash, metadata
- Set the sonarqube, logstash and stash configuration in Jenkins
- Allow jenkins to run docker containers on the host

Sonarqube : install Checkstyle, PMD, Findbugs, Github et Pitest plugins

ELK : install the "HEAD" plugin on elasticsearch (indexes browsing)

# Use case #6 : Tooltips 1/2

Sonar plugins have to be installed from the UI

Jenkins plugins can be installed:

- From the UI : convenient but not automated
- From <https://updates.jenkins-ci.org/download/plugins/> and copied to the plugins folder : painful, no plugin dependencies managed
- Managed by jenkins-cli : better but requires post-start commands
- Included in the image build process : requires custom image build

# Use case #6 : Tooltips 2/2

Jenkins tools can be installed:

- Manually in the container, then declared : painful, not versionned
- Auto-installed by jenkins when possible : better but requires post-start configuration on the slave
- Included in the image build process : requires custom image build

# Use case #6 : Details 1/12

Sonarqube :

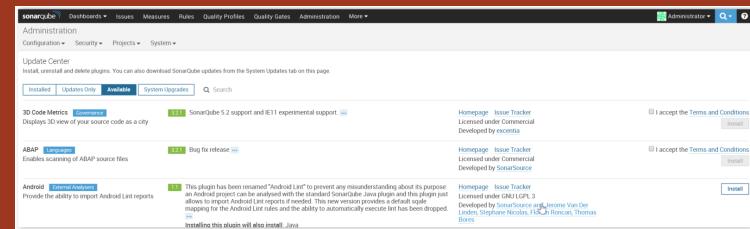
- log on `http://<your_VM>:9000`

user/password : admin / admin

- The available plugins can be installed

from update center : `http://<your_VM>:9000/updatecenter/available`

- Select and install the plugins, then restart the factory



```
docker-compose restart
```

As long as the sonarqube database and extension folders are managed as volumes outside the container, we can consider it is safe to do so

```
[traineegrp@formation vm-formation]$ ls -l /volumes/sonar/extensions/plugins/
total 30072
-rw-r--r--. 1 root      root      5459683 Jan 26 12:37 sonar-checkstyle-plugin-2.4.jar
-rw-r--r--. 1 root      root      7080266 Jan 26 12:37 sonar-findbugs-plugin-3.3.jar
-rw-r--r--. 1 root      root      1601168 Jan 26 12:37 sonar-github-plugin-1.1.jar
-rw-r--r--. 1 root      root      3005657 Jan 26 12:37 sonar-java-plugin-3.9.jar
-rw-r--r--. 1 root      root      31567 Jan 26 12:37 sonar-pitest-plugin-0.6.jar
-rw-r--r--. 1 root      root     4366424 Jan 26 12:37 sonar-pmd-plugin-2.5.jar
-rw-r--r--. 1 lgrangeau lgrangeau 2680676 Oct 12 16:44 sonar-scm-git-plugin-1.0.jar
-rw-r--r--. 1 lgrangeau lgrangeau 6556136 Oct 12 16:44 sonar-scm-svn-plugin-1.2.jar
```

# Use case #6 : Details 2/12

ELK :

- To install head plugin, the command-line is self-sufficient

```
docker exec -ti traineegrp-elk /usr/share/elasticsearch/bin/plugin -install mobz/elastics
```

```
[traineegrp@formation vm-formation]$ docker exec -ti traineegrp-elk /usr/share/elasticsearch/bin/plugin -install mobz/elasticsearch-head
-> Installing mobz/elasticsearch-head...
Trying https://github.com/mobz/elasticsearch-head/archive/master.zip...
Downloaded ...
...
DONE
Installed mobz/elasticsearch-head into /usr/share/elasticsearch/plugins/head
```

This will allow to browse the elasticsearch indexes through a browser (not possible by default!) at [http://<your\\_VM>:9200/\\_plugin/head/](http://<your_VM>:9200/_plugin/head/)

The screenshot shows the Elasticsearch Head plugin interface. At the top, it displays "Santé du cluster: yellow (6 12)". Below this, there's a navigation bar with tabs: "Aperçu", "Index", "Navigateur", "Recherche Structurée [+]", and "Autres requêtes [+]". The "Aperçu du cluster" tab is selected. The main area shows the cluster status with a warning icon and the text "Unassigned". It lists three indices: ".kibana", "jenkins", and "logstash-969481ed9711-36-12028". Each index entry includes an "Info" button and an "Actions" dropdown menu. To the right of the index names, there are green progress bars indicating document processing status. The "jenkins" index has a higher completion rate than the others.

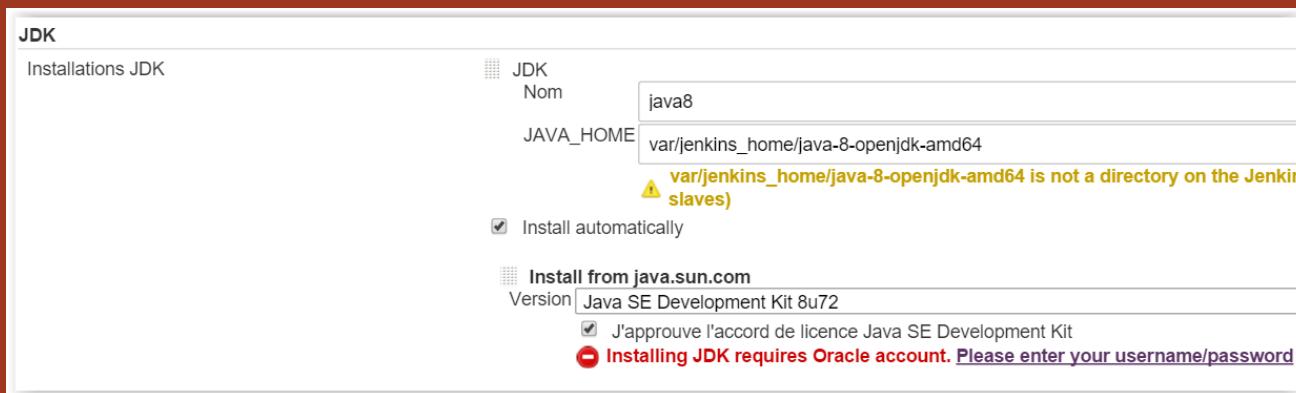
# Use case #6 : Details 3/12

Jenkins:

- install the plugins using the Jenkins CLI

```
docker exec -ti traineegrp-jenkins java \
-jar /var/jenkins_home/war/WEB-INF/jenkins-cli.jar -s http://localhost:8080 install-plugin
git sonar htmlpublisher jobConfigHistory saferestart pitmutation envinject parameterized-t-
build-pipeline-plugin mask-passwords rebuild logstash metadata -restart
```

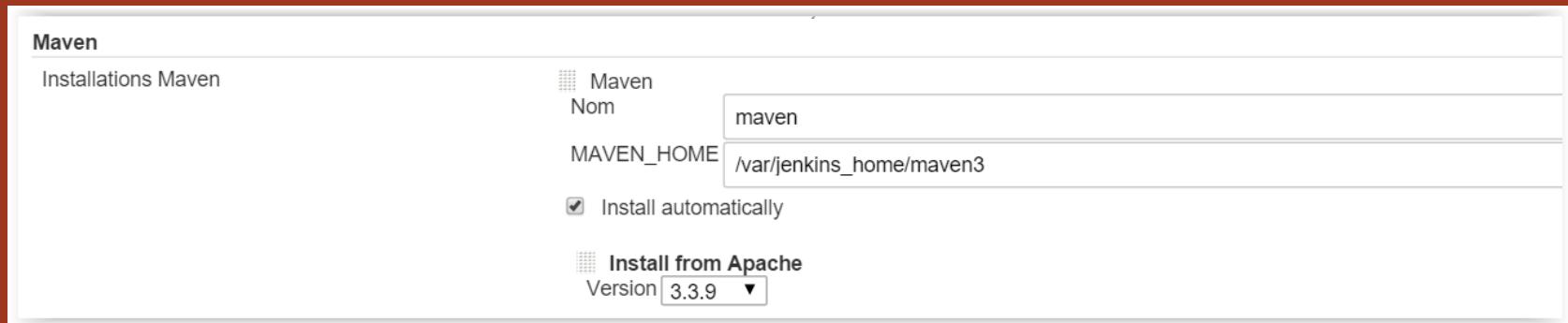
- Java can be "auto-installed" using the appropriate feature in Jenkins



- Oracle registration : <https://login.oracle.com/mysso/signon.jsp>

# Use case #6 : Details 4/12

- Similarly, Maven can be auto-installed

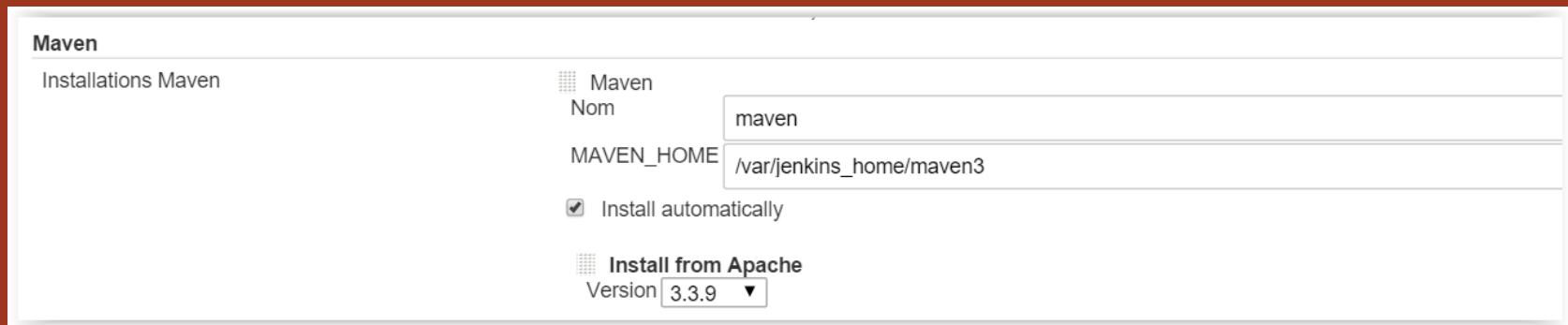


- Protocol buffer cannot be installed as-is : the package must be installed specifically
- The same for Gatling
- Allowing jenkins to run docker means install docker + specific configuration ...

Standard image is NOT enough for that anymore ...

# Use case #6 : Details 5/12

- Similarly, Maven can be auto-installed



- Protocol buffer cannot be installed as-is : the package must be installed specifically
- The same for Gatling
- Allowing jenkins to run docker means install docker + specific configuration ...

Standard image is NOT enough for that anymore ...

# Use case #6 : Details 6/12

- Modify the jenkins definition in docker-compose.yml file as-is:

```
jenkins:  
  build: ./01-custom-jenkins  
  ports:  
    - "8080:8080"  
  volumes:  
    - /volumes/jenkins:/var/jenkins_home  
    - /var/run/docker.sock:/var/run/docker.sock  
  container_name: traineegrp-jenkins  
  privileged: true  
  links:  
    - sonar  
    - nexus  
    - elk
```

Jenkins service will now be built not run from an existing image

The docker.sock share and privileged options allows jenkins to run containers on the host itself instead of inside jenkins itself

# Use case #6 : Details 7/12

- Create a folder 01-custom-jenkins and a file Dockerfile in it:

```
FROM jenkins:1.625.3

# NEED TO BE ROOT FOR APT INSTALLS
USER root

# MODULES INSTALL : GIT, PROTOBUF, PUPPET, SUDO
RUN apt-get update && apt-get install -y git protobuf-compiler apt-transport-https sudo wget curl

# DOCKER REPOS SETUP - YEAH, DOCKER WITH DOCKER ;)
RUN apt-key adv --keyserver hkp://p80.pool.sks-keyservers.net:80 --recv-keys 58118E89F3A912
  echo "deb https://apt.dockerproject.org/repo debian-jessie main" > /etc/apt/sources.list
  apt-get update && apt-cache policy docker-engine

# INSTALL AND ACTIVATE DOCKER
RUN apt-get install -y docker-engine && \
  adduser jenkins sudo && \
  echo "jenkins ALL=NOPASSWD:ALL" | (EDITOR="tee -a" visudo) && usermod -aG docker jenkins

# GATLING INSTALL "A LA MANO"
RUN wget https://oss.sonatype.org/content/repositories/releases/io/gatling/highcharts/gatling-charts-highcharts-bundle/2.1.7/gatling-charts-highcharts-bundle-2.1.7-bundle.zip
  unzip gatling-charts-highcharts-bundle-2.1.7-bundle.zip

# GET BACK TO STANDARD USER
USER jenkins
```

## Use case #6 : Details 8/12

This describes an new Jenkins image overriding official one with:

- additional tools installed with aptitude (ubuntu originated)
- registering of ubuntu packages for docker
- install of the docker-engine allowed for jenkins user as sudo (required to launch containers on the host VM itself)
- Gatling distribution installed

# Use case #6 : Details 9/12

- Now, let's build the factory ...

```
docker-compose build
```

Jenkins image is built from the Dockerfile specification layer per layer

```
[traineegrp@formation FACTORY_SETUP]$ docker-compose build
elk uses an image, skipping
sonar uses an image, skipping
nexus uses an image, skipping
Building jenkins
Step 1 : FROM jenkins:1.625.3
--> eb09f4d99ddf
Step 2 : USER root
--> Using cache
--> 34eafc41cf78
Step 3 : RUN apt-get update && apt-get install -y git protobuf-compiler apt-transport-https sudo wget telnet vim
--> Running in 56835c13fb2a
Get:1 http://security.debian.org jessie/updates InRelease [63.1 kB]
Ign http://httpredir.debian.org jessie InRelease
Get:2 http://httpredir.debian.org jessie-updates InRelease [136 kB]
Get:3 http://httpredir.debian.org jessie-backports InRelease [166 kB]
```

Now, restart the factory : docker-compose knows that Jenkins container is obsolete and runs it instead of restarting old one

```
[traineegrp@formation FACTORY_SETUP]$ docker-compose up
Starting traineegrp-elk
Starting traineegrp-sonarqube
Starting traineegrp-nexus
Recreating traineegrp-jenkins
```

# Use case #6 : Details 10/12

- Configure now the Sonarqube component in Jenkins admin UI

The screenshot shows the Jenkins configuration interface for a SonarQube component. The left sidebar lists sections: SonarQube, Environment variables, and Installations de SonarQube. The main area contains several configuration fields:

- Enable injection of SonarQube server configuration as build environment variables**: A checkbox with a note: "If checked, jobs administrators will be able to inject a SonarQube server configuration as environment variables in the build." Below it is a text input field containing "traineegrp-sonarque".
- URL du serveur**: A text input field containing "http://traineegrp-sonarqube:9000".
- Login du compte SonarQube**: A text input field.
- Mot de passe du compte SonarQube**: A text input field.
- Désactiver**: A checkbox with a note: "Cocher pour rapidement désactiver SonarQube sur tous les jobs." Below it is a text input field containing "Ne pas renseigner si vous utilisez la base embarquée." and a note: "Par défaut à sonar."
- URL de la base de données**: A text input field.
- Nom d'utilisateur BDD**: A text input field.
- Mot de passe BDD**: A text input field.
- Version du sonar-maven-plugin**: A text input field.
- Propriétés additionnelles**: A text input field containing "-Dsonar.junit.mode=reuseReport".
- Additional analysis properties**: A text input field containing "sonar.analysis.mode=issues".
- Conditions d'exécution (seulement pour les projets utilisant SonarQube en action à la suite du build)**: A section with three checkboxes:
  - Ignorer si déclenché par un changement dans l'outil de gestion de version
  - Ignorer si déclenché par un projet amont
  - Ignorer si une variable d'environnement est définie à true

A red button at the bottom right is labeled "Supprimer installation SonarQube".

Thanks to the links, Jenkins accesses Sonarqube using its name, allowing code quality analyses pushed on demand

# Use case #6 : Details 11/12

- Configure now the Logstash component in Jenkins admin UI

The screenshot shows the Jenkins Logstash Plugin configuration interface. It has a header 'Logstash Plugin' and several input fields for configuration:

Setting	Value
Indexer type	ELASTICSEARCH
Host name	http://traineegrp-elk
Port	9200
Username	(empty)
Password	(empty)
Key	jenkins/object

Once again, Jenkins accesses smoothly to ELK container using its name, allowing jobs status reporting in elasticsearch database

# Use case #6 : Details 12/12

- Configure now the Slack component in Jenkins admin UI and test connexion

Take the channel and integration tokens from your stash configuration

Global Slack Notifier Settings

Team Domain	finaxys
Integration Token	<your_TOKEN>
Channel	#<your_CHANNEL>
Build Server URL	http://<your_VM>:8080/

Thanks to the links, Jenkins accesses Slack to notify instantly the build events activated on the jobs (start/stop/failed...)

# Use case #7 : Create QUALIFY job

Create a freestyle job called "QUALIFY" which:

- Sets the value of PIPELINE\_VERSION to internal build number and puts the version information on the application main page
- Builds/tests/packages and deploys the application in Nexus
- Publishes the Junit/Jgiven/Pitest reports in Jenkins UI and Sonarqube
- Pushes a new github tag : ATM SERVER-\${PIPELINE\_VERSION}
- Sends status and reports to ELK and Stash

# Use case #7 : Tooltips

A fully installed and configured software factory is required here...

The best place to put the version is in the page title or in the banner

Build/tests should work in the same way as on your workstation

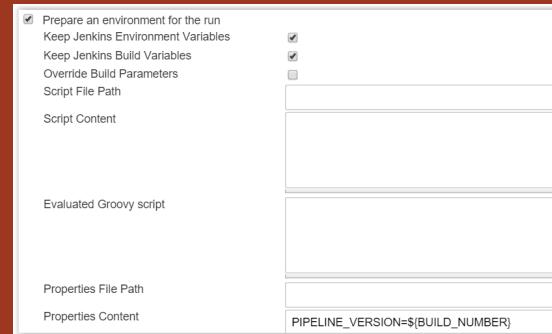
The deployment in Nexus can be done with HTTP connexion parameters to simplify the maven part (with password masking!)

No specific feature for Jgiven publish, use HTML instead (bugged)

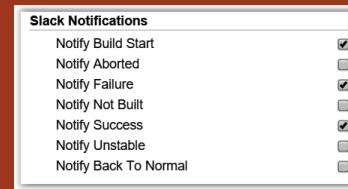
- The tag in github needs configuration to work

# Use case #7 : Details 1/5

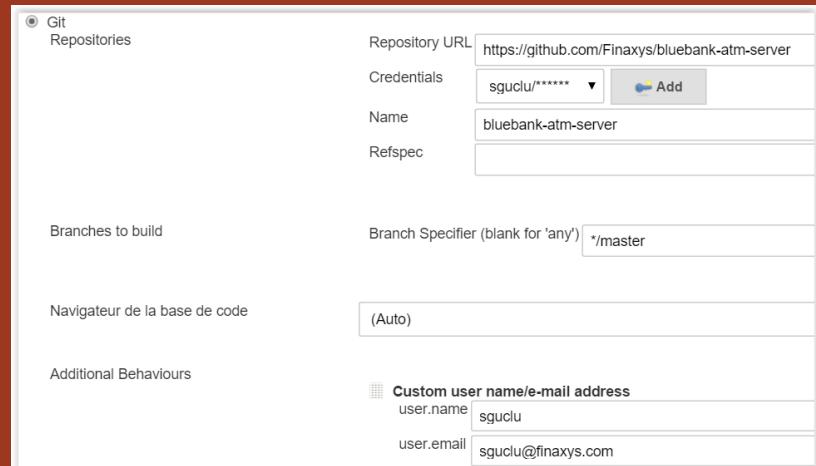
- The pipeline version is configured at the beginning using the Jenkins \${BUILD\_NUMBER} variable



- Slack can be notified in case of a job starts/succeeds/fails

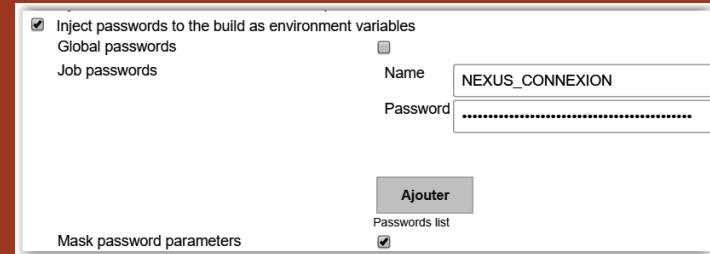


- To push the tag to Github, the SCM configuration requires a proper authentication/authorization on the forked repository URL, and also user name/email for Github checks

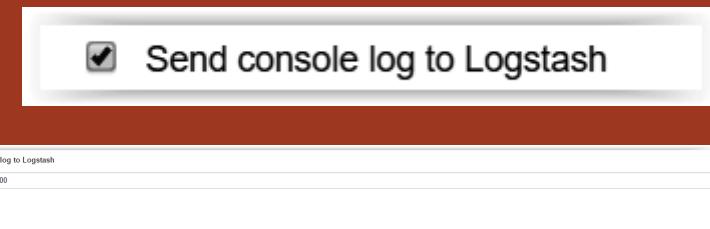


# Use case #7 : Details 2/5

- HTTP connexion to Nexus defaults with "admin:admin123", however to hide this info from logs/UI, we inject and mask it



- Elasticsearch can be notified of all events related to this job  
(don't forget the post-task!)



- Using a shell step, here is a cheap way to inject versions in the application:

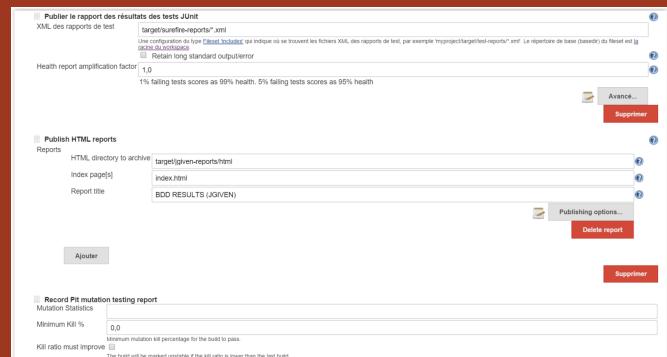
```
sed -i "s|<title></title>|<title>ATM-${PIPELINE_VERSION}</title>|g" node-client/index.html
sed -i "s|BLUEBANK ATM|BLUEBANK ATM v${PIPELINE_VERSION}|g" node-client/index.html
```

# Use case #7 : Details 3/5

- The build is quite similar to the workstation one, except the deployment part using the password obfuscation

```
ean
ckage
sembly:single
ploy
all-tests jacoco:report org.pitest:pitest-maven:mutationCoverage
altDeploymentRepository=releases::default::http://${NEXUS_CONNEXION}@traineegrp-nexus:8081/content/repos
```

- Jenkins reports publish is pretty self-explanatory

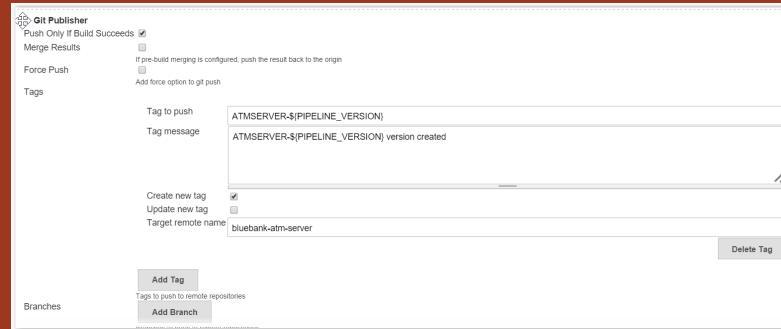


- Add sonarqube invocation



# Use case #7 : Details 4/5

- We push only the tag based on the repository name (no code changed, even the HTML page)



- Nexus URL ([http://<your\\_VM>:8081/content/repositories/releases](http://<your_VM>:8081/content/repositories/releases)) should contain the binaries produced

Index of /repositories/releases/org/bluebank/atm/bluebank-atm/ATM-100			
Name	Last Modified	Size	Description
Parent Directory			
bluebank-atm-ATM-100.jar	Thu Jan 21 22:03:42 UTC 2016	314724	
bluebank-atm-ATM-100.jar.md5	Thu Jan 21 22:03:42 UTC 2016	32	
bluebank-atm-ATM-100.jar.sha1	Thu Jan 21 22:03:42 UTC 2016	40	
bluebank-atm-ATM-100.zip	Thu Jan 21 22:03:42 UTC 2016	14865	
bluebank-atm-ATM-100.zip.md5	Thu Jan 21 22:03:42 UTC 2016	32	
bluebank-atm-ATM-100.zip.sha1	Thu Jan 21 22:03:42 UTC 2016	40	
bluebank-atm-ATM-100.zip	Thu Jan 21 22:03:42 UTC 2016	13571184	
bluebank-atm-ATM-100.zip.md5	Thu Jan 21 22:03:42 UTC 2016	32	
bluebank-atm-ATM-100.zip.sha1	Thu Jan 21 22:03:42 UTC 2016	40	

- When executed, the build/tests must all pass as seen in the reports (for Jgiven, get the zip and open it extracted outside Jenkins)

**Modules**

**Mutation Statistics**

Mutations Undetected	Coverage
3170 (0)	2580 (0) 18.612% (0.0%)

**Components**

Name	Mutations	Undetected	Coverage
Module_0	1565	0	1290 18.612% 0.0%
Module_1	1565	0	1290 18.612% 0.0%

**Résultats des tests**

0 échecs (0)

12 tests (0)  
Avec 0.8 %

**Tous les tests**

Package	Durée	Echec	(err)	Sauté	(err)	Pass	(err)	Total	(err)
org.bluebank.api	13 ms	0	0	0	0	4	0	4	0
org.bluebank.api.command	8 ms	0	0	0	0	1	0	1	0
org.bluebank.arm	7 ms	0	0	0	0	2	0	2	0
org.bluebank.arm.authorization.bdd	0.71 s	0	0	0	0	2	0	2	0
org.bluebank.arm.transaction.bdd	0.16 s	0	0	0	0	3	0	3	0

# Use case #7 : Details 5/5

- On Sonarqube ([http://<your\\_VM>:9000](http://<your_VM>:9000)) you can add the Pitest and Integration tests widgets to have a full status of the code quality  
(N.B. : bug on Pitest reports with Sonarqube 5.2 already known)

The screenshot shows the Sonarqube Main Dashboard with the following data:

- Lines Of Code:** 3 294 (Java: 44 Directories, 4 127 Lines, 128 Classes, 650 Statements, 8 Accessors)
- Duplications:** 0,0%
- Complexity:** 343 (/Function: 1,0 /Class: 2,7 /File: 3,6)
- Events:** All (26 jan. 2016, 26 jan. 2016)
- SQALE Rating:** A
- Technical Debt Ratio:** 0,3%
- Debt:** 4h 46min
- Issues:** 36 (Blocker: 0, Critical: 1, Major: 27, Minor: 8, Info: 0)
- Unit Tests Coverage:** 69,9% (Line Coverage: 70,8%, Condition Coverage: 44,7%)
- Integration Tests Coverage:** 71,8% (Line Coverage: 73,0%, Condition Coverage: 39,5%)
- Overall Coverage:** 92,0% (Line Coverage: 92,9%, Condition Coverage: 65,8%)
- Unit Test Success:** 100,0% (Failures: 0, Errors: 0, Tests: 12, Execution Time: 906 ms)

At the bottom left, it says: Bluebank ATM Server org.bluebank.atm.bluebank-atm Bluebank ATM Services Profiles: Sonar way (Java) Quality Gate: SonarQube way (Default)

# Use case #8 : Create PACKAGE job

Create a job called "PACKAGE" which:

- Run only if the "QUALIFY" passed, and use the same source/version
- Creates a docker container derived from "codenvy/debian\_jdk8"
- Installs the tools required to run the ATM server (JVM) and client (web site)
- Retrieves, installs and run the application packaged from Nexus
- Sends status and reports to ELK and Slack

## Use case #8 : Tooltips

The SCM, Logstash and Slack configuration are the same

A link between both jobs passing the PIPELINE\_VERSION is required

To retrieve the binary, a "simple" wget is already present in the container

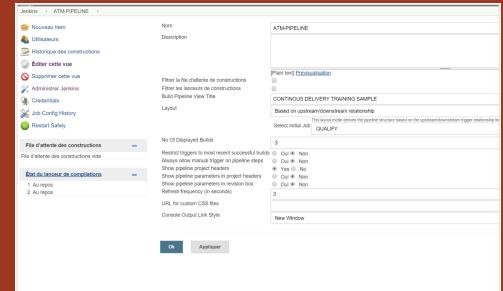
An HTTP client such a Nginx to install implies a custom Dockerfile

To run the container processes, we will need an entry point

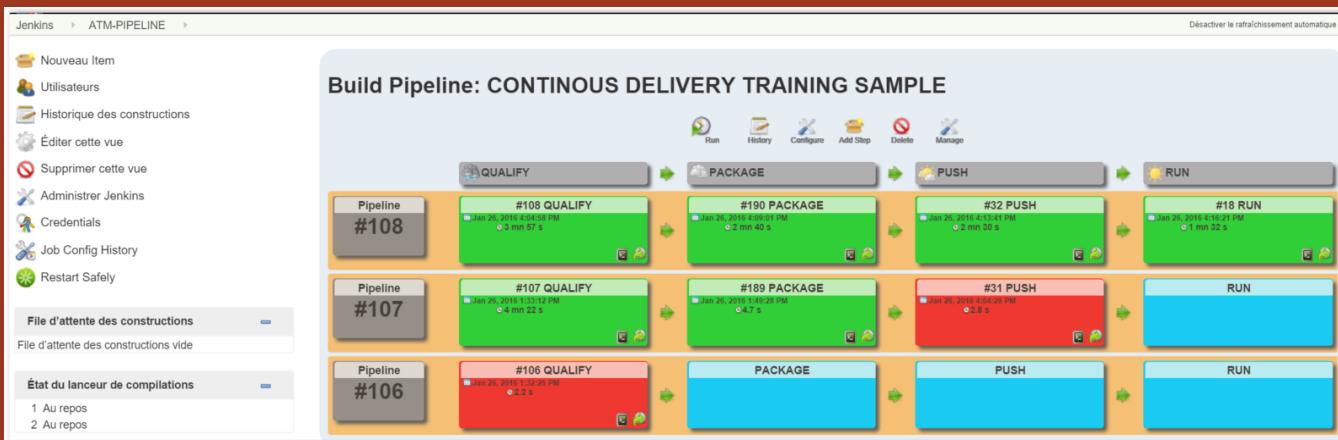
To chain the jobs visually, a pipeline view is perfect ...

# Use case #8 : Details 1/7

- First, create a pipeline view called "ATM-PIPELINE" this will ease the next steps (set first job as "QUALIFY")

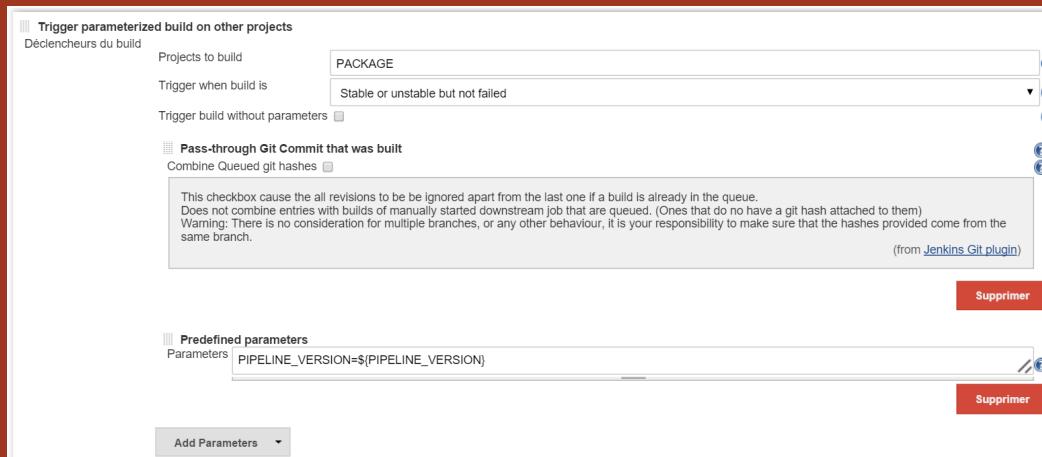


The target (if everything works as expected) :



# Use case #8 : Details 2/7

- Create now a new freestyle job called "PACKAGE"
- Reuse the SCM, Logstash, Slack configuration and save
- In QUALIFY, add a trigger to PACKAGE and pass PIPELINE\_VERSION



- Save and go back to PACKAGE one

# Use case #8 : Details 3/7

- Add a shell build step used to retrieve the binaries (scroll!)

```
wget --no-verbose http://traineegrp-nexus:8081/content/repositories/releases/org/bluebank/a...
```

The tricky part is that:

- The Codenvy image does not have Nginx installed
- The application must be copied, extracted and run in the container

The <https://github.com/nginxinc/docker-nginx.git> shows how to install and run nginx, it is a good basis for our own container

On the Github repository, add a "Dockerfile" file on the root directory containing these entries (scroll!):

# Use case #8 : Details 4/7

```
FROM codenvy/debian_jdk8

USER root

RUN apt-key adv --keyserver hkp://pgp.mit.edu:80 --recv-keys 573BFD6B3D8FBC641079A6ABABF5BD
RUN echo "deb http://nginx.org/packages/mainline/debian/ jessie nginx" >> /etc/apt/sources.

# server ports (8180 for server and 80 for nginx)
EXPOSE 8180 80

RUN apt-get update && apt-get install -y ca-certificates nginx && rm -rf /var/lib/apt/lists

# forward request and error logs to docker log collector
RUN ln -sf /dev/stdout /var/log/nginx/access.log && ln -sf /dev/stderr /var/log/nginx/error

# GET ATM server/client code
COPY ./bluebank-atm.zip /tmp/

# extract the war to run server/UI in standalone
RUN rm -rf /tmp/ATM-SERVER && mkdir -p /tmp/ATM-SERVER && cd /tmp/ATM-SERVER && unzip /tmp/ATM-SERVER.zip

# COPY THE UI DISTRIBUTION IN NGINX HTTP FOLDER
RUN cp -r /tmp/ATM-SERVER/node-client/* /usr/share/nginx/html/

# GET entry point script running server and client binaries
COPY ./atm-startup.sh /tmp/
RUN whoami && ls -l /tmp/atm-startup.sh && chmod +x /tmp/atm-startup.sh
```

# Use case #8 : Details 5/7

This configuration :

- Adds Nginx package sources to authorized list
- Exposes the HTTP port (80) and the ATM server port (8180)
- Retrieves and install Nginx tool
- Copy the ATM bundle from the workspace to the container
- Extracts the bundle (server and UI) and copies the UI in Nginx
- Copies the entry point script from the workspace to the container
- Associates it with the image (called by docker run/start)

To be able to run as expected, add near the Dockerfile the entry point script atm-startup.sh :

# Use case #8 : Details 6/7

```
#!/bin/bash

# ATM run server
cd /tmp/ATM-SERVER/lib
runningJar=$(ls bluebank-atm-*.jar)

echo " -- launching ATM CLIENT (NGINX) : nginx"
nginx

echo " -- launching ATM SERVER : java -jar ${runningJar} org.bluebank.Application"
java -jar ${runningJar} org.bluebank.Application
```

This script find and runs the auto-executable jar and nginx

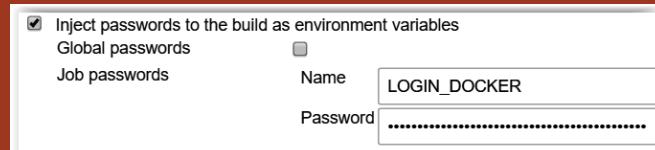
- > UI should be available on http 80 port
- > Server should be available on websocket 8081 port

- Commit and push these files in the github project

Now we have to build and run it for real...

# Use case #8 : Details 7/7

- Set a password variable (value is your Docker hub login)



- Add a shell step building the image (sudo required)

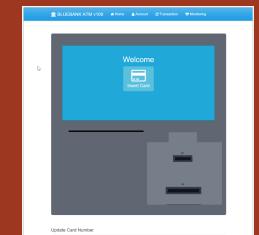
```
sudo docker build --tag=${LOGIN_DOCKER}/atm:${PIPELINE_VERSION} .
```

- Add another shell step running this image in a container

```
sudo docker run --name ${LOGIN_DOCKER}-atm-${PIPELINE_VERSION} -itd -p 80:80 -p ${LOGIN_DOCKER}/atm:${PIPELINE_VERSION}
```

- Launch the whole pipeline ...

The ATM server should be available on `http://<your_VM>`



# Use case #9 : Create PUSH job

Create a job called "PUSH" which:

- Stop/removes the ATM container tested before
- Adds a "latest" tag ATM image for continuous updates
- Logs on the your public Docker Hub registry
- Pushes the ATM image to the public Docker Hub registry
- Sends status and reports to ELK and Slack

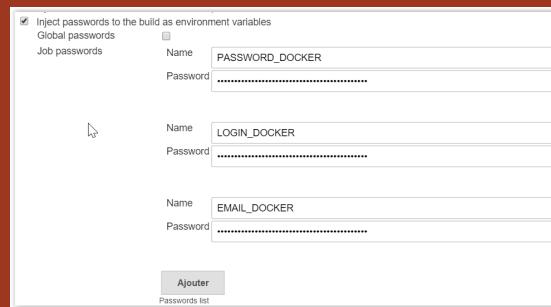
# Use case #9 : Tooltips

Latest is a similar concept as maven snapshots

This job will be set as manual in the pipeline  
=> publication requires manual approval process

# Use case #9 : Details 1/2

- Create a new freestyle job called "PUSH"
- Activate Logstash and Slack pushes as before
- Define your Docker user/password/email as password entries



- Create a shell step used to stop/remove the previous container

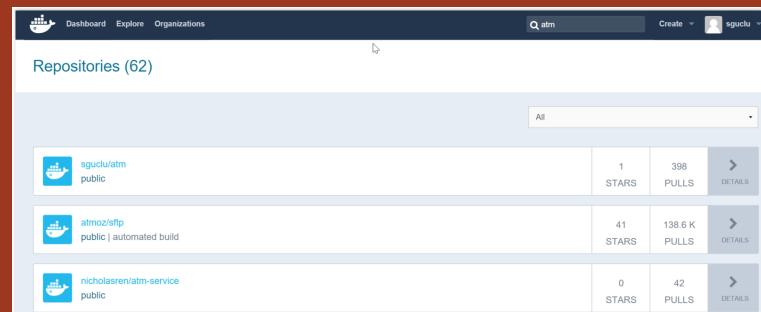
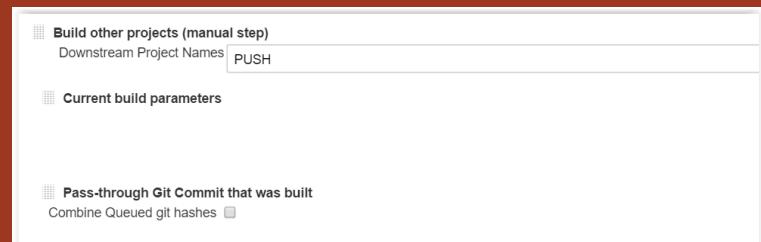
```
sudo docker stop ${LOGIN_DOCKER}-atm-${PIPELINE_VERSION} || true
sudo docker rm ${LOGIN_DOCKER}-atm-${PIPELINE_VERSION} || true
```

# Use case #9 : Details 2/2

- Add a new shell step to tag the image as "latest", then log on and push the images to your public space on Docker hub

```
sudo docker tag -f ${LOGIN_DOCKER}/atm:${PIPELINE_VERSION} ${LOGIN_DOCKER}/atm:latest
sudo docker login --username=${LOGIN_DOCKER} --password=${PASSWORD_DOCKER} --email=${EMAIL_DOCKER}
sudo docker push ${LOGIN_DOCKER}/atm
```

- Create a manual downstream link between PACKAGE and PUSH jobs  
=> application is exposed only after manual approval process



- Run the job ... and voila

# Use case #10 : Create RUN job

Configure your tutum account and "cloud" virtual machine

Create a job called "RUN" which:

- Adds a new project/tag label on the latest image
- Pushed this image to a docker cloud private repository
- Deploys automatically new images on the "run" VM
- Sends status and reports to ELK and Stash

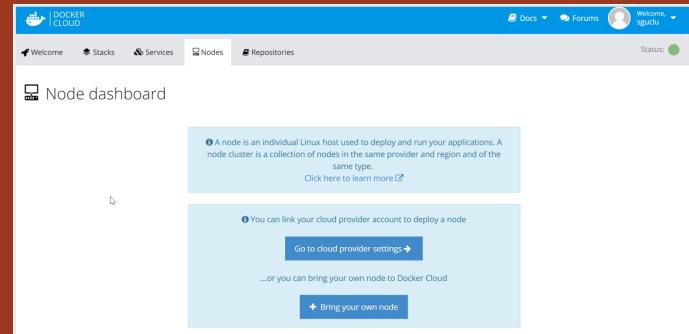
# Use case #10 : Tooltips

Docker Cloud is a container deployment and runtime service

- Formerly known as Tutum, acquired by Docker inc. in 2015
- Provides its own private registry
- Can address multiple clouds (AWS, Azure) including private ones ("Bring Your Own Cloud" feature)
- Provides many cool features to manage a docker-based infrastructure : scalability on the fly, auto-redeployments ...

# Use case #10 : Details 1/5

- Create an account on <https://cloud.docker.com> with your Docker IDs
- On the "Bring your own node" tab, declare your "run" virtual machine
- You will have to run the command provided in the UI to install the tutum agent on the "run" VM



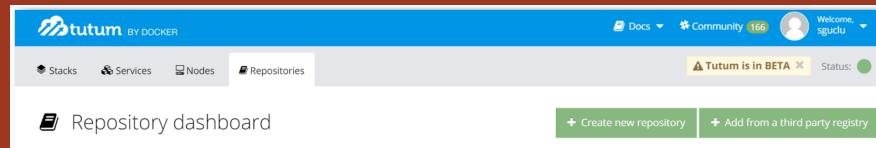
```
curl -Ls https://get.cloud.docker.com/ | sudo -H sh -s *****,
```

If something goes wrong, install can be removed before retry :

```
apt-get remove dockercloud-agent && rm -rf /etc/dockercloud-ag
```

# Use case #10 : Details 2/5

- After a while, the node should be available to accept containers
- Create a private repository called <your\_LOGIN>/atmcloud: it will host the ATM images with autodeploy features



- On Jenkins, create a new freestyle job called "RUN"
- Define the connexion parameters for Docker Cloud (same as Docker)

A screenshot of a Jenkins job configuration page. The "Global passwords" section is expanded, showing three entries for Docker passwords:

- Name: PASSWORD\_DOCKER, Password: [REDACTED]
- Name: LOGIN\_DOCKER, Password: [REDACTED]
- Name: EMAIL\_DOCKER, Password: [REDACTED]

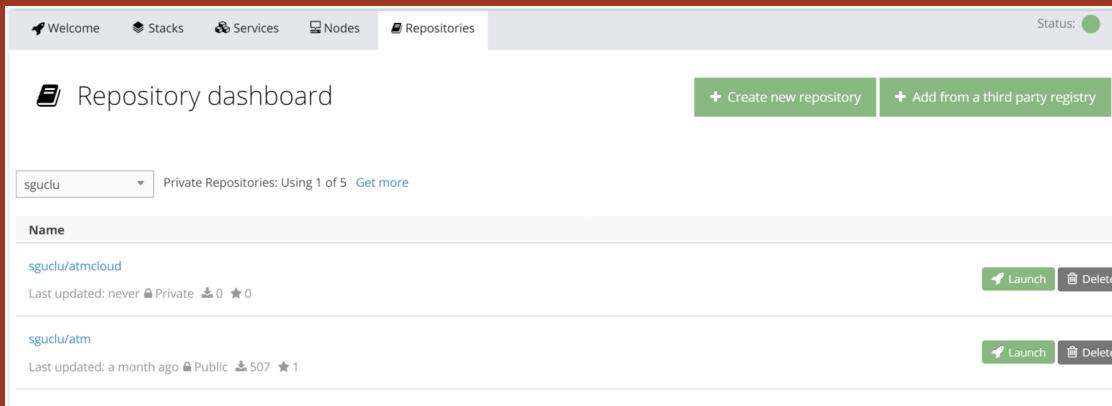
Each entry has a "Supprimer" button to its right.

# Use case #10 : Details 3/5

- Create a shell step used to tag the latest image with atmcloud name, log on docker cloud private registry and push the image

```
sudo docker tag -f ${LOGIN_DOCKER}/atm:latest ${LOGIN_DOCKER}/atmcloud:latest
sudo docker login --username=${LOGIN_DOCKER} --password=${PASSWORD_DOCKER} --email=${EMAIL_DOCKER}
sudo docker push ${LOGIN_DOCKER}/atmcloud:latest
```

- Run the job : the latest ATM image should be on Tutum registry



- We will now try to deploy manually this image ...

# Use case #10 : Details 4/5

- Using the "Launch" button, try to run the container on the node with mapped 80 (32768)/8180 (8180) ports, and autoredeploy activated
- Select "Create and deploy" ... after a while the application should be online "on the cloud"

Ports	Container port	Protocol	Published	Node port
80	tcp	<input checked="" type="checkbox"/>	dynamic	
8000	tcp	<input type="checkbox"/>	—	
8080	tcp	<input type="checkbox"/>	—	
8180	tcp	<input checked="" type="checkbox"/>	8180	

Autorestart: Off      Autodestroy: Off      Autodeploy: On

atm-63de3861 / atm-63de...

Running

Endpoints Logs Environment variables Volumes Monitoring Terminal Timeline

unknown 8180/tcp atm-63de3861-1.sguclu.cont.tutum.io:8180

http 80/tcp atm-63de3861-1.sguclu.cont.tutum.io:32768

8180->8180/tcp 32768->80/tcp

8000/tcp 8080/tcp

atm-63de3861-1.sguclu.cont.tutum.io:32768/monitoring

BLUEBANK ATM v110

Status	Healthcheck	Message
Green	ATM Service	Cash on hand is \$ 10000
Green	Deadlock	Deadlock
Green	Services	All services are started

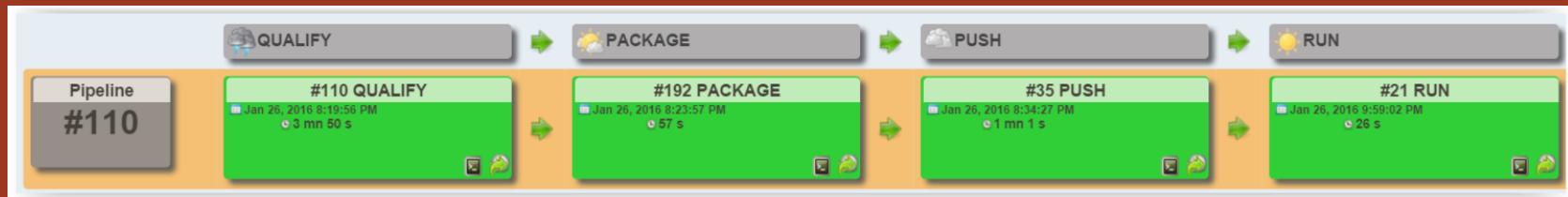
Now, let's automate the redeployment ...

# Use case #10 : Details 5/5

- On the PUSH job, add RUN a downstream job



- Launch the pipeline, then launch manually the PUSH job : ATM should be updated automatically on the "cloud" VM



# Use case #11 : Performance test

This part will be performed on your workstation

For the most courageous/advanced, you can try to execute it in the pipeline too...

- Set up and record a Gatling scenario
- Play it against the ATM server packaged in PACKAGE job

# Use case #11 : Tooltips

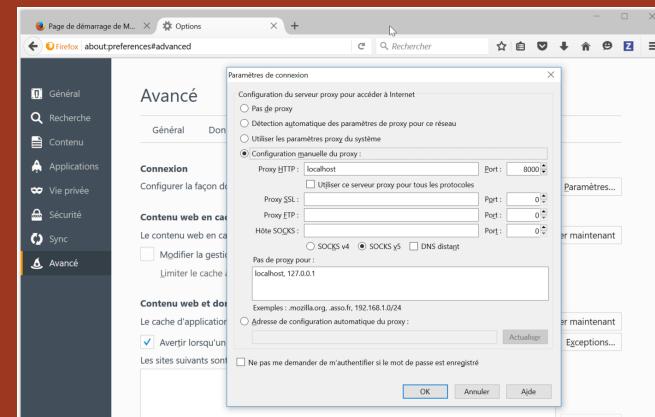
- Gatling provides a recorder to register a scenario (use browser in proxy mode)
- Once recorded (Scala code), the scenario can be replayed in standalone mode
- Reports can be generated and displayed too

# Use case #11 : Details 1/4

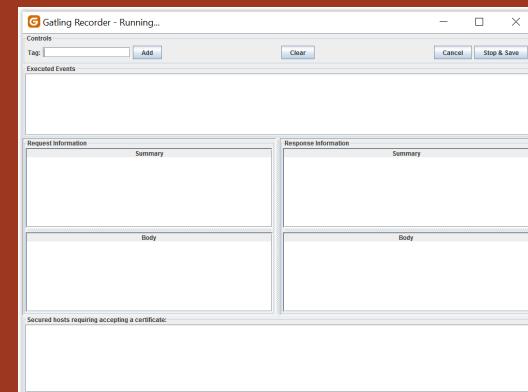
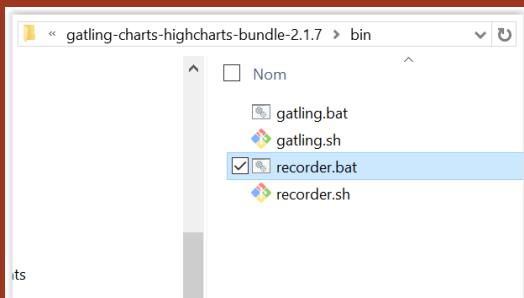
- Start the ATM server and the UI on your workstation

- Install gatling on your workstation

- Configure your browser to use a proxy on localhost, port 8000, e.g. Firefox



- Run the recorder in proxy mode

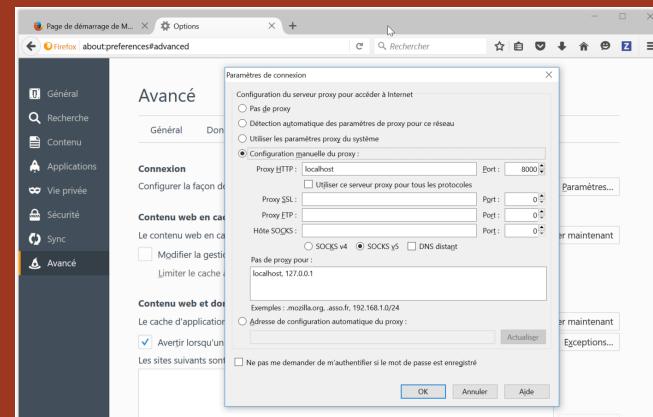


# Use case #11 : Details 2/4

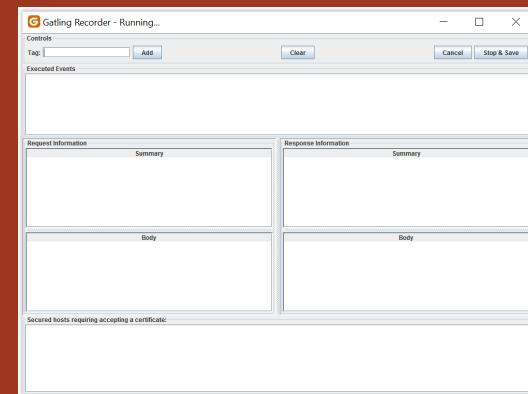
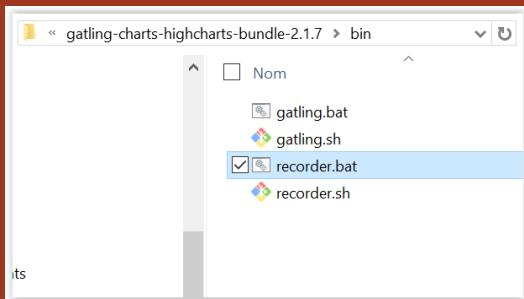
- Start the ATM server and the UI on your workstation

- Install gatling on your workstation

- Configure your browser to use a proxy on localhost, port 8000, e.g. Firefox



- Run the recorder in proxy mode

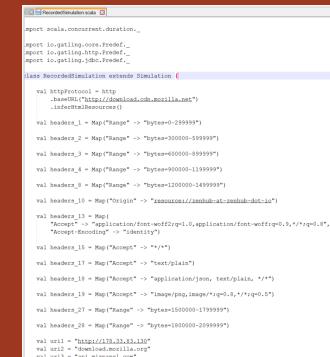


# Use case #11 : Details 3/4

- Access the ATM server running on `http://<your_VM>` with the proxified browser



- Enter PIN (1234) and select a 80\$ withdraw



- Stop the recording, and open the produced simulation file  
\user-files\simulations\RecordedSimulation.scala)

- Using the recorded session, it is possible to run the simulation again and produce reports

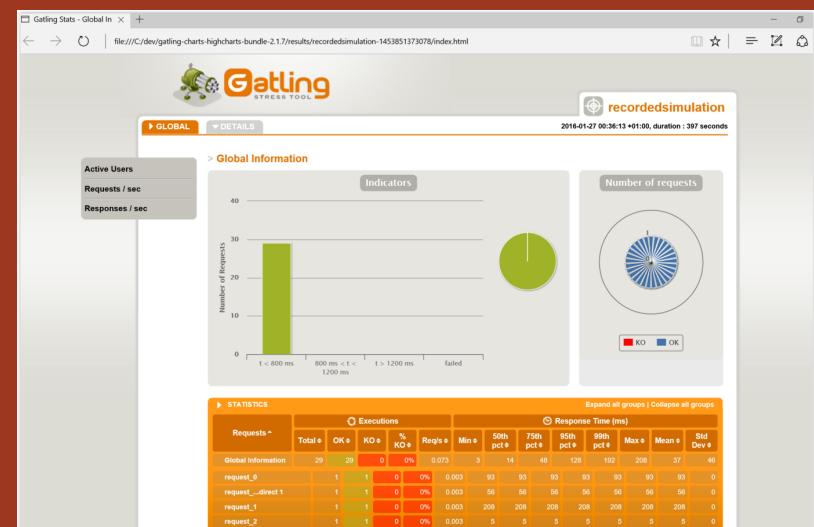
# Use case #11 : Details 4/4

```
gatling --simulation RecordedSimulation
```

- Most interesting, gatling produces plenty reports about the tests performed

```
Reports generated in 2s.  
Please open the following file: C:\dev\gatling-charts-highcharts-bundle-2.1.7\results\recordedsimulation-1453851373078\index.html
```

- Including scala and other recorded resources in the project source, it is possible to play the scenario in the pipeline (after container start in PACKAGE job ?)



## Use case #12 : Get some metrics

Based on the data collected previously, generate some reports about the activity around Jenkins per version produced

- History of successful steps
- Simplified time-to-release per version (cumulated duration between PACKAGE & RUN per version)

## Use case #12 : Tooltips

Data collected by each build will be VERY useful here

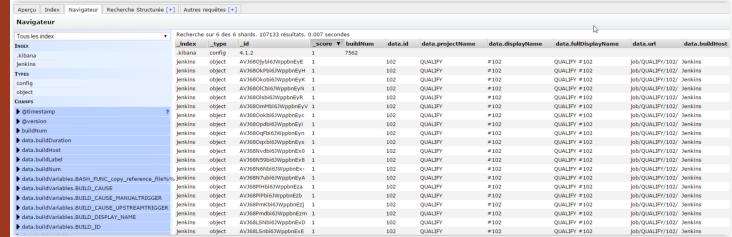
Some sensitive data appear there : a filtering can be added using logstash before inserting into elasticsearch base (not covered here)

Kibana can process indexed data out-of-the-box as long as we define an index based on the timestamp

## Use case #12 : Details 1/

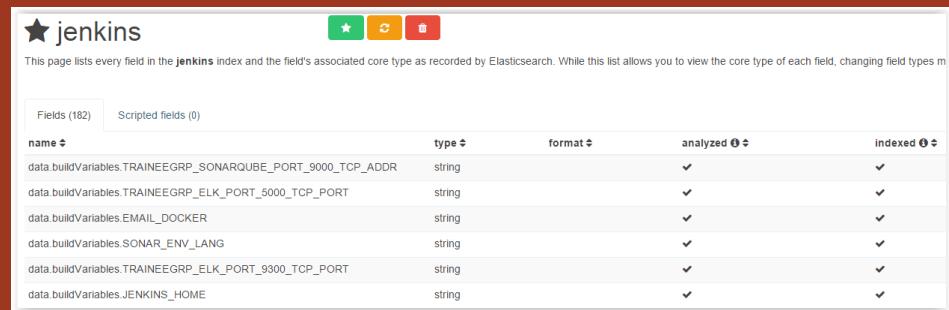
# Open elasticseach "head" UI

- http://<your\_VM>:9200/\_plugin/head
  - => the index name to process is jenkins
  - => the timestamp field is @timestamp



- Open now Kibana UI at `http://<your_VM>:5601/`
- Define an index pattern "jenkins" based on `@timestamp` and save

=> available fields are available now for reporting



## Use case #12 : Details 2/

# Open elasticseach "head" UI

- http://<your\_VM>:9200/\_plugin/head
  - => the index name to process is jenkins
  - => the timestamp field is @timestamp

- Open now Kibana UI at `http://<your_VM>:5601/`
- Define an index pattern "jenkins" based on `@timestamp` and save

=> available fields are available now for reporting



Fields (182) Scripted fields (0)				
name	type	format	analyzed	indexed
data.buildVariables.TRAINEEGRP__SONARQUBE_PORT_9000_TCP_ADDR	string		✓	✓
data.buildVariables.TRAINEEGRP__ELK_PORT_5000_TCP_PORT	string		✓	✓
data.buildVariables.EMAIL__DOCKER	string		✓	✓
data.buildVariables.SONAR_ENV_LANG	string		✓	✓
data.buildVariables.TRAINEEGRP__ELK_PORT_9300_TCP_PORT	string		✓	✓
data.buildVariables.JENKINS_HOME	string		✓	✓

# Use case #12 : Details 3/

Warning : some sensitive data appear!

107,132 hits



- Update the query field (\*) by this one and refresh the search

```
**Finished: SUCCESS** AND _exists_:data.buildVariables.PIPELINE_V
```

=> Few lines should appear now (only successful builds)

19 hits



- Pick the following fields:  
data.projectName  
  
data.buildVariables.PIPELINE\_VERSION

- Save the query HISTORY\_PER\_VERSION => successful builds/version

# Use case #12 : Details 4/

- Update the query field (\*) by this one and refresh the search

```
( "*Finished: FAILED*" OR "*Finished: SUCCESS*" ) AND _exists_:data.buildVariables.PIPELINE
```

- Put the following fields :

data.buildVariables.PIPELINE\_VERSION

data.buildDuration

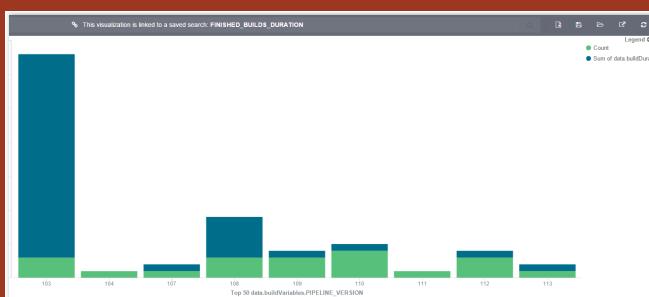
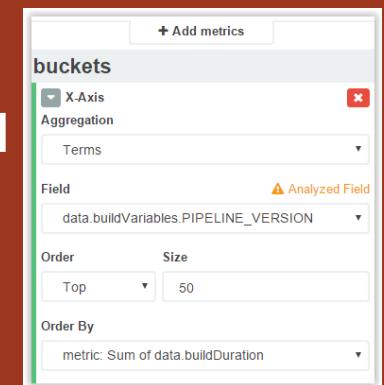
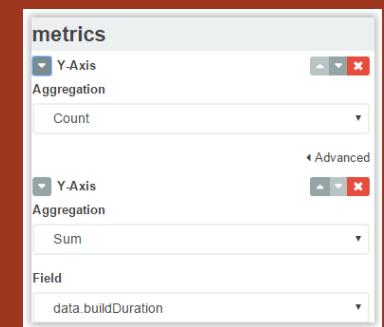
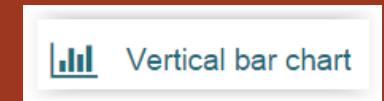
data.projectName

- Save the search as as **FINISHED\_BUILDS\_DURATION**

=> Duration steps for a version on PACKAGE/PUSH/RUN steps

# Use case #12 : Details 5/6

- Create a visualization of type "Vertical Bar Chart"
- Set count on Y-axis
- Add an aggregation on Y-Axis, summing data.build.Duration values
- As a bucket, aggregate X-Axis on the term data.buildVariables.PIPELINE\_VERSION (keep top 50 values)
- Save the view as **BUILD\_DURATION\_PER\_VERSION**

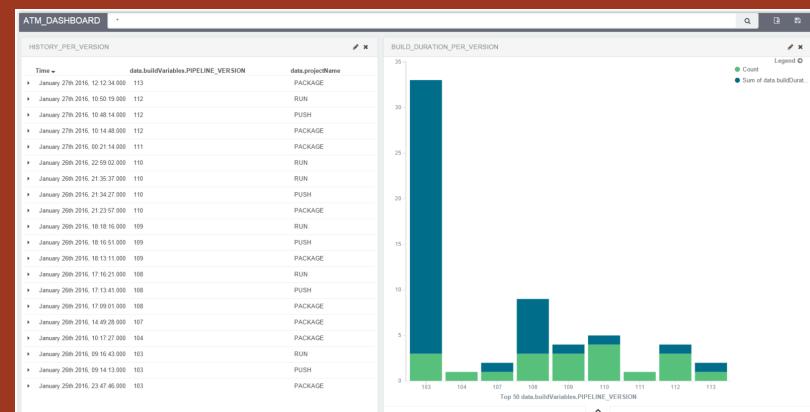


# Use case #12 : Details 6/6

- Finally, create a dashboard including : HISTORY\_PER\_VERSION query on left and BUILD\_DURATION\_PER\_VERSION visualization on right
- Save it as ATM\_DASHBOARD ... and voila

Not a really accurate measurement (QUALIFY step not measured, nor time to qualify the issue ...) but a way to keep an eye on the release efficiency

Going further needs data process during collection (e.g. logstash)



# Use case #13 : Implement a feature

Based on the toolset and practices, implement the following controls:

- Prevents negative accounts (refuse withdraw if > balance)
- Prevents banqueroute (refuse withdraw if > bank balance)

Define the BDD scenario, the tests to perform, the code to implement

Implement, test and validate everything using your software factory

Package and deploy on public registry and on cloud

# Use case #13 : Tooltips

The BDD scenario for first business rule should be something like this:

```
No negative account
    Given the account balance is 100 dollars
        And the valid pin number has been entered
    When a withdraw of 120.00 dollars is made
        Then the ATM rejects the withdraw
            And the balance is 100 dollars
            And the card is returned
            And a receipt is printed

    LOCATION: 100 Main Street, NewYork 12345,USA
    CARD NO: 123456
    DATE 01/07/2015 TIME 17:05:00
    TERMINAL 47
    TRANSACTION#00000000-0000-0001-0000-000000000002
    WITHDRAW OF $120.00
    ----- REJECTED : ONLY 100 USD LEFT -----
```

# Use case #13 : Details

This part is yours ....

# Questions ?



# Thank you

