JENKINS MODERNE ET LIVRAISON CONTINUE

Écrivons ensemble des Pipeline Déclaratifs pour votre application

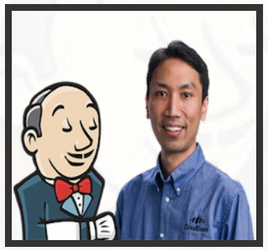
MOTIVATIONS OF THIS TALK

Arguing with an Engineer is a lot like wrestling in the mud with a pig. After a couple of hours, you realize the pig likes it.

JENKINS PROJECT

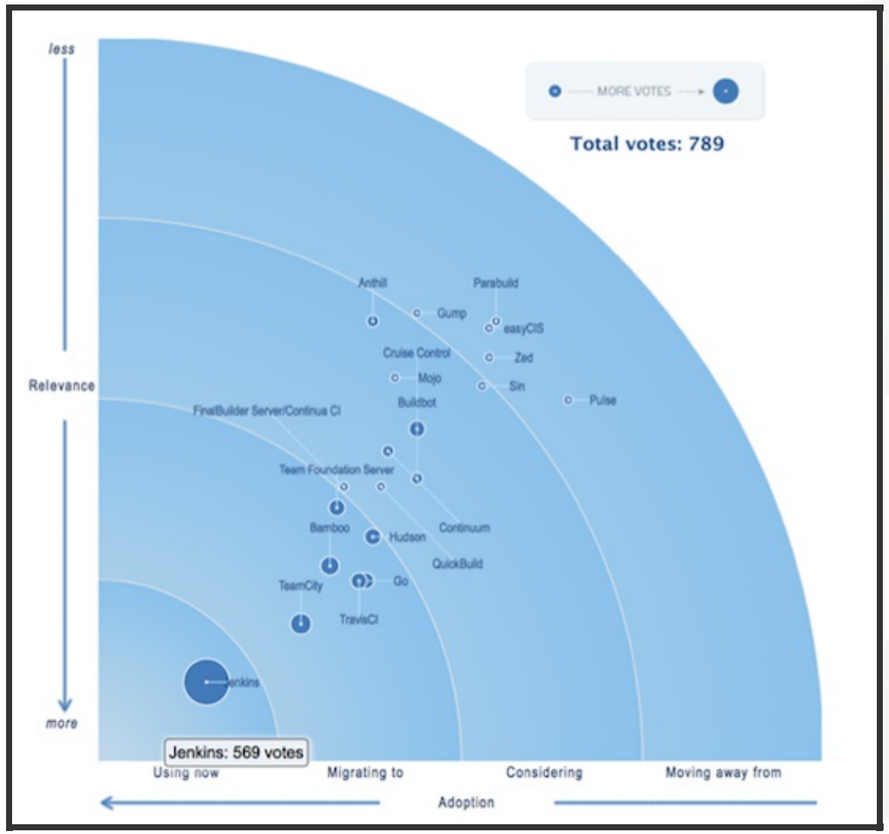
Jenkins

MEET JENKINS



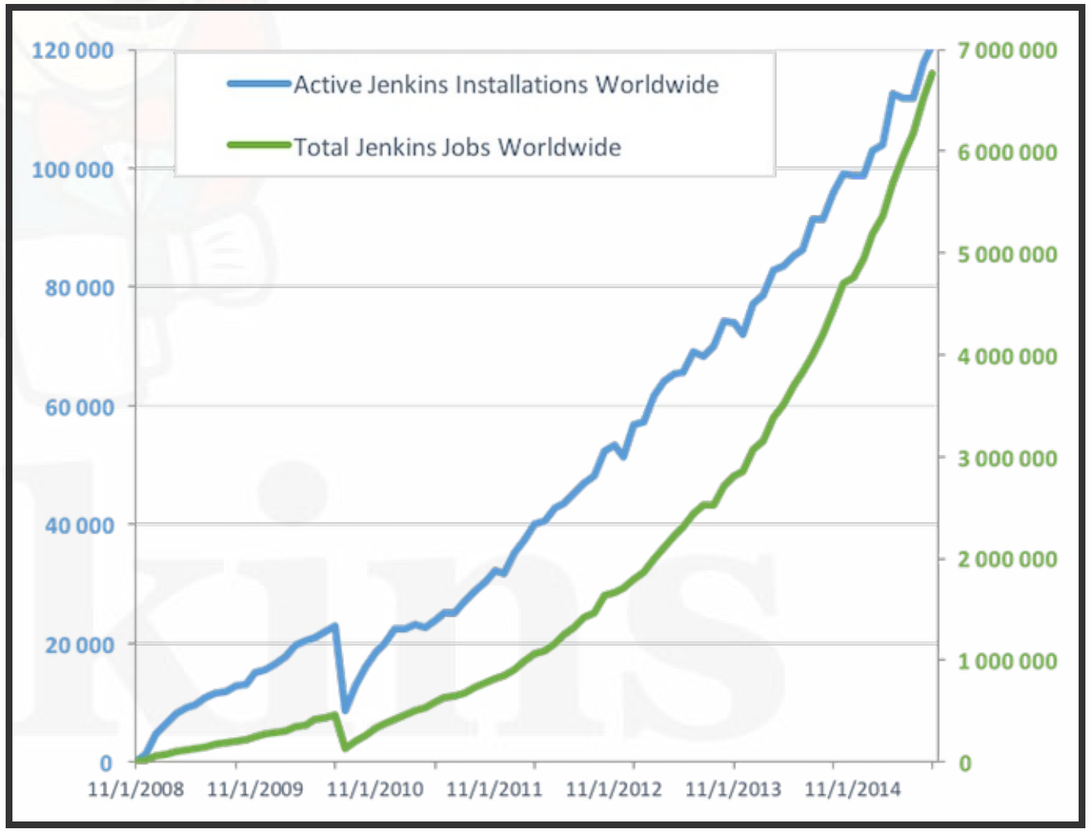
- #1 Continuous Integration and Delivery server
- Created by Kohsuke Kawaguchi 12 years ago (Hudson)
- An independent and active community (https://jenkins.io)
- 100,000 active installations
- 300,000 Jenkins servers
- 1,200+ plugins

JENKINS POPULARITY

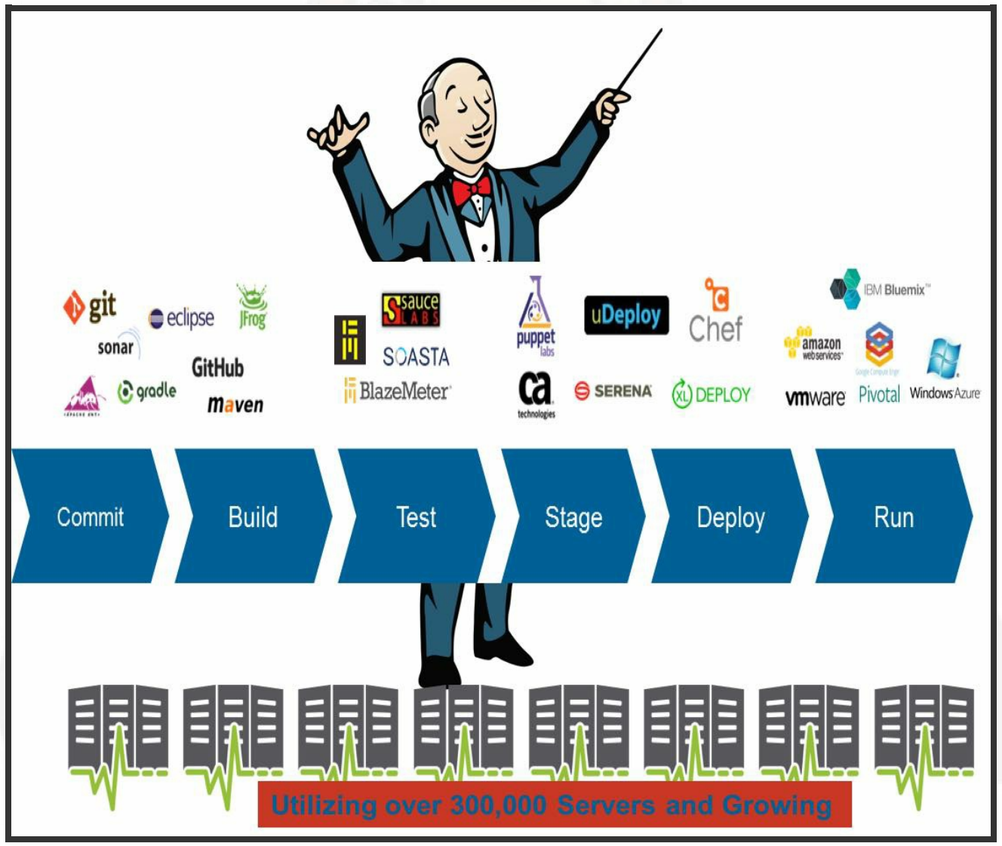


http://stats.jenkins.io/

http://www.infoq.com/research/ci-server



JENKINS IS THE CD ORCHESTRATOR



WHY JENKINS 2

- Jenkins 1 is more than 12 years old
- Because Continuous Integration have changed…
  - jenkins-ci.org !?
- slave → agent
- "Fire and forget"
- "Modern Web":
  - https://jenkins.io
  - https://jenkins.io/docs
  - https://plugins.jenkins.io

JENKINS 2 GOALS

- **Target:** CI → CD
- No breaking changes from Jenkins 1
  - Smooth upgrade
  - Plugins compatibility
- First time experience improvement
  - Brand new Wizard
- **Pipeline-as-Code:**
  - Jenkinsfile stored in SCM
  - Groovy DSL: "Code your Pipeline"

JENKINS IN 2017

- Declarative Pipeline
  - Still Jenkinsfile
  - Easier
  - Compatible with **Scripted Pipeline**
- BlueOcean
  - Brand new GUI
  - Written in ReactJS
  - Opiniated

HELLO

WHOAMI: DAMIEN DUPORTAL

- Training Engineer @ CloudBees
  - Docker & Apple fanboy. Sorry
  - Human stack focused
  - Rock climber
- Contact:
  - Twitter: @DamienDuportal
  - Github: dduportal
  - Google: damien.duportal@gmail.com

CLOUDBEES

<sell>

- Software at the Speed of Ideas
- Hub of "Enterprise Jenkins and DevOps"
- We provide:
  - Jenkins "Enterprise" Distribution
  - Services around Jenkins

</sell>

WHO ARE YOU ?

LET'S GET STARTED

- Access your instance homepage:

  http://localhost:10000

DEMO APPLICATION

DEMO APPLICATION: WHY ?

- **Goal:** Illustrate a Software Supply Chain with a demo application
- **Challenge:** So many languages/framework/toolchains
- **Solution:**
  - Opinionated demo application (language, tooling, etc.)
  - Put everyone on same page with initial exercise

DEMO APPLICATION: WHAT ?

- Web application
- Homepage show a link to /greeting endpoint
- Endpoint /greeting: greets the world
- Provides the parameter **name**: greet the person
  - /greeting?name=Butler prints Hello Butler

DEMO APPLICATION: TECHNICAL STACK

- This is the Spring Boot Starter
- Language: **Java** (OpenJDK 8)
- Toolchain: **Maven** (Maven >= 3.3)
- Source code stored inside a local **Git** repository

DEMO APPLICATION: HOW ?

DEMO APPLICATION: ACCESS IT

- Open the local **GitServer**:
  - http://localhost:10000/gitserver
- **Sign In** using the top-right button
  - User is butler, same for the password
- Browse to the repository. Either:
  - Click on **Explore → butler/demoapp**
  - or direct link: http://localhost:10000/gitserver/butler/demoapp

DEMO APPLICATION: CHECK IT

- Maven configuration: pom.xml
- Application Source code: src/main/java/
- Application Templates/HTML: src/main/resources/
- Application Test code: src/test/java

DEMO APPLICATION: GET IT

- Open the **DevBox**, the Web based command line:
  - http://localhost:10000/devbox
  - WebSockets must be authorized
- Copy the demoapp repository URL from **GitServer**
- Run the following commands:

```
# Get the git repository
git clone http://localhost:10000/gitserver/butler/demoapp.git
# Browse to the local repository
cd ./demoapp
# Check source code
ls -l
cat pom.xml
```

DEMO APPLICATION: DEVBOX TRICKS

- Clear the window: `clean`
- Show command history: `history`
- CTRL + R: search the command history interactively
- CTRL + C: cancel current command and clean line buffer
- CTRL + A: jump to beginning of line
- CTRL + E: jump to end of line

DEMO APPLICATION: MAVEN

- Maven TL;DR:
  - Provide a standardized **workflow**
  - pom.xml describe the application
- Maven Command line : mvn, expects **goals** (workflow steps)

```
mvn dependency:list
```

- Can have **flags** (configuration on the fly)

```
mvn dependency:list -fn
```

DEMO APPLICATION: COMPILE IT

- Maven goal is **compile**
  - Solves dependencies for build
  - Process source code
  - Generate classes
- Content put in the **./target** folder:

```
mvn compile
ls -l ./target
```

DEMO APPLICATION: UNIT-TEST IT

- Maven goal is **test**
  - Solves dependencies for unit test
  - Call compile
  - Compile Unit Test classes
  - Run Unit Test
- Tests Reports put in the **./target/surefire-reports** folder:

```
mvn test
ls -l ./target/surefire-reports
```

DEMO APPLICATION: BUILD IT

- Maven goal is **package**
  - Solves dependencies for packaging
  - Call compile and test
  - Package the application as specified in pom.xml
- Put the artifact (generated packages) in ./target

```
mvn package
ls -lrh ./target/
```

DEMO APPLICATION: RUN IT

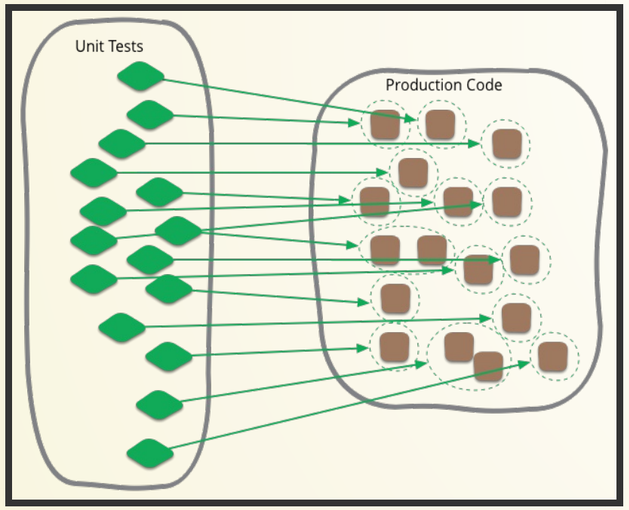
- Spring Boot demo is run as an **"Über-Jar"**
- You only need the java CLI from a JRE:

```
java -jar ./target/demoapp.jar
```

- Check the application on the 10080 port:
  - http://localhost:10080

DEMO APPLICATION: A NOTE ABOUT TESTS

- Unit / Integration Test ?
  - Bedtime reading: https://martinfowler.com/tags/testing.html



DEMO APPLICATION: INTEGRATION TESTING

- Maven goal is **verify**
  - Solves dependencies for integration tests
  - Call compile, test and package
  - Run Tests against the **packaged** application
- Tests Reports put in the **./target/failsafe-reports** folder:

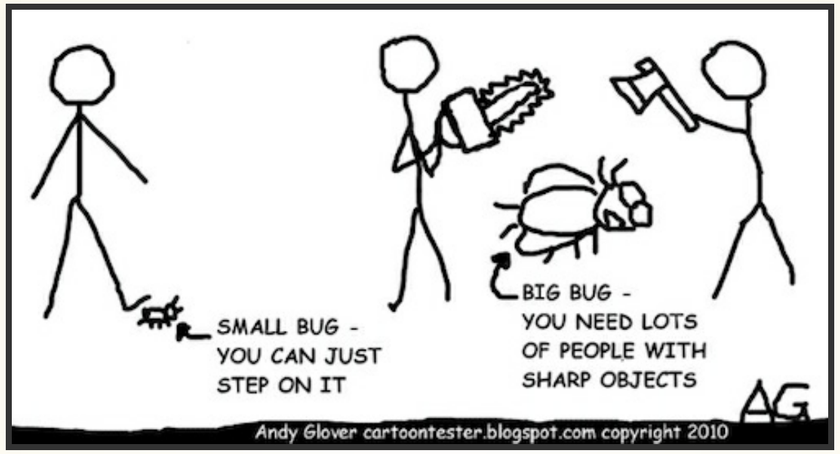
```
mvn verify
ls -l ./target/failsafe-reports
```

THAT'S ALL FOLKS !

CONTINUOUS INTEGRATION WITH JENKINS

aka "CI"

CI: WHY ?



Continuous Integration doesn't get rid of bugs, but it does make them dramatically easier to find and remove.

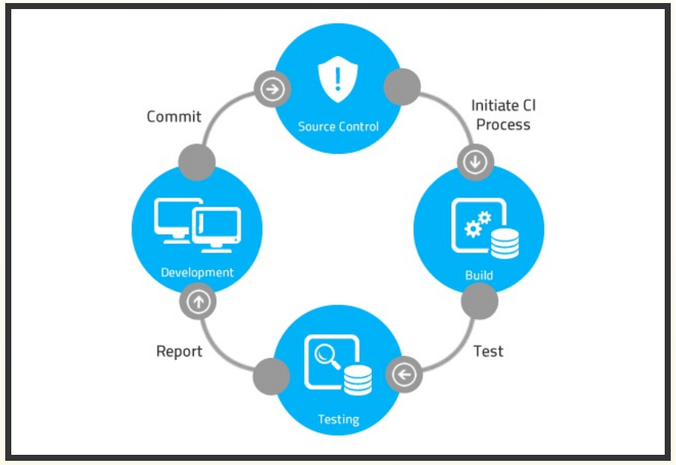
— Martin Fowler

CI: WHAT ?

> *Continuous Integration is a software development* ***practice*** *where members of a team integrate their work* ***frequently****, usually each person integrates at least daily, leading to* ***multiple*** *integrations per day.*
>
> — Martin Fowler - Continuous Integration

CI: HOW ?

- Each integration is verified by an **automated** build (including test)
- Integrate code **often**, at least daily, to make integration a **non-event**
- **Continuously** build and integrate, with a **feedback loop**



CONTINUOUS INTEGRATION WITH JENKINS

CI: ACCESSING JENKINS

- Access your Jenkins instance:
  - http://localhost:10000/jenkins
  - **Log in** as the user **butler** (password is the same)
  - It is the "Legacy UI"

CI: JENKINS BLUEOCEAN

- Switch to BlueOcean, the new UI
  - Direct link: http://localhost:10000/jenkins/blue
  - *Or* click on the top button "Open Blue Ocean"

CI: CREATING PROJECT

- Create your 1st Pipeline:
  - Stored in **Git**
  - Fetch URL from the **Gitserver**
    - Direct link: http://localhost:10000/gitserver/butler/demoapp.git
  - Add a **User/password** credential
    - butler / same for password

CI: FAST FEEDBACK WITH WEBHOOKS

- We want **Fast feedback** !
  - Pushed code to repository ? Tell Jenkins to build it **now**
- Let’s add a **Webhook** to the repository
  - HTTP request **Gitserver → Jenkins**

CI: ADD A GOGS WEBHOOKS

- From repo. in **Gitserver → Settings → Webhooks**
  - Direct link: http://localhost:10000/gitserver/butler/demoapp/settings/hooks
- Add a **Gogs** webhook:
  - Payload URL: http://localhost:10000/jenkins/job/demoapp/build?delay=0
  - When should this webhook be triggered?: **I need everything**

CI: STARTING WITH PIPELINES

- Pipeline-as-code: We need a Jenkinsfile
  - **Declarative** or **Scripted** ?
- Where to start ?
  - Documentation: https://jenkins.io/doc/pipeline
  - Getting Started: https://jenkins.io/doc/pipeline/tour/hello-world/
  - Syntax Reference: https://jenkins.io/doc/book/pipeline/syntax/

CI: BLUEOCEAN PIPELINE EDITOR

- Provides the **full** round trip with SCM
- No Pipeline ? Follow the wizard (not Gandalf fool !)
- Already have a Pipeline ? Edit, commit, run it
- Needs a **compliant** SCM
  - Only Github with BO 1.0.1
  - Interested ? **Open-Source**: Contribute !

CI: USE THE PIPELINE EDITOR

- Let's **hack**: open the **BlueOcean Pipeline Editor**
  - Direct (hidden) URL: http://localhost:10000/jenkins/blue/organizations/jenkins/pipeline-editor/
  - Use CTRL + S (On Mac: CMD +S) to switch to textual version
- Use also the Legacy Pipeline Syntax Snippet Generator:
  - http://localhost:10000/jenkins/job/demoapp/pipeline-syntax/

CI: EXERCISE - YOUR FIRST PIPELINE

- Use the BO editor and **Gitserver**
- Create a Pipeline that have a single stage "Hello"
- This stage have 1 step that prints the message "Hello World"
- Copy/Paste this Pipeline in a new file Jenkinsfile on the repository root
- A build will kick off immediately:
  - demoapp Activity Dashboard

CI: SOLUTION - YOUR FIRST PIPELINE

```
pipeline {
  agent any
  stages {
    stage('Build') {
      steps {
        echo 'Hello World !'
      }
    }
  }
}
```

CI: EXERCISE - SIMPLE BUILD PIPELINE

- Exercise: Implement a simple build pipeline demoapp
- We want 4 stages, for the 4 Maven goals:
  - clean compile, test, package, verify
- We need to build on the maven agent

CI: SOLUTION - SIMPLE BUILD PIPELINE

```
pipeline {
  agent {
    node {
      label 'maven'
    }
  }
  stages {
    stage('Compile') {
      steps {
        sh 'mvn compile'
      }
    }
    stage('Unit Tests') {
      steps {
        sh 'mvn test'
      }
```

CI: EXERCISE - ARTIFACTS

- We want to simplify to 2 stages, based on Unit Tests definition:
  - Build: compile, unit test and package the application
  - Verify: Run Integration Tests
- We also want to **archive** the generated jar file
  - Only if the build is in sucess
- *Clues*: Keywords post + success (not in Editor), and archiveArtifacts

CI: SOLUTION - ARTIFACTS

```
pipeline {
  agent {
    node {
      label 'maven'
    }
  }
  stages {
    stage('Build') {
      steps {
        sh 'mvn clean compile test package'
      }
    }
    stage('Integration Tests') {
      steps {
        sh 'mvn verify'
      }
```

CI: EXERCISE - INTEGRATION TESTS REPORTS

- We want the integration test reports to be published to Jenkins
  - **Better** feedback loop
- If Integration Tests are failing, do NOT fail the build
  - Make it **UNSTABLE** instead
- *Clues*:
  - Maven flag -fn ("Fails Never")
  - keyword junit (Pipeline keyword)

CI: SOLUTION - INTEGRATION TESTS REPORTS

```
pipeline {
  agent {
    node {
      label 'maven'
    }
  }
  stages {
    stage('Build') {
      steps {
        sh 'mvn clean compile test package'
      }
    }
    stage('Integration Tests') {
      steps {
        sh 'mvn verify -fn'
        junit '**/target/failsafe-reports/*.xml'
```

CI: EXERCISE - ALL TESTS REPORTS

- We now want **all** test reports published
  - **Problem:** how to handle Unit test failure ?
- We also want to archive artifacts if build is unstable **only** due to the Verify stage
- *Clues*: post can be used per stage

CI: SOLUTION - ALL TESTS REPORTS

```
pipeline {
  agent {
    node {
      label 'maven'
    }

  }
  stages {
    stage('Build') {
      steps {
        sh 'mvn clean compile test package'
      }
    post {
      always {
      junit '**/target/surefire-reports/*.xml'
    }
```

CI: FAILING TESTS

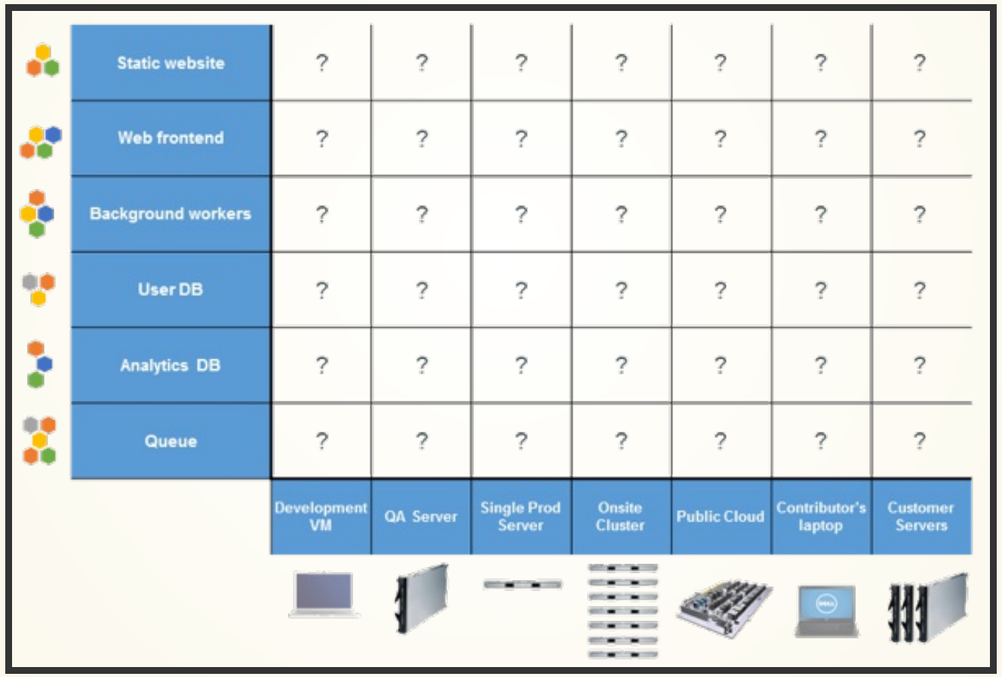
- Validate your changes by making your tests fails.
- Edit each one and uncomment the failing block:
  - Integration: src/master/src/test/java/hello/ApplicationIT.java
  - Unit Tests: src/master/src/test/java/hello/ApplicationTest.java
- Browse the top-level items "Changes", "Tests" and "Artifacts"
- Do **NOT** forget to correct your tests at the end

THAT'S ALL FOLKS !

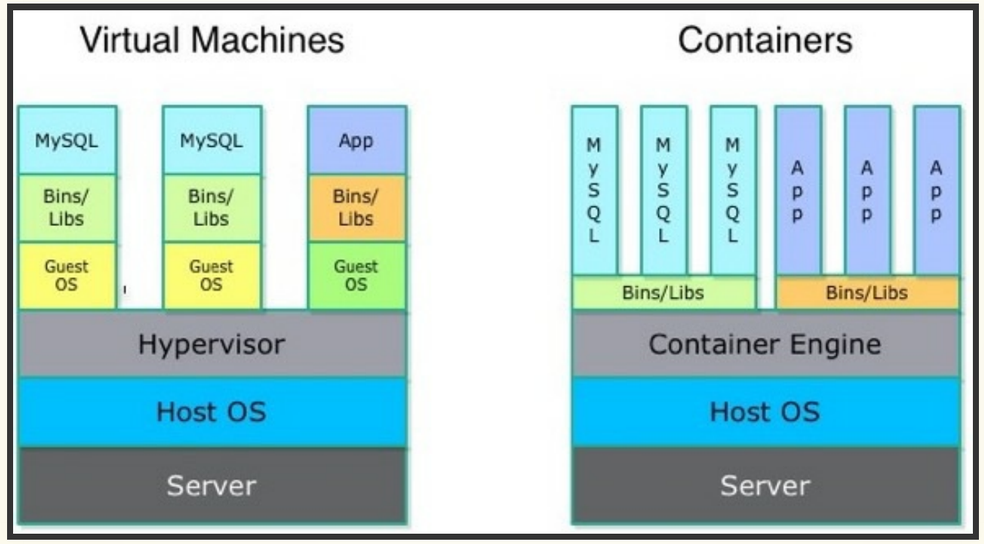
DOCKER

to the Rescue

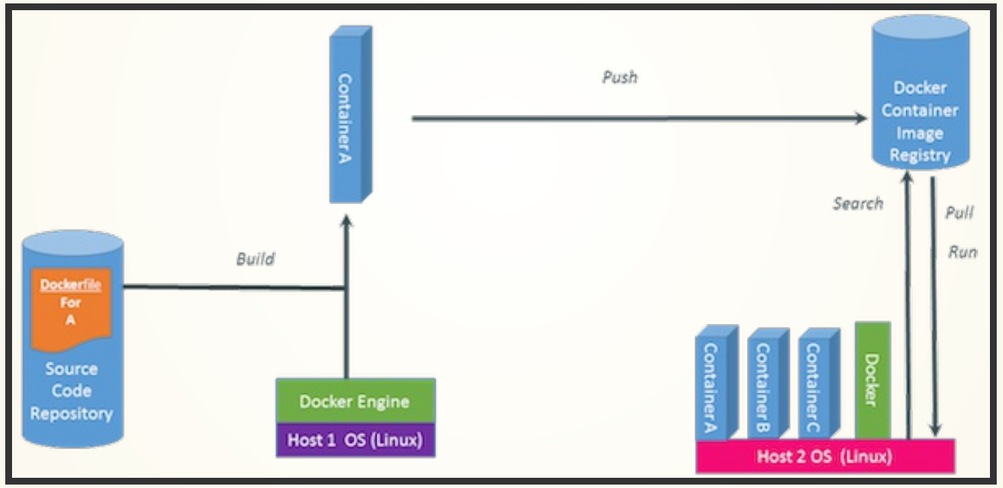
DOCKER: WHY ?


Static website
Web frontend
Background workers
User DB
Analytics DB
Queue
?
Development VM
QA Server
Single Prod Server
Onsite Cluster
Public Cloud
Contributor's laptop
Customer Servers

DOCKER: WHAT ?


Virtual Machines
MySQL
Bins/ Libs
Guest OS
MySQL
Bins/ Libs
Guest OS
App
Bins/ Libs
Guest OS
Hypervisor
Host OS
Server
Containers
MySQL
MySQL
MySQL
App
App
App
Bins/Libs
Bins/Libs
Container Engine
Host OS
Server

DOCKER HOW ?


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

DOCKER: DOCKERFILE

- Dockerfile: recipe for building your **immutable** image

```
FROM debian:jessie
LABEL Maintainer="Damien DUPORTAL"

RUN apt-get update && apt-get install -y nginx

VOLUME ["/tmp","/app"]

EXPOSE 80

ENTRYPOINT ["/usr/sbin/nginx"]
CMD ["-g","daemon off;"]
```

DOCKER: BUILDING DOCKER IMAGE

- Using the docker CLI:

```
docker build -t my_image:1.0.0 ./
```

DOCKER: RUNNING A DOCKER

- Using the docker CLI:

```
docker run -P -d my_image:1.0.0
```

DOCKER: DEMO APPLICATION'S DOCKERFILE

- Using **GitServer**, from the repository root
  - Check the Dockerfile content

DOCKER: BUILDING DEMO APPLICATION

- Using **Devbox**, from the local repository root
  - Checking images with `docker images`
  - Build an image named `demoapp:latest`
  - Check again images

DOCKER: RUNNING DOCKER CONTAINER

- Using **Devbox**, from the local repository root
  - Check running containers with `docker ps`
  - Run the image with this command:

```
docker run -p 8080:8080 -d my_image:1.0.0
```

  - Test it
  - Stop it with `docker stop <Container ID>`

CONTINUOUS DELIVERY WITH JENKINS

aka "CD"

CD: WHY ?

How long would it take to your organization to deploy a change that involves just one single line of code?

- Reduce deployment risks
- Allow more frequent user feedback
- Make progress believable by everyone

CD: WHAT ?

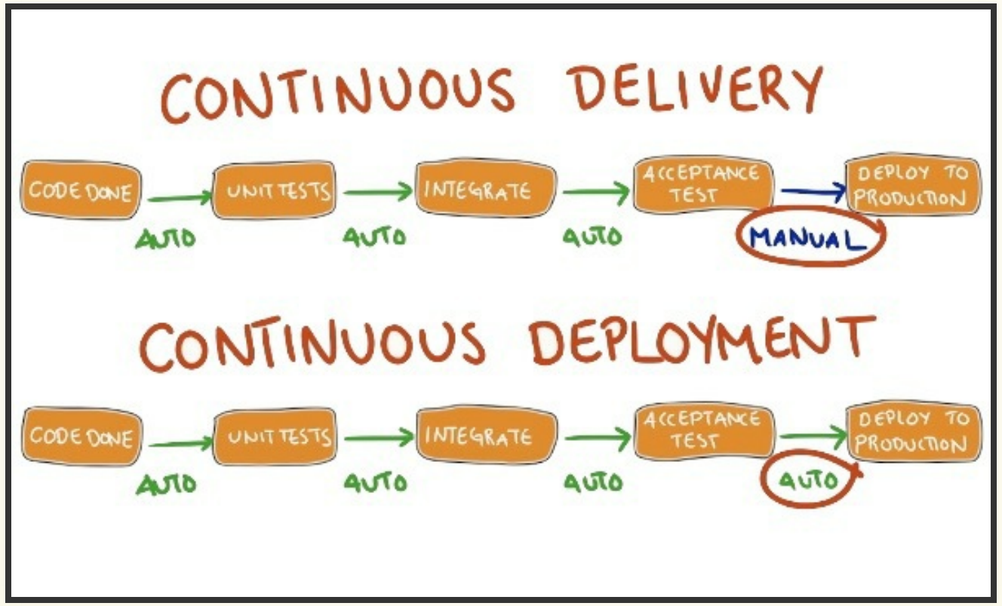
Continuous Delivery is the next step after Continuous Integration:

- Every change to the system **can** be released for **production**
- Delivery can be done at **any** time, on **any** environment

> *Your team prioritizes keeping the software **deployable** over working on new features*
>
> — Martin Fowler

CD IS NOT CONTINUOUS DEPLOYMENT

Both are always confused:



CD: HOW ?

- Having a collaborating working relationship with **everyone** involved
- Using **Deployment Pipelines**, which are automated implementations of your application's build lifecycle process

CD: DELIVERY TARGET

- Production runs on Docker
- Your Ops team use a Docker Registry
- Expected Artifact:
  - Not a jar file
  - But a Docker image

CD: EXERCISE - GENERATE DOCKER IMAGE

- Goal: Add a stage named "Docker" for building the Docker image
  - **Before** Integration Test
  - This steps should use the name `demoapp:latest`
  - **Challenge:** we need the `jar` file
- *Clues:* Keywords `stash` and `unstash`

CD: SOLUTION - GENERATE DOCKER IMAGE

```
pipeline {
  agent {
    node {
      label 'maven'
    }
  }
  stages {
    stage('Build') {
      steps {
        sh 'mvn clean compile test package'
        stash(name: 'app', includes: 'target/demoapp.jar')
      }
      post {
        always {
          junit '**/target/surefire-reports/*.xml'
        }
```

CD: EXERCISE - DOCKER SMOKE TEST

- Goal: Run a Docker **Smoke Test** before Integration Tests
  - Use the command `bats ./src/test/bats/docker.bats`
    - It takes care of building and naming the docker resources
  - We do not need the archive the artifact unless Integration Test is unstable

CD: SOLUTION - DOCKER SMOKE TEST

```
pipeline {
  agent {
    node {
      label 'maven'
    }
  }
  stages {
    stage('Build') {
      steps {
        sh 'mvn clean compile test package'
        stash(name: 'app', includes: 'target/demoapp.jar')
      }
      post {
        always {
          junit '**/target/surefire-reports/*.xml'
        }
```

CD: EXERCISE - APPROVAL AND DELIVERY

- Goal: Manual Approval for Delivery
- Add a stage named `Delivery` that will:
  - Ask for a manual validation, after Integration Tests
  - And will push the Docker Image to the registry `localhost:5000`
- *Clues:* Keyword `input`

CD: SOLUTION - APPROVAL AND DELIVERY

```
pipeline {
  agent {
    node {
      label 'maven'
    }
  }
  stages {
    stage('Build') {
      steps {
        sh 'mvn clean compile test package'
        stash(name: 'app', includes: 'target/demoapp.jar')
      }
      post {
        always {
          junit '**/target/surefire-reports/*.xml'
        }
```

CD: EXERCISE - BUILDING WITH DOCKER

- Goal: Use Docker to provide the build environment
  - Use the agent allocation to build and run builds within a Docker container
  - Use the `Dockerfile.build` from the repository
- *Clues:* Keyword `agent any`, `agent { dockerfile }`

CD: SOLUTION - BUILDING WITH DOCKER

```
pipeline {
  agent any
  stages {
    stage('Build') {
      agent {
        dockerfile {
          filename 'Dockerfile.build'
        }
      }
      steps {
        sh 'mvn clean compile test package'
        stash(name: 'app', includes: 'target/demoapp.jar')
      }
      post {
        always {
          junit '**/target/surefire-reports/*.xml'
```

CD: EXERCISE - PARALLEL STAGES

- Goal: Run Stages in parallels to gain time
  - We can safely run Docker Smoke and Integration Tests in parallel
  - To specify a specific agent, use Scripted Pipeline Block and the node allocation
- *Clues:* Keyword parallel, script, node

CD: SOLUTION - PARALLEL STAGES

```
pipeline {
  agent any
  stages {
    stage('Build') {
      agent {
        dockerfile {
          filename 'Dockerfile.build'
        }
      }
      steps {
        sh 'mvn clean compile test package'
        stash(name: 'app', includes: 'target/demoapp.jar')
      }
      post {
        always {
          junit '**/target/surefire-reports/*.xml'
```

CD: EXERCISE - SCALING PIPELINE

- **Goal:** Share Pipeline across your teams
- We want to use **Shared Libraries**
- There is one autoconfigured named deploy
- Use the annotation to load the Library, on master branch
- Check the library here

CD: SOLUTION - SCALING PIPELINE

```
@Library('deploy@master') _

pipeline {
  agent {
        label 'docker'
        }
  stages {
    stage('Build') {
      agent {
        dockerfile {
          filename 'Dockerfile.build'
        }
      }
      steps {
        sh 'mvn clean compile test package'
        stash(name: 'app', includes: 'target/demoapp.jar')
```

THAT'S ALL FOLKS !

THANK YOU !

- Mix-IT organization
- CloudBees and Jenkins Community
- YOU