Learn DevOps

Jenkins

Jenkins intro video

Why Jenkins?

Why Jenkins?

- **Jenkins is the most popular tool** to do Continuous Integration and Continuous Delivery of your software
- It's **free and open source**
- Jenkins is still being actively developed
- It has a **strong community** with thousands of plugins you can use
- Jenkins is used in a lot of companies, from **startups** to **enterprises**
- If you're looking for a job in the **DevOps space**, Jenkins is a must have skill

Why this course?

- This course gives you a **fresh overview** of Jenkins using the **latest Jenkins features** to efficiently automate your **delivery pipeline**
  - I'll teach you how to use Jenkins "**the DevOps way**"
    - Using the Job DSL and the Jenkins Pipelines (the Jenkinsfile)
    - Building your software projects with Docker
    - Installing Jenkins on the cloud, also using docker

Why this course?

- I'll show you how to integrate Jenkins with **popular development software**, like:
  - Slack
  - JFrog
  - Sonar
  - Github and Bitbucket
  - Docker Hub

Course Layout

Introduction	Jenkins, the DevOps way	Integrations	Advanced Jenkins usage
Introduction to Jenkins	Infrastructure as code and automation	Email integration	Jenkins slaves
Installation	Job DSL	Slack integration	Blue Ocean
Building a NodeJS app with Jenkins and Docker	Jenkins Job Pipeline	Github & Bitbucket integration	ssh-agent plugin
	Running build, test, deploy in docker	JFrog integration	Security best practices
		Custom API integration	authentication and authorization strategies
		Sonar integration	Onelogin integration

Introduction

Introduction

- Who am I
- Course Layout
- Course Objectives

Who am I

- My name is Edward Viaene
- I am a **consultant** and **trainer** in Cloud and Big Data technologies
- I'm a big advocate of **Agile** and **DevOps techniques** in all the projects I work on
- I held various roles from **banking** to **startups**
- I have a **background** in Unix/Linux, Networks, Security, Risk, and distributed computing
- Nowadays I specialize in everything that has to do with **Cloud** and **DevOps**

Online Training

- **Online training** on Udemy
  - **DevOps**, **Distributed Computing**, **Cloud, Docker, Terraform, Kubernetes, Big Data (Hadoop)**
  - Using online video lectures
  - 15,000+ enrolled students in 100+ countries

Course Layout

Introduction	Jenkins, the DevOps way	Integrations	Advanced Jenkins usage
Introduction to Jenkins	Infrastructure as code and automation	Email integration	Jenkins slaves
Installation	Job DSL	Slack integration	Blue Ocean
Building a NodeJS app with Jenkins and Docker	Jenkins Job Pipeline	Github & Bitbucket integration	ssh-agent plugin
	Running build, test, deploy in docker	JFrog integration	Security best practices
		Custom API integration	authentication and authorization strategies
		Sonar integration	Onelogin integration

Course Objectives

- To be able to:
  - Use Jenkins to perform **Continuous Integration** within your Software Development Lifecycle
  - **Install Jenkins** using docker
  - Configure Jenkins “The DevOps way”, using **Docker**, **Jobs DSL** and **Jenkins Pipelines**
  - Use **plugins** to **integrate Jenkins** with popular development software
  - Configure the **authentication and authorization** options to tighten security on your Jenkins UI

Feedback and support

- To provide feedback or get support, use the discussion groups
- We also have a Facebook group called Learn DevOps: Continuously Deliver Better Software
- You can scan the following barcode or use the link in the next document after this introduction movie

Procedure Document

- Use the next procedure document in the next lecture to download all the resources for the course
- All resources are in a github repository
  - You can clone that git repository
  - You can download a zip file on the github website
- Repository URL: https://github.com/wardviaene/jenkins-course

Course Goals

Course Goals

- This course is a bit different than other Jenkins courses around
- It has all to do with DevOps
  - This course has a big focus on **automation**, **ownership** and **reusability**
  - The developers are given the **capability** to **automate** their **own** jenkins build scripts by using Jenkins Pipelines
  - Using docker, a closer dev-qa-staging-prod **parity** should allow you to build software the same way locally, as on the build servers

Course Goals

- In this course I focus on teaching you the **best practices** to **succeed** in **adopting a DevOps culture**
  - Not only by setting up a Continuous Integration tool
  - But by **empowering Developers** to make their own decisions on **how** the **tools** should be **built**
  - And by trying to make building a lot more **transparant** and easier to do, by making use of **isolated containers** with **known dependencies**

Course Goals

- A fair bit of this course is spend on **integrations**
    - Integrations will help you to **keep your infrastructure simpler**
    - You don’t want to spend time on managing software, you just want to be able to use it
    - By using managed services you can **focus on what’s important**
    - And you can keep your **team small and focussed** on improving delivering software

Course Layout

Introduction	Jenkins, the DevOps way	Integrations	Advanced Jenkins usage
Introduction to Jenkins	Infrastructure as code and automation	Email integration	Jenkins slaves
Installation	Job DSL	Slack integration	Blue Ocean
Building a NodeJS app with Jenkins and Docker	Jenkins Job Pipeline	Github & Bitbucket integration	ssh-agent plugin
	Running build, test, deploy in docker	JFrog integration	Security best practices
		Custom API integration	authentication and authorization strategies
		Sonar integration	Onelogin integration

Course Objectives

- To be able to:
  - Use Jenkins to perform Continuous Integration within your Software Development Lifecycle
  - Install Jenkins using a package manager or docker
  - Configure Jenkins “The DevOps way”, using Docker, Jobs DSL and Jenkinsfiles
  - Use plugins to integrate Jenkins with popular development software
  - Configure the authentication and authorization options to tighten security on your Jenkins UI

Introduction to Jenkins

What is Jenkins

- Jenkins is an open source continuous integration (CI) and continuous delivery (CD) tool written in Java.
- It's an automation server used to **build and deliver software projects**
- Jenkins was forked from another project called Hudson, after a dispute with Oracle
- A major benefit of using jenkins is that it has **a lot of plugins available**
- There is easier to use CI software available, but Jenkins is open source, free and still **very popular**

What is CI / CD

- **Continuous integration** (CI) is the practice, in software engineering, of **merging all developer working copies** to a shared mainline several times a day. (Wikipedia)

- **Continuous delivery** (CD) is a software engineering approach in which teams produce software in short cycles, ensuring that the software can be reliably released at any time. (Wikipedia)

- In Practice: **verify and publish** work by triggering automated builds & tests

  - For **every commit** or at least once a day

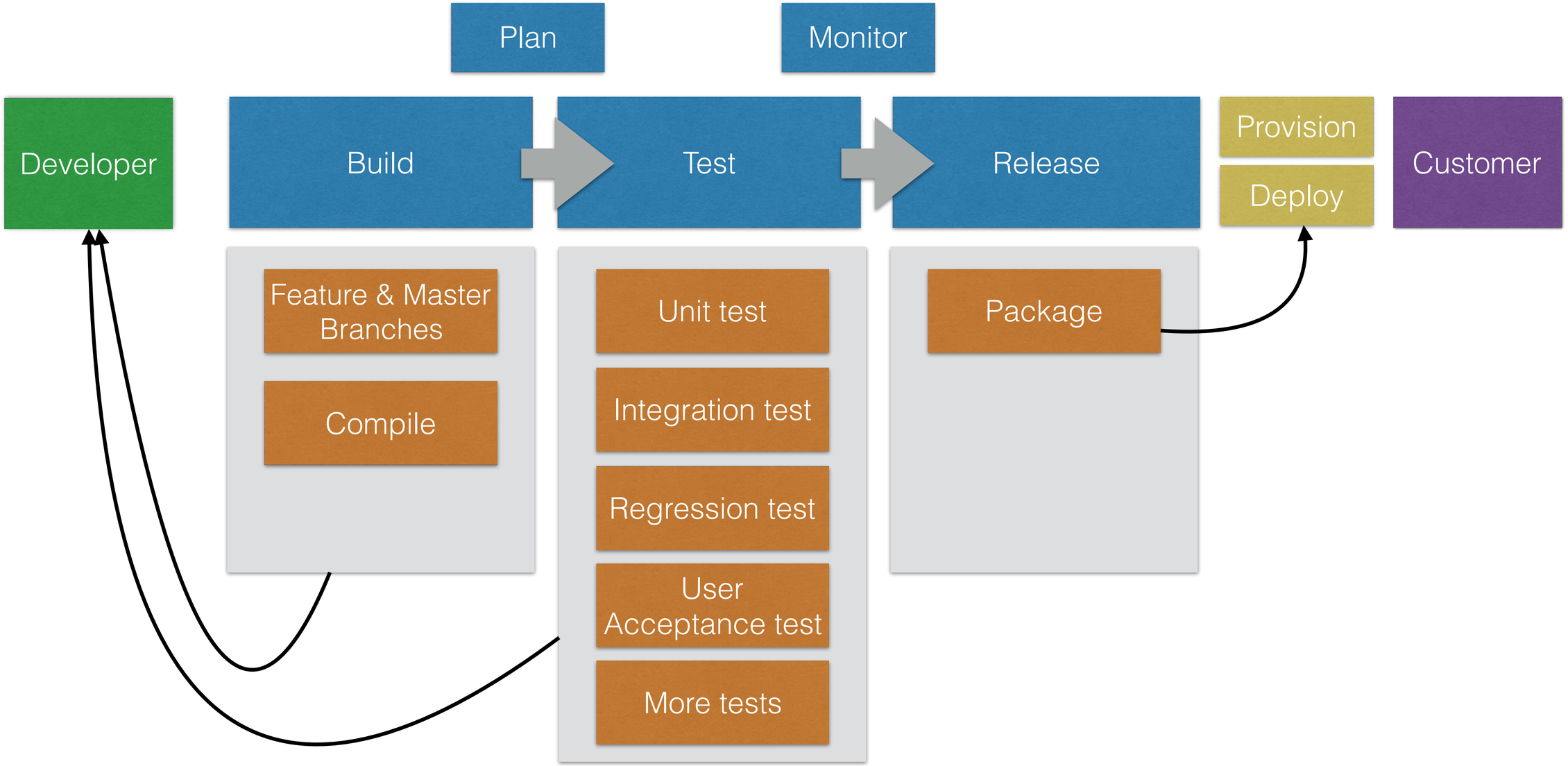
  - All developers should push their changes to a version control, which should then be built and tested - which can be done by Jenkins

  - **Jenkins doesn't merge code** nor it resolves code conflicts, that's still for the developer to do (using for instance git and merge tools)

Benefits

- Jenkins provides a **feedback loop** back to the developer to fix build errors
  - Research has shown that it's a lot **quicker** to have a developer **fix** the error **immediately** (when the code is still fresh in memory)
- Jenkins can **publish** every build of your software
  - This build already has gone through automated testing
  - When published and deployed to a dev/qa/staging server, you can advance the Software development lifecycle (SDLC) much quicker
    - The quicker you can go through an iteration of the SDLC the better

CI/CD within the SDLC



Jenkins Alternatives

- Self-hosted
  - Drone CI (Continuous delivery platform written in Go)
  - TeamCity (by Jetbrains)
- Hosted (as a service)
  - Wercker
  - CircleCI
  - CodeShip
  - SemaphoreCI
  - Amazon AWS CI/CD tools

Jenkins Installation

Installation methods

- On the Cloud using docker (works on AWS, DigitalOcean, Google Cloud, Azure)
- www.virtualbox.org can run ubuntu in a **VM locally**
- www.vagrantup.com together with virtualbox can spin up a running ubuntu instance in minutes (see my other DevOps course for an introduction to Vagrant)
- **Docker for windows** can run docker on a windows PC (https://docs.docker.com/docker-for-windows/)
- **Docker for mac** can run docker on a mac (https://docs.docker.com/docker-for-mac/)
- Jenkins is written in java, so you can also run it without docker on **any OS**
  - Still I recommend docker for this course
- Even when using linux and docker, you can still run **jenkins slaves on Microsoft Windows** to build applications that need a Microsoft Windows OS

Demo

Install Jenkins

DigitalOcean Coupon Code

- Get $10 in free DigitalOcean credits:
  - https://m.do.co/c/007f99ffb902

Docker

What is docker?
Why use docker for Jenkins?

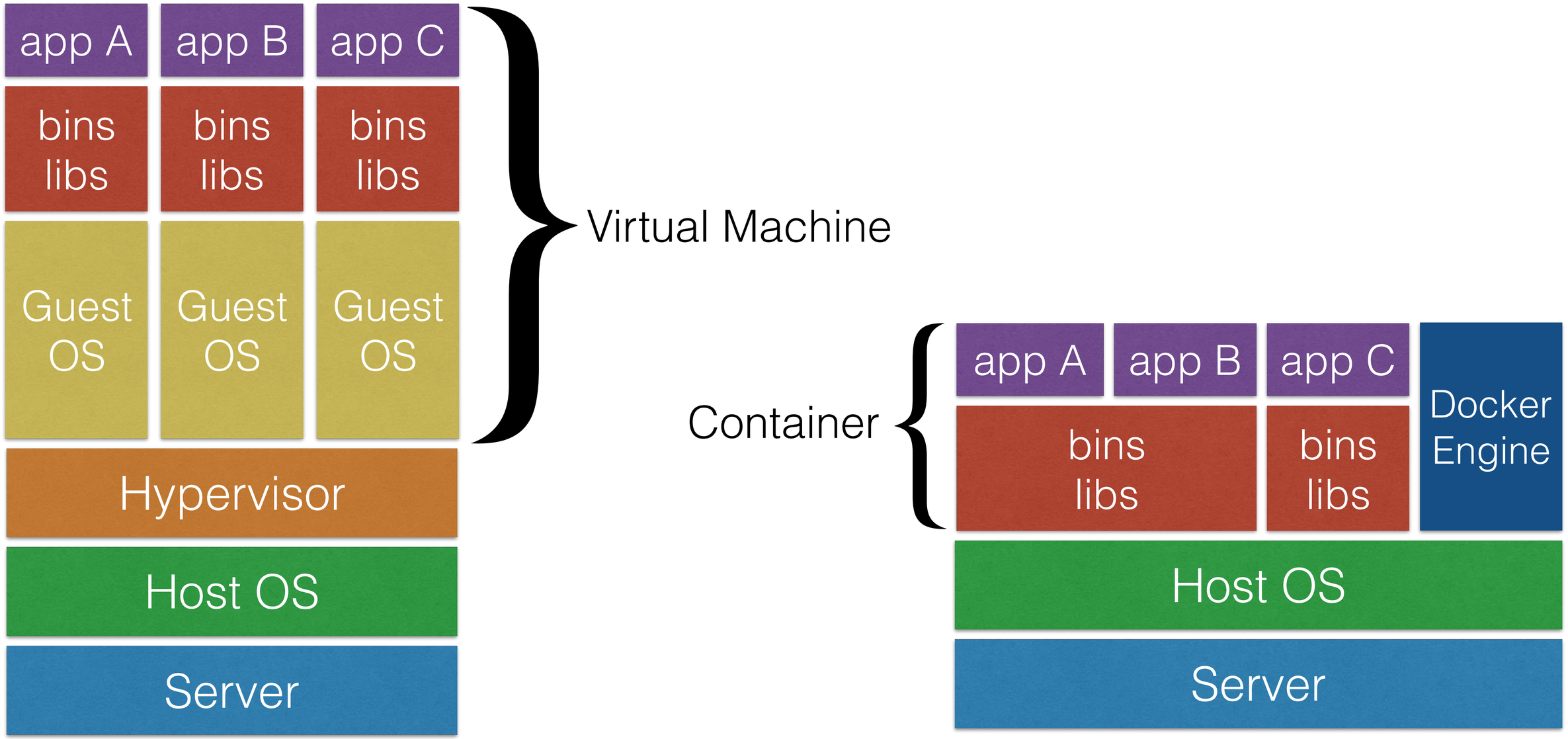
What is docker?

- Docker is the most popular **container software**
- **Docker Engine**
  - The Docker runtime
  - Software to make run docker images
- **Docker Hub**
  - Online service to store and fetch docker images
  - Also allows you to build docker images online

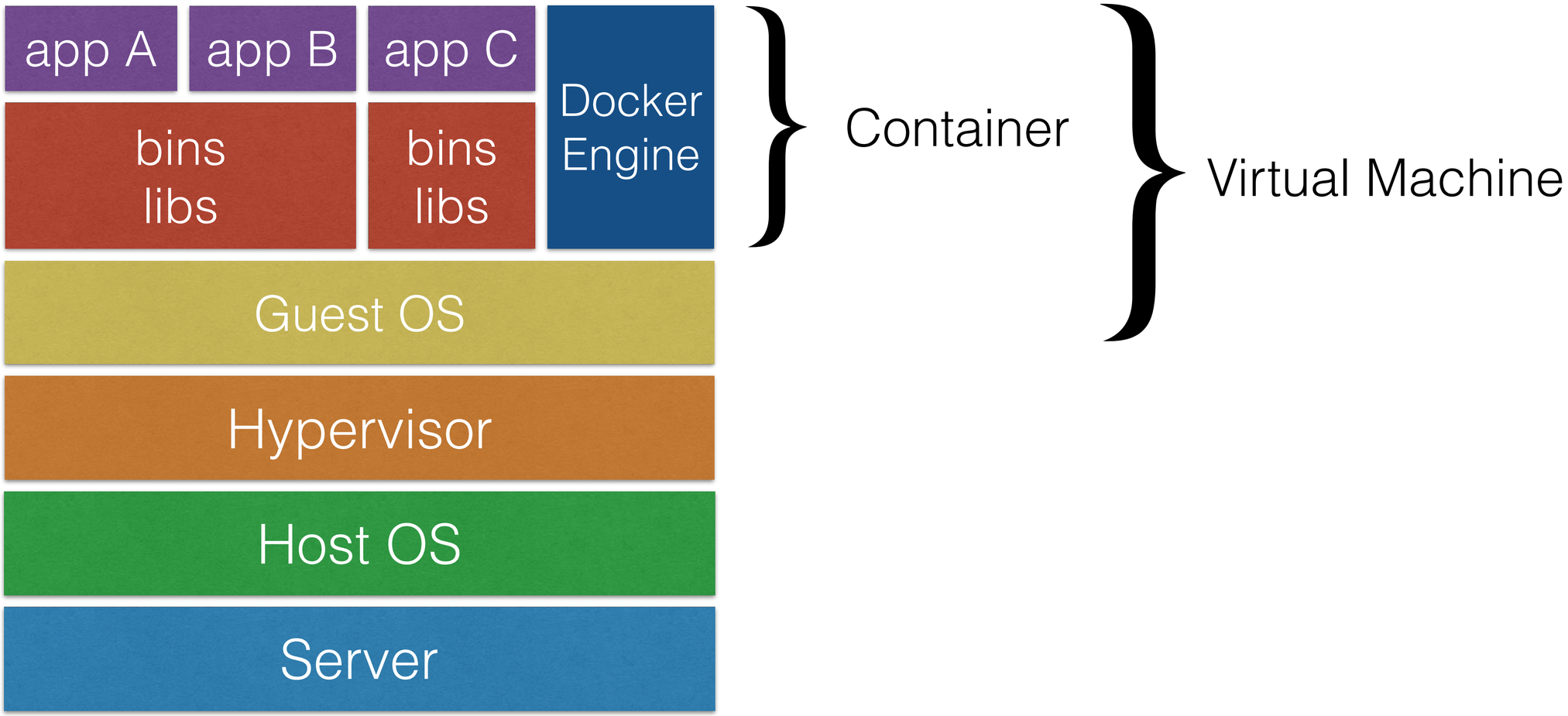
What is docker?

- Isolation: you ship a binary with all the dependencies
  - No more “it works on my machine, but not in production”
- Closer parity between dev, QA, and production environments
- Docker makes development teams able to ship faster
- You can run the same docker image, unchanged, on laptops, data center VMs, and Cloud providers.
- Docker uses Linux Containers (a kernel feature) for operating system-level isolation

Virtual Machines vs Containers



Containers on Cloud Providers



Why use docker to start Jenkins?

- Everyone will have the **same container image** of Jenkins
  - Regardless of your Operating System
  - You can run docker on an ubuntu on DigitalOcean, but also on a Mac, on Windows, etc
- You can **easily upgrade** jenkins by pulling the latest container image
- You can **modify** the jenkins **container** image by creating your own **Dockerfile**
  - Or modify the Dockerfile I have for jenkins in my github repository

Build and deploy the first app

What is Node.js

- Node.js is a **javascript runtime environment** which is:
  - **Open Source**
  - **Cross-Platform**
  - Used to execute javascript code **server side** instead of client side
  - Has an **event-driven** architecture capable of **asynchronous I/O**
    - This makes NodeJS capable of **responding very fast** to requests
    - e.g. it can immediately give a client the result and asynchronously handle database updates which generally take a longer time
    - This is one of the reasons why Node.js is used so much

Node.js users

- Node.js is used by:
  - Paypal (using Node.js for its customer-facing side of their web applications)
  - Linkedin (to deal with scale)
  - Uber (to process lots of data quickly, address errors on the fly - without restart, and active community that makes NodeJS better over time)
  - Yahoo, Mozilla, Netflix, New York Times, Medium

Why Node.js

- We use a Node.js example project because it's **relatively easy to understand**, even when you never have used it before
- It **doesn't need to be compiled**, so building the app doesn't take a lot of memory (unlike building java projects)
  - Which is handy if you just want to do this course on small instances
- Once you understand how to build a simple node.js project, you can use the knowledge to start building larger, more **complex projects**

How to build a Node.js app

- **Step 1: install dependencies (npm install)**
  - Downloads and installs all the dependencies
- **Step 2: run tests (npm test)**
  - Runs all the NodeJS tests
  - Typically if a test fails, the build fails and the developer(s) should be notified
  - I'll explain about slack/bitbucket integrations later on this course
- After step 1 and 2 you have an application that is ready to be started
- The only thing you'll be missing is a package: how do you package and distribute this project so it can be deployed on a server instance?

How to build a Node.js app

- **Step 3: package** the app in docker
  - You can create a container that includes the Node.js binaries and your project
  - Rather than creating a .zip, .jar, or .tgz package, you package a binary that includes all binaries and dependencies
  - That way you're sure that the code will **behave the same** on the production system as your dev/test/staging/qa machines
    - It gives you closer dev-prod parity
- **Step 4: distribute** the docker image
  - You can use a docker registry to upload your docker images
  - Those images can be set public (everyone can access them) or private
  - Examples: Docker Hub, Amazon AWS's Container Registry

Demo

Build and deploy the first app

Demo

Package and push to docker repository

Infrastructure as code and automation

Infrastructure as code and automation

- Jenkins allows you to use a web UI to input all the **build parameters**
- This leads to:
  - No proper **audit trail**
  - No easy **history of changes**
  - **Segregation** between Jenkins admins and developers
    - Users (often developers) will have to contact a Jenkins administrator to make changes
    - **Long lead times** for changes
  - Difficult to **backup and restore** (e.g. how to restore just one setting to how it was the day before)

Infrastructure as code and automation

- The solution is to **completely write jobs as code** and **save it in version control**
  - Version Control (git / subversion / mercurial) gives you a **history** and **audit trail**
  - Easy **roll back** to **older** versions of jobs and build instructions
  - Allows operations to give **more control to the developers**
    - Developers can **bundle the jenkins build instructions** with their own project repository (e.g. a **Jenkinsfile** in their project directory)
    - This is what a company that wants to embrace **DevOps** should do: **allow developers to control their own builds**

Job DSL and Jenkins Pipelines

- In the next sections, I'll cover 2 Jenkins capabilities to enable Jenkins to be a **true DevOps tool** within your software organization:

  - **Jenkins Job DSL**

    - Write **code** that creates and modifies jenkins jobs automatically

  - **Jenkins Pipeline (Jenkinsfile)**

    - **Bundle** the **build parameters** within the project

    - Allow **developers** to **change** jenkins build parameters

    - Enable **audit trail, history, rollbacks** using version control

Jenkins Job DSL

Jenkins Job DSL

- The Jenkins Job DSL is a plugin that allows you to define jobs in a programmatic form with minimal effort
- DSL stands for **Domain Specific Language**
  - You can describe jobs using a **Groovy based language**
    - Think about Groovy as a **scripting language** for the Java platform
    - It's similar to java, but **simpler**, because it's much more **dynamic**

Jenkins Job DSL

- The Jenkins Job DSL plugin was designed to make it **easier to manage jobs**
  - If you don't have a lot of jobs, using the **UI** is the **easiest** way
  - When the **jobs grow**, maintaining becomes **difficult** and a **lot of work**
- The Jenkins Plugin solves this problem, and you get a lot of extra **benefits**:
  - Version control, history, audit log, easier job restore when something goes wrong

Jenkins Job DSL

- An example (npm install)

```
job('NodeJS example') {

    scm {
        git('git://github.com/wardviaene/docker-demo.git') {  node ->
            node / gitConfigName('DSL User')
            node / gitConfigEmail('jenkins-dsl@newtech.academy')
        }
    }

    triggers {
        scm('H/5 * * * *')
    }

    wrappers {
        nodejs('nodejs') // this is the name of the NodeJS installation in
                         // Manage Jenkins -> Configure Tools -> NodeJS Installations -> Name
    }

    steps {
        shell("npm install")
    }

}
```

Demo

Job DSL with node example

Demo

Job DSL with node & docker example

Jenkins Pipelines

Jenkins Pipelines

- Jenkins Pipelines allow you to write the **Jenkins build steps in code**
  - **1) Build steps** allow you to write build (compile), test, deploy in code
  - **2) Code** means you can put this code in version control

It's about automating this cycle:

Jenkins Pipelines vs Job DSL

- **How are Jenkins Pipelines different than the Jenkins Job DSL?**
  - They both have the capability to write all your CI/CD in code
  - The difference is in implementation in Jenkins
  - Jenkins Job DSLs creates **new jobs** based on the code you write
  - The Jenkins Pipelines is a **job type**, you can create a Jenkins pipeline job that will handle the build/test/deployment of one project

Jenkins Pipelines vs Job DSL

- Can I now start using pipelines and forget everything about Jenkins Job DSL?
  - You could use **Jenkins Job DSLs** to create **new pipeline jobs**
  - Until now we've only created **freestyle projects** with the Jenkins Job DSL
  - Another possibility would be to use an **"Organization folder"**, which is a feature of Jenkins Pipelines to **detect** the project repositories, removing the need to add new jobs
    - It'll really depend on your needs for what option you'll need to go

Jenkins Pipelines

- The pipeline is a **specific job type** that can be created using the Jenkins UI or the Jenkins Job DSLs

- You can choose to write the pipeline in **Jenkins DSL** (declarative pipeline) or in **groovy** (scripted pipeline)

  - Groovy is a **scripting language** for the **java** platform, syntactically very similar to Java and it runs in the JVM (Java Virtual Machine)

  - Under the hood the Jenkins DSL is **interpreted by groovy**

Jenkins Pipelines Example

```
node {

   def mvnHome

   stage('Preparation') {
      git ‘https://github.com/wardviaene/java-demo.git'
      // Get the Maven tool.
      // ** NOTE: This 'M3' Maven tool must be configured
      // **       in the global configuration.
      mvnHome = tool 'M3'
   }

   stage('Build') {
      // Run the maven build
      if (isUnix()) {
         sh "'${mvnHome}/bin/mvn' -Dmaven.test.failure.ignore
clean package"
      } else {
         bat(/"${mvnHome}\bin\mvn" -Dmaven.test.failure.ignore
clean package/)
      }
   }

   stage('Results') {
      junit '**/target/surefire-reports/TEST-*.xml'
      archive 'target/*.jar'
   }

}
```

- Node: influence on what jenkins worker node the job will be ran (here: any node)
- def: allows you to declare variables
- Stage: defines a building stage: build, test, or deploy
  - Conceptually distinct step
  - Used by other plugins in Jenkins later on to visualize the stages of a job
    - e.g. clean->build->test->publish

Demo

Jenkins pipeline with nodeJS example, with docker

Build, test, run everything in docker

Docker Pipeline plugin

- The docker pipeline plugin, let's you **spin up any container** within your pipeline
    - I just used the same plugin in the previous demo, to **build/push** images
- You can not only build new containers, but also **run existing containers**
    - This is useful for when you don't want to bundle development tools with your production container, but you still want to run **all stages** in an isolated environment
        - You can **build/test your application first** with an **existing** container with all the development tools
        - As a next step, you can **build a new container**, much tinier, with only the **runtime environment**

Docker Pipeline plugin

- Spinning up new docker containers lets you **bring in any new tool**, easily
    - You can specify exactly what **dependencies** you want, **at any stage** in the job
    - You can **start a database** during the test stage to run tests on
    - After the database tests have been concluded, the **container** can be **removed**, together with all the data
    - Next time you run a new database container during tests, you'll have a brand new container again
    - This also works for **multiple builds at the same time** (think multiple git branches), every build has its own database container

Demo

Jenkins pipeline tests with docker

Email Integration

Email integration

- The goal is to **alert** the developer of a broken build **as soon as possible**
  - The **earlier** you give a developer feedback of something that is wrong, the better
  - This increases the **productivity** of the developer team tremendously
    - The **longer** it takes for a developer to **know** he needs to fix a bug, the **more time** it'll take to resolve
    - The developer will able to fix the code the quickest, when the code a developer wrote is still **fresh** in **his memory**

Email integration

- **Every time** when a developer **commits** a change in version control, the build in jenkins should start

- Changes from version control can either be **pulled** or **pushed**:

  - **Pull**: Jenkins polls the version control **every x minutes**

  - **Push**: The version control system (e.g. Github, Bitbucket) will send a **notification** (push) to Jenkins (using http requests)

    - You can use the Github Plugin and Bitbucket to configure the push notifications

Demo

E-mail notifications

Slack Integration

Slack Integration

- Slack is a **chat and collaboration** tool, comparable with Atlassian's HipChat
- It's a new and popular team **communication tool**, that is much better than using Skype or older enterprise chat tools for collaboration
- Slack (or hipchat) is better than its competitors, because it allows you to **integrate** all your **tools** within slack, to avoid switching between apps
    - It can give you a realtime "war room" to collaborate on problems
    - This is also called "ChatOps"
    - If you're familiar with IRC (Internet Relay Chat), it's possible that you already do this since the nineties

Slack Integration

- ChatOps is a **collaboration mode** that **connects** people, tools, processes, and automation in a transparant workflow [1]

- It allows you to do **conversation driven collaboration**, while having your **tools integrated** and keeping you up to date of the state of your systems

- For example:

    - You’re in a team chat with 5 people, when the following happens

[1] Source: https://www.atlassian.com/blog/software-teams/what-is-chatops-adoption-guide

Slack Integration

- 10:05 AM: You receive a message in the chat that a backend service is malfunctioning
- 10:05 AM: OpsGenie (a pager tool) puts a message on the chat that Bob has been paged
- 10:10 AM: OpsGenie sends a message that Chris has been pages, because Bob hasn't been responding
- 10:11 AM: OpsGenie puts another message on the chat that Chris has acknowledged the problem
- 10:15 AM: You tell Chris on the chat that the backend service has been misbehaving since yesterday and that it might be because of a new experimental feature that has been pushed late last night

Slack Integration

- 10:15 AM: Chris acknowledges and says he'll look for the cause
- 10:45 AM: Chris finds a solution for the problem and commits his changes
- 10:45 AM: Bitbucket puts a message in the room that a code change has been committed and pushed to the git repository
- 10:55 AM: Jenkins notifies the channel that a new build has been pushed to the staging server
- 10:56 AM: Jenkins notifies the channel that the staging version was promoted to production

Slack Integration

- 11:00 AM: The monitoring system puts a message on the channel that all systems are green again

- 11:30 AM: Bob comes back from a meeting and sees the history of the chat, the actions that have been taken, the commit that been done and that all systems are OK again

- 12:00 PM: The rest of the engineers see the same history, everyone is up-to-date with what happened that morning

Slack Integration

- This is an example of how **ChatOps** could work, relieving the team of a lot of **inefficient emails** that would've been send around
- If you want to use ChatOps, your task now is to **integrate Jenkins** with your collaboration tool, in a way that you can see the relevant messages in your channel
- I'll show you how to do this with Slack

Demo

Slack integration

GitHub / Bitbucket integration

GitHub / Bitbucket Integration

- Up until now I've always added git repositories manually
- If you have a lot of repositories, you don't want to add every single repository manually
- It is then more interesting to have Jenkins auto-detect new repositories
- A developer could then create a new repository for a new (micro) service, add a Jenkinsfile, and the project will automatically be built in Jenkins

GitHub / Bitbucket Integration

- The implementation differs from which version control provider you're using
    - When using GitHub: the **GitHub Branch Source plugin** will scan **all the branches and repositories** in a **GitHub organization** and build them via Jenkins Pipelines
    - When using Bitbucket: the **Bitbucket branch source plugin** can scan **team/project folders** and automatically projects

Demo

GitHub integration

Demo

Bitbucket integration

JFrog Integration

JFrog Integration

- In the previous demos I've always submitted the **Docker image** to the docker registry (Docker Hub)
- This resulting image is in Jenkins called "The **artifact**"
- It's the resulting **binary** from a build
- It can be a docker image, or a .jar file, a .tgz/.zip file, really anything
- These artifacts, the result of your build, you want to **store** somewhere
- **JFrog Artifactory** is a product that can store for you the artifacts resulting from a build

JFrog Integration

- You can either **download** Artifactory for free and run it yourself, or you can use their hosted version
- In the demo I'll use the **hosted** version
- It's **best practice** to **store** all the **artifacts** of the builds that are getting deployed
  - If you need to roll back, you have the artifact already available
  - You are 100% sure about the binary if you're promoting the same version from dev to staging or from staging to production
  - No lengthy rebuilds

JFrog Integration

- JFrog integration is done using a **JFrog Plugin**
- The JFrog plugin allows you to add **extra steps** to your Jenkinsfile
- One of the last steps of the build will be to send the artifact to JFrog
    - In this step, you can put a conditional, to only do this for develop/ master branch, not for feature branches

Demo

JFrog integration

Custom API Integration

Custom API Integration

- Sometimes you want to integrate an API, but there is **no plugin available**
- Even if there is a plugin available, it can **lack the features** you want
    - You might want to get more information from an endpoint (e.g. the full JSON from the request)
    - You want to do a POST or PUT request on the API
- One solution is to write your **own Jenkins Plugin**
    - But this requires a lot of **effort**
    - And is not always feasible, as a developer, you might not have access to Jenkins Plugins

Custom API Integration

- Another solution is to use functionality in the Jenkins pipelines to do **http requests**
- You only need one plugin: the **http request plugin**
- This plugin can do a **generic HTTP request** on any API
- It's up to you (the developer / SysOps / DevOps) to write the code to do the requests and interpret them
- This is where groovy becomes handy, because **groovy allows you to do scripting within the Jenkinsfile**

Demo

Custom API integration

Sonarqube integration

Sonarqube

- Sonarqube **continuously inspects** your software project on **code quality**
- It can report on:
  - Bugs (code issues)
  - Vulnerabilities (security issues)
  - Code smells (maintainability related issues)
  - Technical debt (estimated time required to fix)
  - Code coverage (test coverage of code)

Sonarqube

- Sonarqube is a **very popular** piece of software that is often integrated with Jenkins
- In Jenkins it's just a build step
  - the code needs to be scanned by the sonar-scanner
  - the sonar-scanner sends its results to the Sonarqube server
    - The sonarqube server needs to be installed
    - Sonarqube server also uses a database to maintain its state

Sonarqube

- One possible solution:
    - Install database and sonarqube as a docker container on the master node
    - We can use docker-compose to manage the containers
    - Docker-compose is a handy tool from Docker that can spin up containers based on a container definition in yaml format
    - We'll have 3 containers, a Jenkins container, a database container, and sonarqube

Demo

Sonarqube install

Demo

Sonarqube

Advanced topics

Jenkins Slaves

Jenkins Slaves

- Currently, only one node (one droplet) is hosting the jenkins web UI and doing all the builds
- In production environments, you typically want to host the **Jenkins web UI on a small master node**, and have **one or more worker nodes** (Jenkins Slaves)
- Using worker nodes, you can easily **expand your build capacity**
- Typically one worker has one or more **build executors** (building slots)
  - If a Jenkins node has **2 executors**, only **2 builds** can run in **parallel**
  - Other builds will get **queued**

Jenkins Slaves

- **Static** or **manual** scaling:
    - You can have more workers during working hours (or no workers outside working hours)
    - You can add more workers ad-hoc, when necessary
        - During periods when a lot of code is created
        - During periods when developers have to wait long for for their builds to be finished
            - i.e. the jobs stay a long time in the queue

Jenkins Slaves

- Dynamic worker scaling
    - You have plugins that can scale Jenkins slaves for you
        - **The Amazon EC2 Plugin**: if your jenkins build cluster gets overloaded, the plugin will start new slave nodes automatically using the AWS EC2 API. If after some time the nodes are idle, they'll automatically get killed
        - **Docker plugin**: This plugin uses a docker host to spin up a slave container, run a jenkins build in it, and tear it down
        - **Amazon ECS Plugin**: Same as docker plugin, but the host is now a docker orchestrator, the EC2 Container Engine, which can host the docker containers and scale out when necessary
        - **DigitalOcean Plugin**: dynamically provisions droplets to be used as jenkins slaves

Jenkins Slaves

- Builds can be executed on specific nodes
  - Nodes can be labeled
    - e.g. “windows64-node”
  - Builds can then be configured to only run on nodes with a specific label
    - This can be configured in the UI or using the Jenkinsfile

```
node(label: ‘windows64-node’) {
   stage('build') {
     […]
   }
}
```

Jenkins Slaves Benefits

- **Reduced cost**: only have the capacity you really need
- Slaves are easily **replaceable**: if a slave crashes, in can be spun up again
- The master can run on a separate node that isn't affected by the CPU/Memory load that builds generate
    - i.e. the UI will always be responsive
- You can **respond to sudden surges** in builds, capacity can be added on the fly
    - Even with manual scaling, you can quickly spin up a new machine as a Jenkins slave

Jenkins Slaves

- It’s important to standardize your Jenkins slaves
  - **Don’t manually install tools** on the slaves
    - Use **Plugins** to provide tools (NodeJS, Docker, Java, Maven)
    - Use **Docker** to provide images that can build jobs, and use the **Docker pipeline** plugin to execute builds in a specific docker image
  - A slave should be **disposable**, you should be able to throw it away and start it again from scratch

Jenkins Slaves

- There are **2 solutions** I'll demo:
  - **Master node connects to slave** over SSH
    - I'll set up a new machine and let the master connect to the slave
  - **Slave node connects to master** over JNLP
    - The slave will initiate the contact
    - Good solution for windows slaves
    - Useful if the slave is behind a firewall

Demo

Slave demo over ssh

Demo

Slave demo using jnlp

Blue Ocean

Blue Ocean

- Blue Ocean is a new frontend for Jenkins
  - Built from the ground up for Jenkins Pipeline
  - Provided as plugin
  - Still compatible with freestyle projects
  - Should eventually replace the normal Jenkins UI over time

Blue Ocean

- New features:
  - **Sophisticated visualizations** of the pipeline
  - A **pipeline editor**
  - **Personalization**
  - **More precision** to quickly find what's wrong during an intervention
  - Native **integration** for **branch** and **pull** requests

Demo

Blue Ocean

ssh-agent

ssh-agent

- When you work with Jenkins Slaves, those **slaves** also need **access** to repositories
    - A **mistake** I often see is that people **customize** the **software** on their slave
    - **Private** keys and **credentials** often end up on the **slaves**
    - This makes it **more difficult to scale** out slaves: adding another slaves suddenly means manually copying over credentials and private keys

ssh-agent

- For ssh keys, the solution is to use an **ssh-agent**
- The ssh-agent will run on the **master**, and will contain the **private keys** that are necessary to authenticate to the external systems you need access to
    - Predominantly, this is a **GitHub/Bitbucket private key** to get access to the repositories
- When you need access to a system or a git repository within the Jenkinsfile, you can wrap the ssh-agent around your code, to be able to authenticate to the systems
    - ssh-agent uses the same keys stored within your credentials

ssh-agent

```
node {
  stage(“do something with git") {
  sshagent (credentials: ['github key']) {
    // get the last commit id from a repository you own
    sh “git ls-remote -h --refs git@github.com:wardviaene/jenkins-course.git master |awk '{print $1}’”
  }
}
```

Demo

ssh-agent

Security

Best practices

Security

- Best practices for Jenkins:
    - Try to keep your Jenkins shielded from the internet: using firewall rules and behind a **VPN**
        - You'll need to **whitelist** the bitbucket/github IP addresses for push requests
        - In the past there were **security vulnerabilities** discovered that can be exploited without being logged in, so it's better to keep Jenkins shielded away from the internet

Security

- Best practices for Jenkins:
    - Keep your Jenkins **up-to-date**
        - Always upgrade to the latest version
            - Use the **lts** (long term support) edition if you want to be on a stable version
            - If you're using docker, use the lts or latest tag, and do a docker (or docker-compose) pull, and restart the container
            - Also read the **Changelog**
        - Always keep the **plugins up to date**

Security

- Best practices for Jenkins:
  - Configure **authentication / authorization**
    - If you don’t want to give administrator access to your users, make sure that they can’t execute scripts that give them elevated permissions
      - Keep in mind that administrators can decrypt credentials
    - Use Onelogin (SAML) / LDAP / centralized directory for users
    - Don’t use weak passwords / logins

Security

Authentication & Authorization

Security

- **Authentication:**
  - “The process or action of verifying the identity of a user or process.” [1]
  - Basically, verifying the credentials of the user: the username (or email) and password
- **Authorization:**
  - “Authorization is the function of specifying access rights to resources related to information security and computer security in general and to access control in particular” [1]
  - Once a user is authenticated, what does he have access to?

[1] Source: Wikipedia

Security

- I'll cover authentication in the next section, by using Onelogin as our "authentication / identity provider"
- For authorization, you have a few options:
  - Anyone can do anything (not recommended)
  - Anyone who is logged in can do anything
  - Matrix based authorization (just a big table to give users access)
  - Role Strategy plugin (for more granular control, like projects and slave access)

Security

- What do to when you lock yourself out?

```
$ docker stop jenkins
$ # edit /var/jenkins_home/config.xml
$ docker start jenkins
```

Option 1:

```
Make sure you have:
<useSecurity>false</useSecurity>

And remove all
<authorizationStrategy> references
```

Option 2:

```
<authorizationStrategy
class="hudson.security.ProjectMatrixAuthorizationStrategy">
<permission>hudson.model.Hudson.Administer:YOUR-USER</permission>
</authorizationStrategy>
```

Demo

Authorization demo

Security

Onelogin integration

Security

- Rather than managing users locally, it's best to manage users **elsewhere**
- All enterprise companies use a central user database, often a **directory service** like Active Directory or LDAP
  - If you already have this setup, then it's quite easy for you
  - At the "Configure Global Security" page, select **LDAP** and input the settings provided to you by the AD group
- If you're not in an enterprise environment, you'll either have to setup LDAP yourself, or go for a **hosted solution**

Security

- One of the best hosted solutions is onelogin.com
- It's used by a lot of companies to manage their users directory
- Onelogin can also be **linked to Jenkins**, to let **onelogin handle** the **authentication** process
  - You link it through **SAML**
    - Security Assertion Markup Language (SAML, pronounced sam-el) is an XML-based, open-standard data format for exchanging authentication and authorization data between parties (wikipedia)

Security

- Once SAML has been setup, Onelogin will handle the **authentication** part
- When you want to add more users, you'll have to add more users to Onelogin
  - Onelogin can provision users from other sources, like Google
  - Onelogin also provides multi-factor authentication
- Unfortunately Onelogin is not cheap (and suffered from some data breaches in the past), but it's still a very straightforward and secure way to manage your users

Demo

Onelogin

Congrats!