

# CloudBees Jenkins Platform: Certification Training

## 3 - Advanced Usage of Jenkins

# Table of Contents

3 - Advanced Usage of Jenkins	1
Journey Description	1
Enable and Configure Jenkins Security	1
Internal LDAP overview	1
Enabling Authentication	2
Trying to set up Authorization policy	4
Enabling (Seriously) Authorization policy	7
Accounting: Privacy and Credentials	9
Distribute Jenkins Builds	14
Adding an SSH agent	14
Validating SSH	14
Adding SSH Agent in Jenkins	15
Adding JNLP Agent in Jenkins	18
Reconfiguring Master	21
Reconfiguring Jobs	22
Running Distributed Builds	25
Job Promotion	27
Organize your jobs with Folders and Views	31
Creating Folder	31
Copy Folder	32
Folders on FileSystem	34
Track your artifacts and dependencies with fingerprinting	35
Jenkins Command Line Interface	37
Requirements	37
Step 1. Download the client jar	38
Step 2. Run the help command	38
Step 3. Run a build	39
Step 4. Managing Agents with the CLI	39
Step 5. Creating a 3rd agent with the CLI	39
Manage Jenkins using its REST API	40
Launch a Job with the REST API	41
Journey Summary	41

## 3 - Advanced Usage of Jenkins

### Journey Description

**IMPORTANT**

This lab requires having covered the [Lab 2](#)

The goal of this Journey is to cover advanced usage of Jenkins Open Source, still using the same application.

We will cover:

- Enabling and Configuring Jenkins Security
- Distribute Jenkins Builds
- Using Job Promotion
- Organizing your jobs with Folders and Views
- Tracking your artifacts and dependencies with fingerprinting
- Managing Jenkins using the REST API and the Command Line client

### Enable and Configure Jenkins Security

The goal of this exercise is to set up a basic security model on your Jenkins instance and manipulate Credentials.

### Internal LDAP overview

The Lab instance contains a Directory Service (OpenLDAP) that we will use as the authentication source for Jenkins.

**TIP**

OpenLDAP is not the only Directory Service system that Jenkins can use.

This LDAP instance comes **pre-populated** with some data:

- It contains:
  - 3 users: butler, colleague and contributor
  - 3 groups: jenkins-admin, jenkins-developer, jenkins-users

**NOTE**

This LDAP is **only** for demonstration purposes:

- No LDAP\*S\* (TLS) support
- Anonymous binding to the LDAP server
- No Replicas
- It **tries** to follow [RFC2798](#)

## Enabling Authentication

First, we will enable Security and Authentication on Jenkins, based on the LDAP.

- Start by browsing to the **Configure Global Security** page of [Jenkins](#):
  - **Enable security** by checking the appropriate checkbox
  - Select LDAP as **Security Realm**
    - Use this value for the **Server** field: `ldap://ldap-service:389`
  - Click on the **Advanced Server Configuration...** button in the LDAP configuration section, and set the following settings:
    - **root DN**: `dc=ldap,dc=cloudbees,dc=training,dc=local`
    - **User search base**: `ou=people`
    - **User search filter**: `uid={0}`
    - **Group search base**: `ou=groups`
    - **Group search filter**: `cn={0}`
    - **Group membership**: Search for LDAP groups containing user
    - **Group membership filter**: `member={0}`
    - Set **Manager Password** empty if your webbrowser inserted a random password
    - **Display Name LDAP attribute**: `displayname`
    - **Email Address LDAP attribute**: `mail`

LDAP

Server

Server

root DN

☐ Allow blank rootDN

User search base

User search filter

Group search base

Group search filter

Group membership

☐ Parse user attribute for list of LDAP groups

☒ Search for LDAP groups containing user

Group membership filter

Manager DN

Manager Password

Display Name LDAP attribute

Email Address LDAP attribute

Delete

Figure 1. LDAP Security Realm configuration

- Before saving, we want to test the LDAP settings:
  - Click on the button **Test LDAP settings** in the LDAP configuration section

- A panel named **Test LDAP settings** should open. Set both **User** and **Password** field to the value `butler`:

Figure 2. Test LDAP settings

- Click on the button **Test** to validate your settings. You should see the following output (if not the case, check your settings based on the error):

Figure 3. LDAP settings validated

- You can now **Save** this configuration to enable Authentication and go back to the **Jenkins dashboard**:
  - A new button **log in** appeared on the upper right.
  - Use it to identify yourself as butler (password is same as username):

Figure 4. Log In as Butler

- You are now logged as the **Butler** user, authentication works.
  - Verify that the buttons in the upper right of the screen print your "display name" and **logout** link.

- Browse to your user configuration page to check your settings:

**TIP**

Some fields have been populated using LDAP data. Updating **only** changes it inside the Jenkins database, not in the LDAP Directory Service.

The screenshot shows the Jenkins user configuration page for the 'Butler' user. The left sidebar contains links for People, Status, Builds, Configure (circled in red), My Views, and Credentials. A red arrow points from the 'Configure' link to the 'Full Name' field. The 'Full Name' field contains 'Butler'. The 'Description' field is empty. The 'API Token' section has a 'Show API Token...' button. The 'Credentials' section has a message: 'Credentials are only available to the user they belong to'. The 'E-mail' section has a field for 'E-mail address' containing 'butler@localhost.local'. Below this is a note: 'Your e-mail address, like joe.chin@sun.com'. The 'Extended Email Job Watching' section has a message: 'No configuration available'.

Figure 5. User configuration for Butler

## Trying to set up Authorization policy

- Log out from "Butler" user. Don't notice anything on the left-menu ?
  - Even if you are unauthenticated, you can still access **Manage Jenkins**

**NOTE**

Current (and default) authorization policy is **Anyone can do anything**.

- Staying unauthenticated, browse to the security page

**IMPORTANT**

If you have an error message (in red) on the **LDAP** section, click on the **Advanced Server Configuration...** button, and set the **Manager DN** and **Manager password** fields to empty value

These fields can be populated if you are using the "autosave/autofill" functionality of your web-browser.

Auto-filling forms on the web browser without requiring any action from the user is not recommended: [Autofill on Chrome and Safari Can Give Hackers Access to Your Credit Card Info](#)

- Set the **Authorization** policy to **Logged-in users can do anything**
- **Save** this new configuration; you have been kicked out of the Jenkins Management system.
  - You can still view jobs, but cannot Configure or Launch Build:

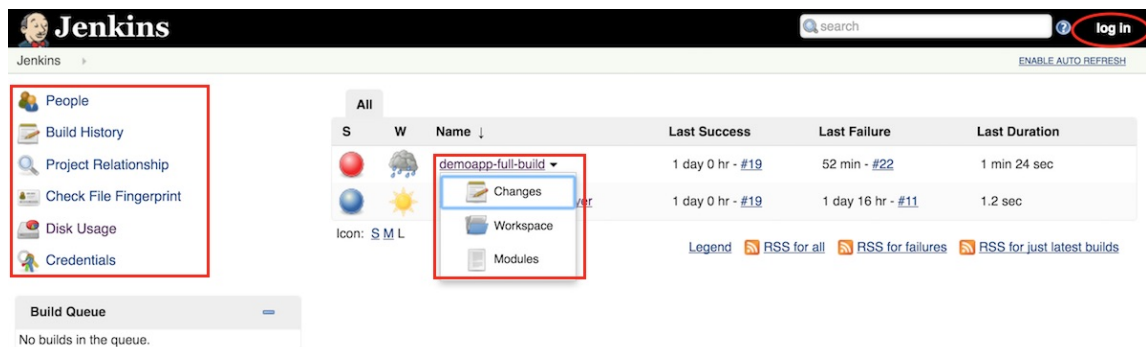


Figure 6. Dashboard for non logged user

- Time to login again as butler
  - The Jenkins Management page is now allowed.
- It is fine, but this policy is only for a small set of users in a trusted environment.
  - We want to separate rights based on group membership so that Jenkins can be scaled for the organization's growth.
- Browse again to the **Global Security Configuration:** page.
  - Enable the **Project-based Matrix Authorization Strategy** policy:

#### Authorization

- ☐ Anyone can do anything  
☐ Legacy mode  
☐ Logged-in users can do anything  
☐ Matrix-based security  
☒ Project-based Matrix Authorization Strategy

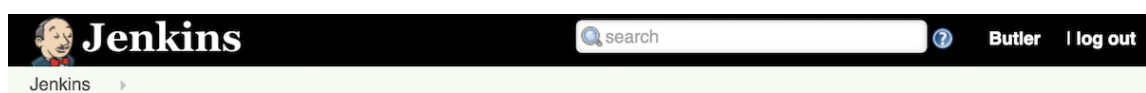
User/group	Overall	Credentials					Agent							
	Administer	Read	Create	Delete	Manage	Domains	Update	View	Build	Configure	Connect	Create	Delete	Disconnect
Anonymous	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

User/group to add:

Figure 7. Enabling Project-Matrix Policy with default settings

- Save the configuration using the default settings
  - You are now stuck; you can not move beyond the following screen:

**TIP** Read again the current section title: "**Trying** to set up..."



## Access Denied

butler is missing the Overall/Read permission

Figure 8. Butler kicked out of Jenkins

In order to get access to the Jenkins configuration pages again, we need to disable the security.

#### IMPORTANT

We are going to disable security from the Jenkins machine, using the command line to edit the Jenkins `config.xml` file.

- Using the [Devbox](#) command line:
  - Spawn an interactive shell line inside jenkins
  - Browse to the `${JENKINS_HOME}` directory
  - Verify the content of the `config.xml` file
  - Use the `sed` command shown below to disable security
  - Verify **again** the `config.xml`
  - Restart Jenkins using `docker restart` and use `docker logs` to control restarting status



```
# Spawn a shell in the Jenkins machine
cloudbees-devbox $ docker exec -ti jenkins /bin/bash

# Browse to the JENKINS_HOME directory
bash-4.3$ cd /home/jenkins

# Show content of the config.xml file
bash-4.3$ cat config.xml
...

# Show status of the "Enable Security" setting
bash-4.3$ grep useSecurity config.xml
...

# Show status of the "Authorization Strategy" setting
bash-4.3$ grep authorizationStrategy config.xml
...

# Set the XML attribute useSecurity to false in config.xml
bash-4.3$ sed -i \
    's#<useSecurity>true#<useSecurity>false#g' \
    config.xml

# Disable security by deleting the lines containing "authorizationStrategy"
# in the config.xml file
bash-4.3$ sed -i \
    '/authorizationStrategy/d' \
    config.xml

# Validate our changes
bash-4.3$ cat config.xml
...

# Exit the Jenkins machine
bash-4.3$ exit
exit

# Restart Jenkins to apply the configuration change
cloudbees-devbox $ docker restart jenkins
```

- Go back to the Jenkins homepage:
  - No more annoying message
  - Security is disabled
  - You have access to the **Manage Jenkins** page again !

## Enabling (Seriously) Authorization policy

- Re-enable Security from the Jenkins Configuration pages
- Configure LDAP settings **again** from the [previous exercise](#)
- Log-in as butler

# IMPORTANT

We want to apply the following rights:

- Non authenticated users (Jenkins system's **anonymous** user):
  - No access to Jenkins
- Authenticated users, members of **jenkins-users** LDAP group:
  - Can view Jenkins dashboard and non-private jobs
- Authenticated users, members of **jenkins-developers** LDAP group:
  - Can Launch, Create/View/Update/Delete (CRUD), Configure, Move jobs
  - Can CRUD Credentials
- Authenticated users, members of **jenkins-admin** LDAP group:
  - Can administer Jenkins

- Select the **Project-based Matrix Authorization Strategy** policy
  - Use the **User/group to add** field add your 3 groups

# TIP

A tiny icon will appear to tell you that Jenkins has a mapping to the configured LDAP's groups:



- Select the right permissions using the permissions table below

# TIP

Drag your cursor over each permission to display a description popup. Read **Job:Discover**, **Job:Cancel** and **Run:Update** as mandatory examples.

Table 1. Groups, Users and Permissions summary

Matrix Group/User	Members	Permissions (section, checkbox)	
anonymous	None (unauthenticated users)	None	None
jenkins-users	butler, colleague, contributor	Overall Job View	Read Read Read
jenkins-developers	butler, colleague	Overall Credentials Credentials Credentials Credentials Job Job Job Job Job Job Job Run Run Run View View View View	Read Create Delete Update View Build Cancel Configure Create Delete Discover Read Workspace Delete Replay Update Configure Create Delete Read
jenkins-admin	butler	Overall	Administer

- Check your settings with the following screenshot, and **Save** the configuration:

Project-based Matrix Authorization Strategy

User/group	Overall	Credentials	Agent	Job	Run	View	SCM
Anonymous	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
jenkins-users	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
jenkins-developers	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
jenkins-admin	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Figure 9. Project Matrix Settings

- Once saved, test this new configuration by:
  - Accessing as non-authenticated user: not allowed to do anything.
  - Logging in as butler still lets you have rights
  - Logging in as colleague, contributor and checking rights:

Figure 10. Expected dashboard for Contributor

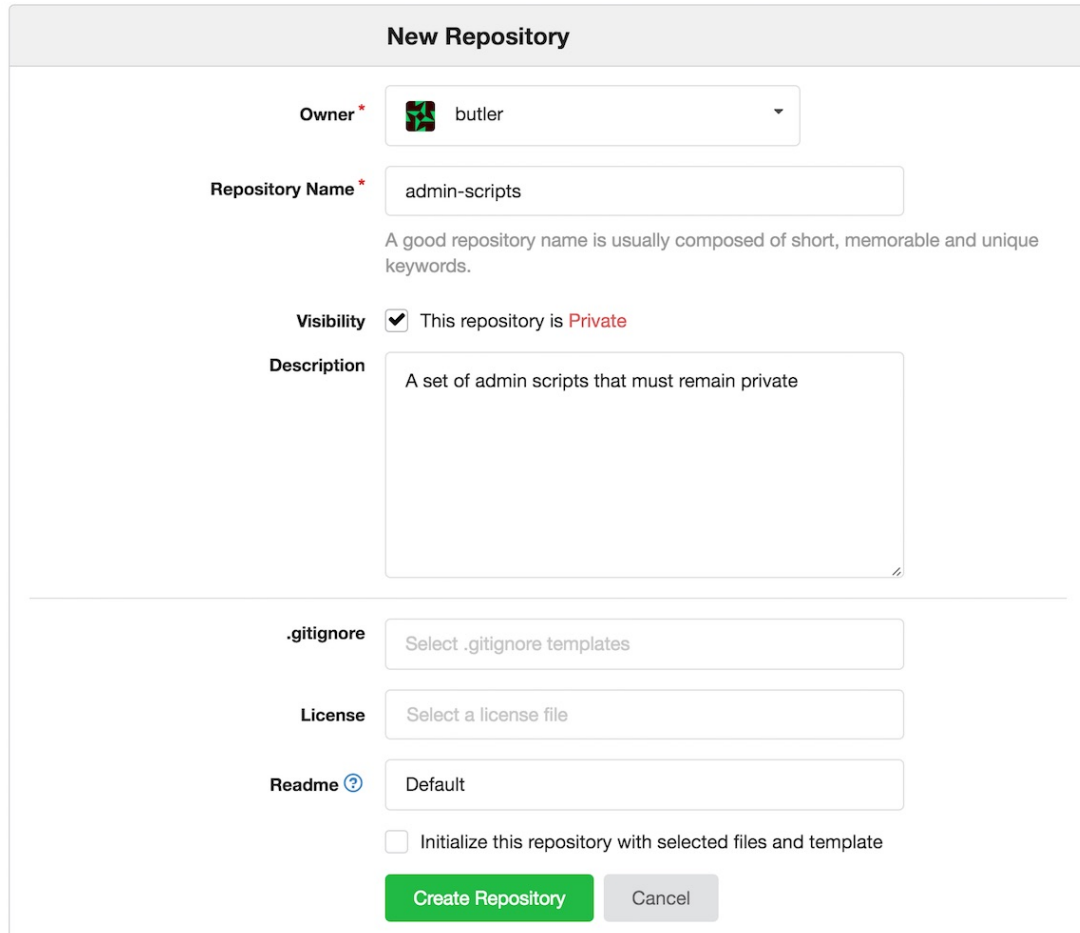
Figure 11. Expected dashboard for Colleague

## Accounting: Privacy and Credentials


The goal of this exercise is to create an administrator **private** job. It will be hidden for **ALL** users excepts administrators. Credentials are also used to access the related **private** repository.

- Start by logging into the [GitServer](#) as butler
- From the butler "homepage", create a new repository with these settings:
  - **Owner:** butler
  - **Repository Name:** admin-scripts

- **Visibility:** Select "This repository is Private"
- **Description:** A set of admin scripts that must remain private
- Use the **default** values for the other options:



**New Repository**

**Owner \***  butler ▼

**Repository Name \***

A good repository name is usually composed of short, memorable and unique keywords.

**Visibility** ☒ This repository is **Private**

**Description**

---

**.gitignore**

**License**

**Readme ?**

☐ Initialize this repository with selected files and template

Figure 12. New Private Repository in Gogs

- Once created, verify that it is **empty** and **private**, and copy the HTTP URL:

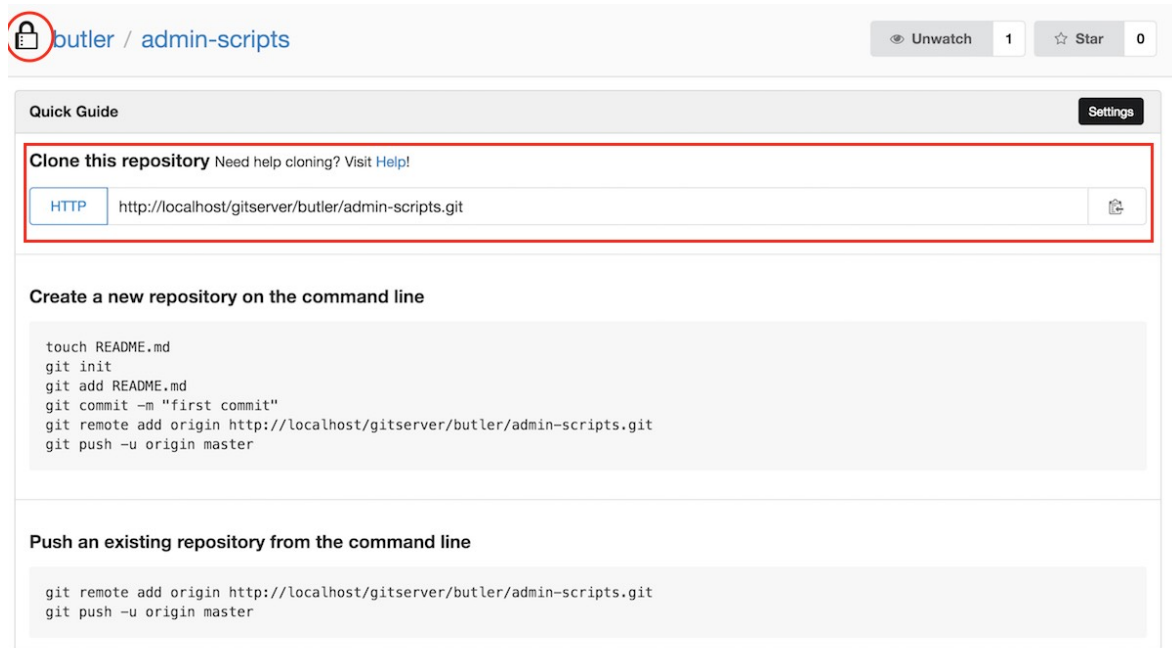


Figure 13. A private and empty Repository in Gogs

- Open the [WebIDE](#) in a new tab, and create a new project based on this repository:
  - **Project Name:** admin-scripts
  - **Folder Name:** admin-scripts
  - **...from Git Repo** ◦ **Git Repository:** <http://localhost:5000/gitserver/butler/admin-scripts>
  - It will be cloned **empty**
- In the [WebIDE](#), create a new file named `run-admin-task.sh` at the root, with the following content:

```
#!/bin/bash

set -e
set -u

echo "Some secret admin task to run there"
```

- Add it to git tracking, commit and push (providing credentials)

**TIP** Use butler, for the Git authentication when pushing.

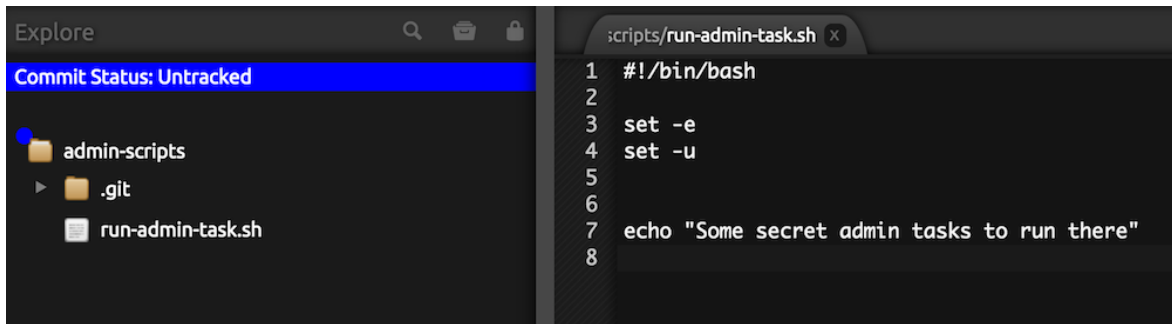


Figure 14. New script in private repository

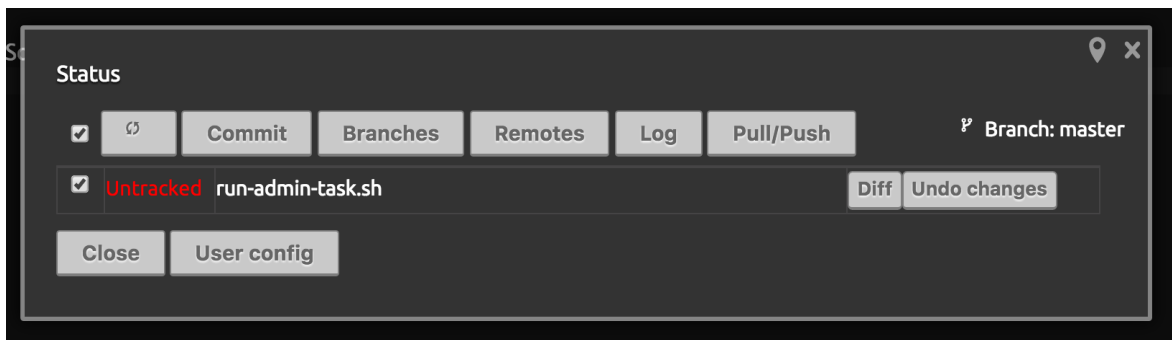


Figure 15. Git-tracking the new script

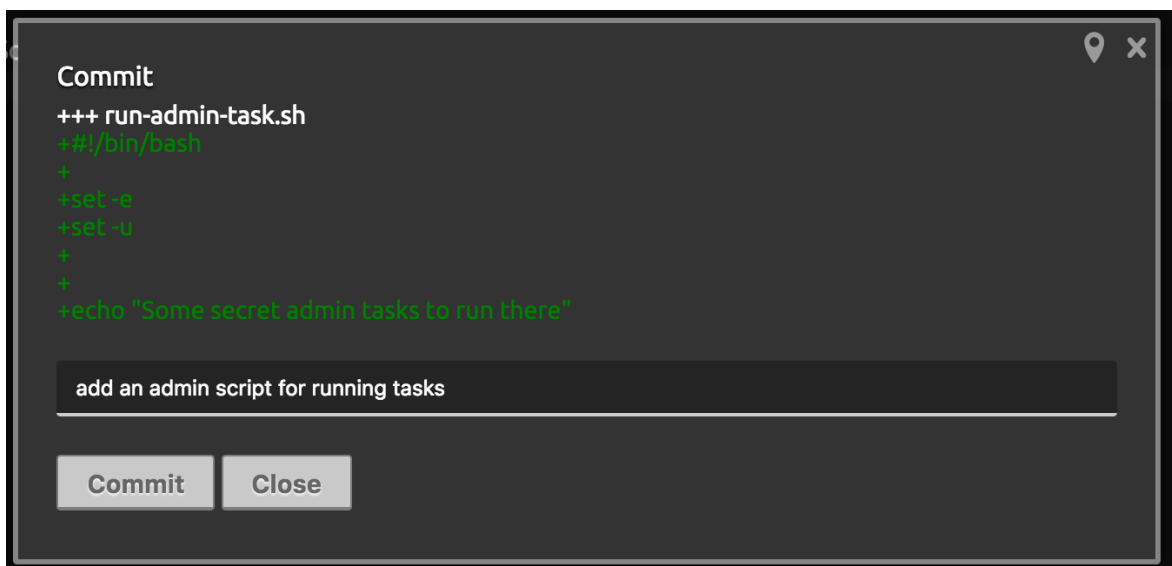


Figure 16. Git-committing the new script

- Go back to Jenkins, log in as butler
- Create a new **Freestyle** job named `admin-tasks` and configure it as following:
  - Select the checkbox **Enable project-based security**
    - Select the **Block inheritance of global authorization matrix** checkbox
    - Ensure that **only** the `jenkins-admin` group has all rights:

☒ Enable project-based security

☒ Block inheritance of global authorization matrix


User/group	Credentials				Job								Run			SCM		
	Create	Delete	Manage	Domains	Update	View	Build	Cancel	Configure	Delete	Discover	Move	Read	Workspace	Delete	Replay	Update	Tag
Anonymous	<input type="checkbox"/>	<input type="checkbox"/>		<input type="checkbox"/>						<input type="checkbox"/>		<input type="checkbox"/>		<input type="checkbox"/>		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
 jenkins-admin	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

Figure 17. Correct rights for admin project

- **Build:**

- Execute a shell script with this content:

```
bash -x run-admin-task.sh
```

- **SCM:** Git repository, from the previously created **private** repository

- Notice the **Red** error message that should appear: username and password are required to access this repository

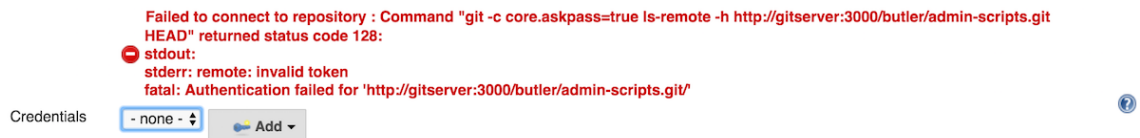


Figure 18. No access to this repository without authentication

- To add Credentials, click the **Add** button

- Select **Jenkins** in the drop-down
- Generate the Credential with these properties:
  - **Domain:** Global credentials (unrestricted)
  - **Kind:** Username with password
  - **Username:** butler
  - **Password:** butler
  - **ID:** butler-gogs-creds
  - **Description:** Butler's credentials for Gogs

Figure 19. Adding a Credentials for the private Repository

- Make the **Red** message go away by selecting the **newly** created **Credential**
- Save the job and build it
  - It should end in success, thanks to this credential
- Finally, log-in as **colleague** in Jenkins:
  - The job should not be visible

That's all for this exercise !

## Distribute Jenkins Builds

The goal of this exercise is to improve the scalability of our Jenkins instance by distributing our builds.

The target is to have:

- 2 agents that handle builds
  - An SSH agent used for Maven builds
  - A JNLP agent used for Docker-related tasks
- Each agent will have 1 executor
  - We have 2 CPUs available; rule of thumb says ~2 executors
- The master will not have any executors to run build anymore
- Security must follow up

## Adding an SSH agent

The Lab instance has an SSH "node" already running with these properties:

- Hostname: **ssh-agent**
- Port: **5022**
- Username: **jenkins**
- User authentication: RSA private Key
  - Key Passphrase: **No Passphrase**
  - Private Key location: In **Devbox**, in the file **/ssh-keys/vagrant\_insecure\_key**

**TIP** | A Passphrase is different than a password !

## Validating SSH

We are going to "validate" the SSH connection first:

- Spawn a shell in the **Devbox**
- Use ssh to connect to the SSH node using previous settings:




```
cloudbees-devbox $ ssh -i /ssh-keys/vagrant_insecure_key -p5022 \
jenkins@ssh-agent \
echo "-- Hello from ssh-agent"
...
Are you sure you want to continue connecting (yes/no)? yes
...
-- Hello from ssh-agent
```

- Copy the content of `/ssh-keys/vagrant_insecure_key` somewhere; we will use it in Jenkins

## Adding SSH Agent in Jenkins

- Log in to Jenkins as an administrator account
- Browse to the **Node Management Page**:
  - Use **Manage Jenkins**, and then **Manage Node** links
  - or use direct URL: <http://localhost:5000/jenkins/computer/>

**TIP** You see that your master has already declared there a Node by default

- Create a **New Node** using the left-menu:  **New Node**
  - Name: `ssh-agent`
  - Type: **Permanent Agent**
- Configure it with these properties:
  - # of executors: **1**
  - Remote root directory: `/home/jenkins`
  - No Labels
  - Usage: **Utilize as much as possible**
  - Launch method: **Launch slave agents via SSH** and configure it with previous properties
    - Host is `ssh-agent`
    - Command Line: click the button **Advanced** to access the **Port** field, and set it to `5022`
  - **Host Key Verification Strategy**: **Manually trusted key Verification Strategy**

**TIP** In case of error and retry, use the **Trust SSH Host Key** button if it has changed

The screenshot shows the Jenkins configuration page for an SSH Agent. The fields are as follows:

- Name:** ssh-agent
- Description:** (empty)
- # of executors:** 1
- Remote root directory:** /home/jenkins
- Labels:** (empty)
- Usage:** Use this node as much as possible
- Launch method:** Launch slave agents via SSH
- Host:** ssh-agent
- Credentials:** jenkins (SSH Agent Key) [Add button]
- Host Key Verification Strategy:** Manually trusted key Verification Strategy
- Require manual verification of initial connection:** ☐
- Port:** 5022

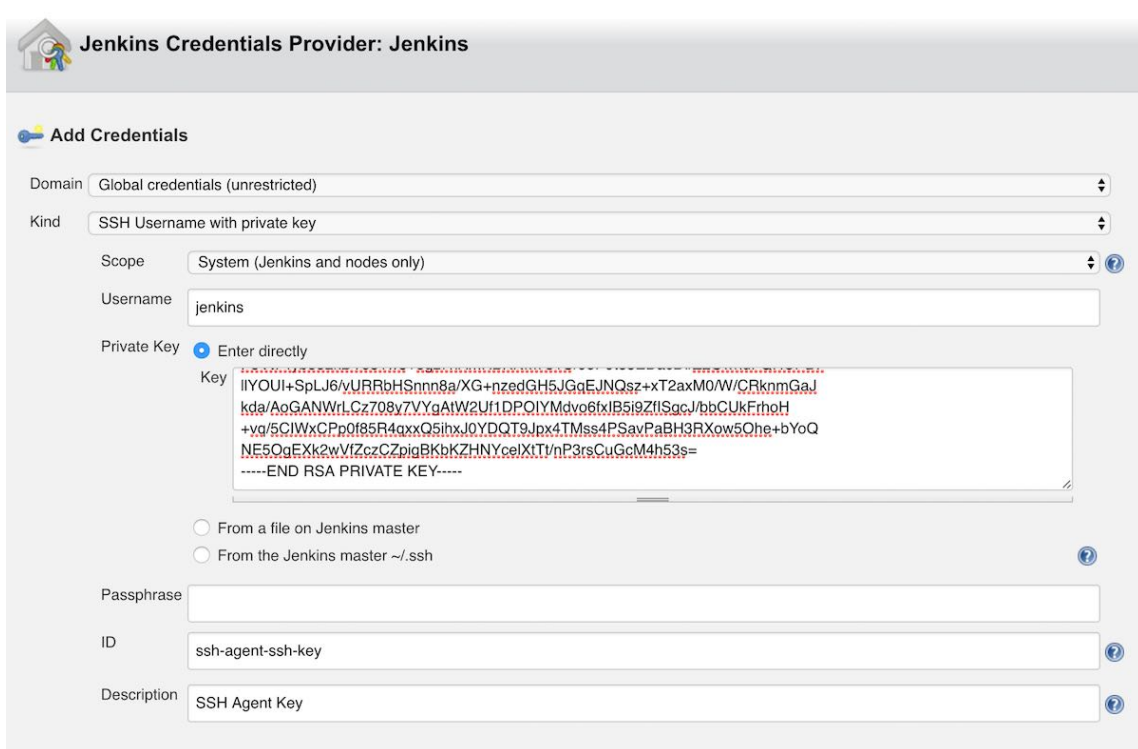
Figure 20. SSH Agent Configuration

- Don't forget to create a new Credential with the following properties:
  - Kind: **SSH Username with private key**
  - Scope: **System** (only admin can use it)
  - Username: **jenkins**
  - Private Key: **Enter directly** and copy/paste the SSH key used previously
  - Passphrase: leave it empty
  - ID: **ssh-agent-ssh-key**
  - Description: **SSH Agent Key**

**TIP**

Jenkins Credentials have a **SSH Username with private key** type. It supports:

- Typing directly private key in a text field
- Or picking keys from the Jenkins master file system
- Passphrase can also be stored encrypted



**Jenkins Credentials Provider: Jenkins**

**Add Credentials**

Domain: Global credentials (unrestricted)

Kind: SSH Username with private key

Scope: System (Jenkins and nodes only)

Username: jenkins

Private Key: ☒ Enter directly

Key: `IIYOUH+SpLJ6/yURRbHSnnn8a/XG+nzedGH5JGqEJNQsz+xt2axM0/W/CRknmGaJ  
kda/AoGANWrlCz708v7VYgAtW2Uf1DPOIYMdvo6fxIB5i9ZfISgcJ/bbCUkErhoH  
+vg/5CIWxCfp0f85R4qxxQ5ihxJ0YDQT9Jpx4TMss4PSavPaBH3RXow5Ohe+bYoQ  
NE5QqEXk2wVfZczCZpigBKbKZHNYcelXITV/nP3rsCuGcM4h53s=  
-----END RSA PRIVATE KEY-----`

☐ From a file on Jenkins master

☐ From the Jenkins master ~/.ssh

Passphrase:

ID: ssh-agent-ssh-key

Description: SSH Agent Key

Figure 21. SSH Agent Credentials Example

- You can see that your Node now appears in the Node list.
  - However it may have a little **red cross**: the agent is not yet started on this Node or is experiencing troubles
- Browse to the Log Console of the Node to see what is happening:
  - Command Lineck on the Node ssh-agent on the list
  - On the left-menu, Command Lineck on the **\*Log**
  - Notice** that the Agent is now online, by reading the log

**TIP** Agent took a few seconds to start. This is why you may have experienced the red cross

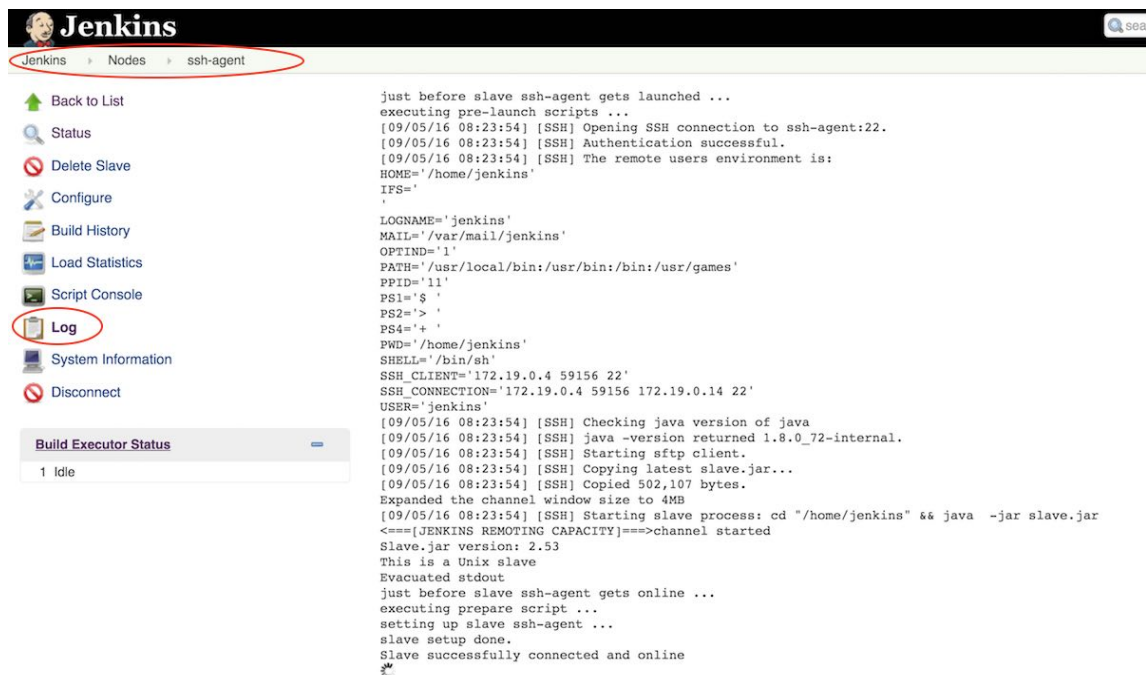


Figure 22. SSH Agent Logs

- Browse back to the Node list
  - The Agent is fully online:

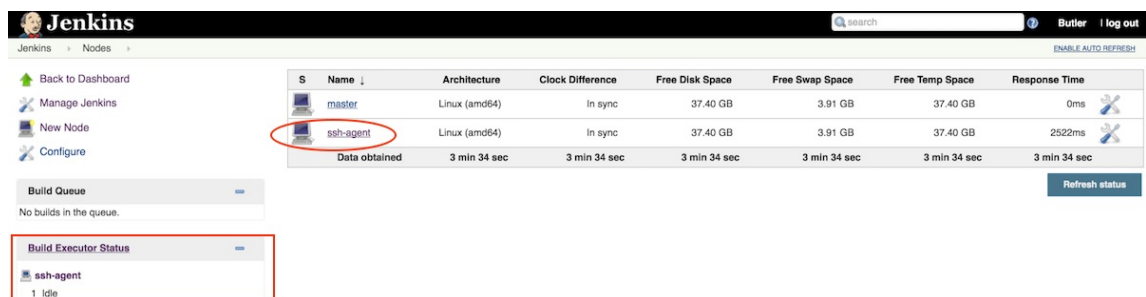


Figure 23. Jenkins SSH Agent Online

## Adding JNLP Agent in Jenkins

Target is to now add a JNLP agent. A dedicated **Node** run with Docker is already started.

First, we will experiment to understand how it works and then we will launch the JNLP agent in background

- Reusing what you did in the previous exercise, create a new JNLP agent:
  - Name: `jnlp-agent`
  - Kind: **Permanent Agent**
  - # of executors: **1**
  - Remote root directory: `/home/jenkins`
  - Labels: `docker`

- Usage: **Use this node as much as possible**
- Launch method: **Launch slave agent via Java Web Start**
- Add the DOCKER\_HOST Environment Variable, in the **Global Properties** section

Figure 24. JNLP Agent Configuration

- Once created, browse to this agent "status page":
  - Click on its name in the **Node List**
  - Or use this direct URL: <http://localhost:5000/jenkins/computer/jnlp-agent/>
- This agent is not running:
  - The current page explains how to launch it.
    - **Do not use this:** Using GUI Java Web Start by clicking the **Launch** button.
    - Showing an **example** command to launch it **headlessly** on a remote Node.
  - Copy the **secret** value somewhere; we will reuse it later

Figure 25. JNLP Agent Get Command

Now, it is time to launch the jnlp instance:

- Spawn a Command Line in the [Devbox](#)
- Using Docker, we're going to run a **bash** shell in the **jnlp-agent** container.
- A shell script is there **for you** to automate the JNLP startup:
  - **Read it !**
  - It expects the "secret" as only parameter
  - The Jenkins URL is already written inside; it can be overwritten by exporting the **\$JENKINS\_URL** variable before launching the script.

**TIP**

Starting a JNLP agent is a 2 step process. The shell script executes these steps:

1. Download the **slave.jar** from the Jenkins master
2. Execute a java command, passing parameters:
  - The jar file with **-jnlpUrl** switch
  - The Jenkins secret with **-secret** switch

```
cloudbees-devbox $ docker exec -ti jnlp-agent /bin/bash
bash-4.3$ bash /usr/local/bin/start-jnlp.sh
ERROR: Please provide a secret as parameter
bash-4.3$ bash /usr/local/bin/start-jnlp.sh INSERT_SECRET_HERE
...
INFO: Setting up slave: jnlp-agent
...
INFO: Connected
```

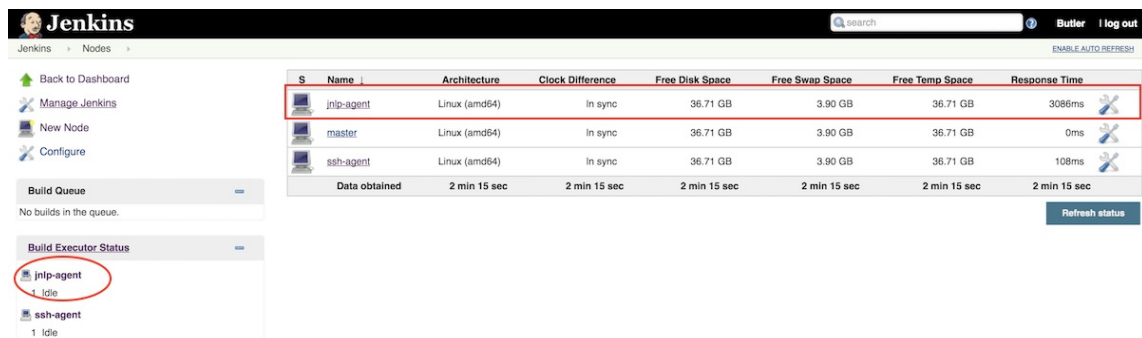
- Without killing this Command Line, browse back to the Jenkins agent Node page
  - The JNLP agent is online !
  - But if we stop the command line, the agent will go down: type the keyboard shortcut **CTRL+C** to stop the foreground script
  - Check the JNLP agent status in Jenkins: it is offline.
- Launch this script in background for having the agent running permanently:
  - Go back to your [Devbox](#) Command Line
  - Launch the script again, with the same parameters, but prepend the command **nohup** and append the **&** special shell character to run the process in background **and** detached (see below), and exit the container with **exit**:

**TIP**

You have to use the **Enter** key 2 times to gain control of the Command Line again.

```
bash-4.3$ nohup bash /usr/local/bin/start-jnlp.sh INSERT_SECRET_HERE &
[1] 165
bash-4.3$ nohup: appending output to nohup.out
bash-4.3$
bash-4.3$ ps faax # Verifying that the process is running
PID   USER     TIME   COMMAND
    1  jenkins   0:00   sh -c while true; do sleep 5;done
    7  jenkins   0:00   /bin/bash
   53  jenkins   0:00   bash /usr/local/bin/start-jnlp.sh YOUR_SECRET
   55  jenkins   0:02   java -jar /home/jenkins/slave.jar -jnlpUrl
http://jenkins:8080/jenkins/computer/jnlp-agent/slave-agent.jnlp -secret
3ac5eb9261b5d7cb46450e6d4cc957f730419dd3ecbb0f82b12391ebd66cbac0
   77  jenkins   0:00   sleep 5
   78  jenkins   0:00   ps faux
bash-4.3$ exit
exit
```

- Go back (again) to the Jenkins agent Node page
  - The JNLP agent is **still** online:



The screenshot shows the Jenkins 'Nodes' page. On the left, there's a sidebar with links like 'Back to Dashboard', 'Manage Jenkins', 'New Node', and 'Configure'. Below these are sections for 'Build Queue' (showing no builds) and 'Build Executor Status'. In the 'Build Executor Status' section, the 'jnlp-agent' is highlighted with a red circle, showing it is '1 idle'. The main table lists three nodes: 'jnlp-agent', 'master', and 'ssh-agent'. The 'jnlp-agent' row is highlighted with a red border, indicating it is online. The table columns include Name, Architecture, Clock Difference, Free Disk Space, Free Swap Space, Free Temp Space, and Response Time.

S	Name	Architecture	Clock Difference	Free Disk Space	Free Swap Space	Free Temp Space	Response Time
	jnlp-agent	Linux (amd64)	In sync	36.71 GB	3.90 GB	36.71 GB	3086ms
	master	Linux (amd64)	In sync	36.71 GB	3.90 GB	36.71 GB	0ms
	ssh-agent	Linux (amd64)	In sync	36.71 GB	3.90 GB	36.71 GB	108ms
Data obtained		2 min 15 sec	2 min 15 sec	2 min 15 sec	2 min 15 sec	2 min 15 sec	2 min 15 sec

Figure 26. JNLP Agent online

## Reconfiguring Master

It is now time to reconfigure the Jenkins master, to stop executing builds and enforce security to the Nodes:

- Browse the Jenkins **Configure System** page
  - Set the **# of executors** to 0

### TIP

It is a good practice to avoid having executors on the Jenkins master. Any job running on the master can access the JENKINS\_HOME where it could leak data (Credentials, etc.) or delete things accidentally.

- Add a **label** master
- Set the **Usage** to "Only build jobs with label restrictions matching this node"
- Remove the DOCKER\_HOST Environment variable in **Global properties**
- Don't forget to save the configuration !

### TIP

We are going to use only the JNLP agent for Docker builds

# of executors	0
Labels	master
Usage	Only build jobs with label restrictions matching this node
Quiet period	5
SCM checkout retry count	0
<input type="checkbox"/> Restrict project naming	
<b>Global properties</b>	
<input checked="" type="checkbox"/> Environment variables	
<input type="checkbox"/> Tool Locations	

Figure 27. Master Global Configuration

- Now, browse to the **Configure Global Security** page
  - Enable the **Enable Slave → Master Access Control** by checking the appropriate checkbox (scroll down)
  - Save the configuration
- Go back to the Jenkins dashboard,
  - You should now see **only** 2 executors available: one per agent.

**Build Queue**

No builds in the queue.

**Build Executor Status**

**jnlp-agent**  
1 Idle

**ssh-agent**  
1 Idle

Figure 28. Jenkins Executors Count

## Reconfiguring Jobs

Next step is to create a dedicated Job for the "Docker build" step.

**TIP** This job is triggered after a successful Maven build, but it only runs in the jnlp agent.

- Create a new **Freestyle** job:
  - Name: `demoapp-docker-builder`
  - Add a parameter:
    - Type **String Parameter**
    - Name: `APP_VERSION`



- No **Build Trigger**
- NO **SCM**
- Only a **Build step**:
  - Type **Execute Shell**
  - Content:

```
# Build Application
docker build -t "demo-app:${APP_VERSION}" ./
```

#### IMPORTANT

We now need to pass some archived artifacts from the Maven build to this new Docker Builder:

- Dockerfile
- The application: `demoapp.jar` file
- The configuration: `hello-world.yml` file

We will use the [Copy Artifact Plugin](#)

- Enable the authorization of "Copying artifacts" from job `demoapp-build`

The screenshot shows the Jenkins configuration page for a Docker Builder. The 'General' tab is selected. The 'Project name' is 'demoapp-docker-builder'. Under the 'Permission to Copy Artifact' section, 'demoapp-build' is listed as a project to allow copying artifacts. A 'String Parameter' named 'APP\_VERSION' has been added with a default value field. Other options like 'Enable project-based security', 'Discard old builds', 'GitHub project', 'Use Gogs secret', 'Promote builds when...', 'Throttle builds', 'Disable this project', 'Execute concurrent builds if necessary', and 'Restrict where this project can be run' are all unchecked. The 'Add Parameter' button is visible at the bottom of the parameter section.

Figure 29. Docker Builder Global Configuration

- Add a Build Step, **before** the shell one, that will **Copy Artifacts from another project**:
  - Project name: `demoapp-build`

- Which build: **Upstream build that triggered this job**, use the **latest** as fallback
- Artifacts to copy: `target/demoapp.jar`, `Dockerfile`, `hello-world.yml`

Figure 30. Docker Builder Build Configuration

- Also, add a **Post-Build** action, that triggers the `demoapp-staging-deployer` job by passing it the Docker Image name as a parameter
- Double check the parameters name:

```
DOCKER_IMAGE=demo-app:${APP_VERSION}
```

- **Save** this configuration when ready and reviewed

Figure 31. Docker Builder Post-Build Configuration

- Go back to the `demoapp-build` configuration
- Now, remove the **Execute shell** build step that handled the docker commands, letting only the **Invoke top-level Maven**

**targets** step alone:

Figure 32. Maven Builder New Build Steps

- Update the Archiving section, in **Post Build Actions**, by appending these files: `Dockerfile`, `hello-world.yml`
- Update the **Trigger Parameterized build on other project** section, in **Post Build Actions** with the following:
  - Change **Project to build** to the new `demoapp-docker-builder` job
  - The **Predefined parameter** value is now: `APP_VERSION=${GIT_COMMIT}`

Figure 33. Maven Builder New Post Build Actions

- You can **Save** the configuration

## Running Distributed Builds

It is time to run our builds. Still, some questions need to be answered:

- How to ensure that the **Docker** related job sticks to the JNLP agent ?
  - This is the only place where the `DOCKER_HOST` is set. Running docker commands from the ssh-agent will end with the error `Cannot connect to the Docker daemon. Is the docker daemon running on this host?`

The goal is to specialize the jnlp-agent by using the label `docker` from the **Docker** related Jobs.

- Edit the 2 Docker related jobs, and for each:
  - Select the checkbox **Restrict where this project can be run**
  - Provide the `docker` value to the **Label Expression** field
  - Save the configuration

#### IMPORTANT

Both agents are "fungible" outside the docker specialization: the demoapp-build with Maven can occur on any of those agents.

Since we use the Jenkins **Tools** configuration, Jenkins ensures that Maven is installed before any builds on an agent. Check the next demoapp-build output log for this.

It is now time to kick a build on demoapp-build and see how everything works !

#### TIP

If your build fails at the SCM stage, remember to add the butler credential now

Some things to notice:

- Each Build Output now starts with a line telling which agent was used for: **Building remotely on \$AGENT\_NAME (\$LABEL) in workspace \$WORKSPACE\_ON\_AGENT**
- Browse the Node list on the **Manage Node** page:
  - You can see which project are tied to a **specific Agent** (in our case: None)
  - If you Command Lineck on a label like `docker`, you can see which projects are tied to this **specific label**. (In our case, it is the 2 Docker related projects)
- The Node Management system has a lot of admin utilities.
  - Browse to a Node page, using **Manage Node** > Command Lineck a Node >
  - Check each Node's **System Information**: For jnlp-agent you have `$DOCKER_BUCKET` that comes from the template used.
  - Check the **Load statistics** graph to view the node activity after a few builds

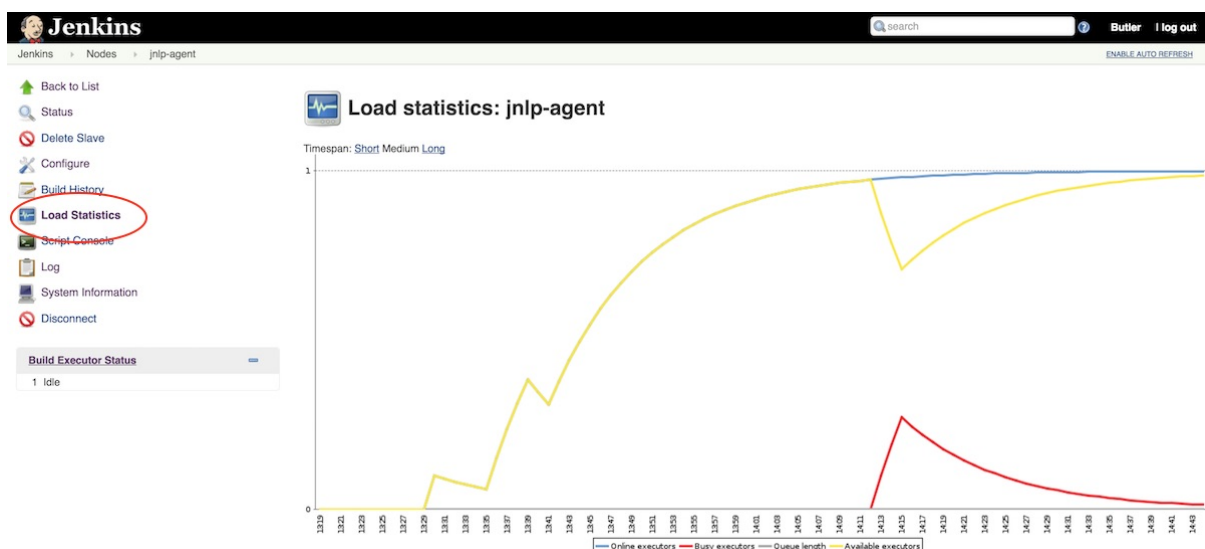


Figure 34. Example of Load Statistics

That's all for this exercise!

## Job Promotion

The goal of this exercise is to add some "tags" to our build, depending on its status.

We want to run the Docker build as soon as the build has generated artifacts, without waiting for the Integration Tests to run.

The tags will allow us to quickly identify which step(s) ran with success, and which did not.

**TIP** It is a kind of "parallelization of build steps of our pipeline"

- Start by editing the demoapp-build configuration with the following:
  - The build step **Invoke top-level Maven targets** will only run the `package` goal

**TIP** The Maven `package` goal **only** builds the application; the package goal is composed of "compile", "unit-tests", and "package". It "makes" the application but does not run integration tests.

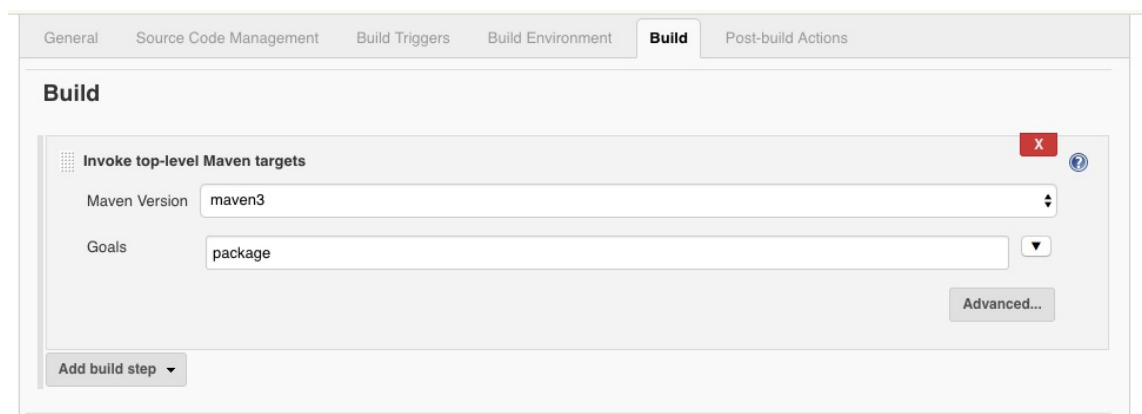


Figure 35. New Maven Job Configuration

- Enable the **Promote builds when...** in the **General Section**
- Add a new **Promotion process**. The goal of this process is to add an "Orange Star" tag to the build as soon as the build steps succeeds, and then to launch the Integrations Tests:
  - **Name:** Integration Tests
  - **Icon:** select Orange Star
  - **Criteria:** Promote immediately once the build is complete
  - Add a new **Action** with the following settings:
    - **Type:** Invoke top-level Maven targets
    - **Goal:** verify
  - Enable the **Restrict where this promotion process can be run** and set it to `ssh-agent`
  - Enable the **Restrict where this project can be run** option, in the **General** section, and set the field to `ssh-agent`:

This "Restricting" setting is used to prevent the integration test from running the promotion process on a different agent than the one where the build steps ran; the promotion process might have another version of the code since the SCM updates did not occur on all agents.

**TIP**

[Promotion Plugin's documentation](#) warns us that we never should rely on the "Workspace" for promotions.

For simplicity, this exercise restricts the build AND the promotion to the same node.

A good implementation would be to use the **Copy Artifact** to follow documented good practices.

Figure 36. Orange Promotion Configuration

- Add another **Promotion process**. The goal of this 2nd process is to add a "Green Star" tag to the build as soon as `demoapp-docker-builder` has completed successfully:
  - **Name:** Docker Build
  - **Icon:** select Green Star
  - **Criteria:** When the following downstream projects build successfully, with the **Job name** field set to `demoapp-docker-builder`

The screenshot shows the 'Promotion process' configuration for a job named 'Docker Build'. The 'General' tab is selected. The 'Name' field is 'Docker Build'. The 'Visible' field is empty. The 'Icon' is set to 'Green star'. There is a checkbox 'Restrict where this promotion process can be run' which is unchecked. Under the 'Criteria' section, the checkbox 'When the following downstream projects build successfully' is checked. Below this, the 'Job names' field is set to 'demoapp-docker-builder'. There is also an unchecked checkbox 'Trigger even if the build is unstable'. Other criteria like 'Custom Groovy script', 'Only when manually approved', 'Promote immediately once the build is complete', and 'Promote immediately once the build is complete based on build parameters' are unchecked. Under the 'Actions' section, there is an 'Add action' button. At the bottom right, there is a button 'Add another promotion process'.

Figure 37. Green Promotion Configuration

- Finally, we want the job admin-tasks launched as soon as the **Green Star** has been applied to a build:
  - Go to the admin-tasks job's configuration
  - Set a **Build Trigger** to `Build when another project is promoted`, with the **Job Name** field set to demoapp-build:

The screenshot shows the 'Build Triggers' configuration for a job named 'demoapp-build'. The 'Build Triggers' tab is selected. The 'Trigger builds remotely (e.g., from scripts)' checkbox is unchecked. The 'Build after other projects are built' checkbox is unchecked. The 'Build periodically' checkbox is unchecked. The 'Build when another project is promoted' checkbox is checked. Below this, the 'Job Name' field is set to 'demoapp-build'. The 'Promotion' field is set to 'Docker Build'. At the bottom, there are two unchecked checkboxes: 'GitHub hook trigger for GITScm polling' and 'Poll SCM'.

Figure 38. Triggering a build based on promotion

- Make the Integration tests fail:
  - Open a new tab with the [WebIDE](#)
  - Edit the file `HelloWorldIntegrationTest.java`, which is located in the `src/test/java/com/cloudbees/demoapp/test/` directory

- Edit the line `assertThat(result.getContent()).isEqualTo("Hello, dropwizard!");`
- Replace by `assertThat(result.getContent()).isEqualTo("Hello, Butler!");`
- Commit and Push
- A new build should start, resulting in:
  - A Green Star applied (Docker build pass), staging deployed and admin-tasks built.
  - No Orange Star but a *forbidden sign*:




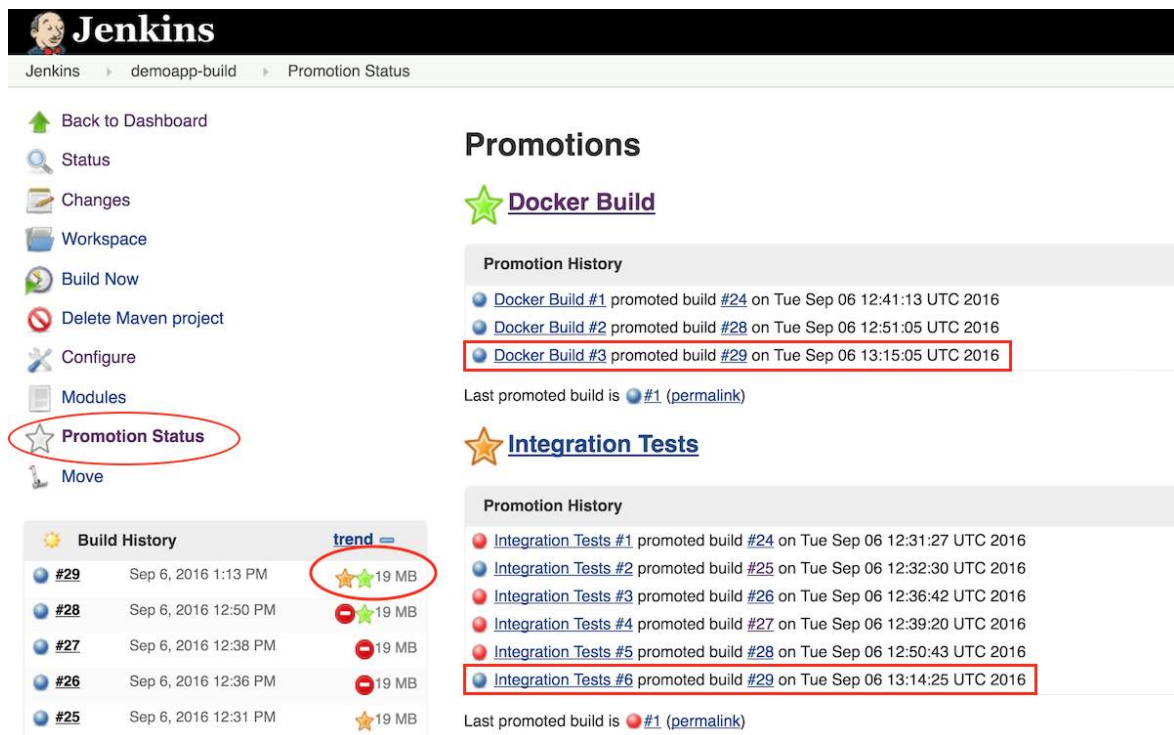
Build History <span>trend</span>		
 <b>#28</b>	Sep 6, 2016 12:50 PM	  19 MB

Figure 39. Only Green Promotion

- Now roll back your change, commit & push:
  - Both Green and Orange Promotion will be done
- Browse to the **Promotion Status** link (left-menu):

**TIP**

Note that, from the "Promotion Status" page, you can access the Promotions history, output logs, and even re-launch promotions to avoid rebuilding



**Jenkins**

Jenkins > demoapp-build > Promotion Status

Back to Dashboard

Status

Changes

Workspace

Build Now

Delete Maven project













Configure

Modules


**Promotion Status**

Move




**Build History** trend


 <b>#29</b>	Sep 6, 2016 1:13 PM	  19 MB
 <b>#28</b>	Sep 6, 2016 12:50 PM	  19 MB
 <b>#27</b>	Sep 6, 2016 12:38 PM	 19 MB
 <b>#26</b>	Sep 6, 2016 12:36 PM	 19 MB
 <b>#25</b>	Sep 6, 2016 12:31 PM	 19 MB


**Promotions**

 **Docker Build**







**Promotion History**

-  [Docker Build #1](#) promoted build [#24](#) on Tue Sep 06 12:41:13 UTC 2016
-  [Docker Build #2](#) promoted build [#28](#) on Tue Sep 06 12:51:05 UTC 2016
-  [Docker Build #3](#) promoted build [#29](#) on Tue Sep 06 13:15:05 UTC 2016

Last promoted build is  [#1](#) ([permalink](#))

 **Integration Tests**

**Promotion History**

-  [Integration Tests #1](#) promoted build [#24](#) on Tue Sep 06 12:31:27 UTC 2016
-  [Integration Tests #2](#) promoted build [#25](#) on Tue Sep 06 12:32:30 UTC 2016
-  [Integration Tests #3](#) promoted build [#26](#) on Tue Sep 06 12:36:42 UTC 2016
-  [Integration Tests #4](#) promoted build [#27](#) on Tue Sep 06 12:39:20 UTC 2016
-  [Integration Tests #5](#) promoted build [#28](#) on Tue Sep 06 12:50:43 UTC 2016
-  [Integration Tests #6](#) promoted build [#29](#) on Tue Sep 06 13:14:25 UTC 2016


Last promoted build is  [#1](#) ([permalink](#))

Figure 40. Promotion Status Details

That's all for this exercise!



## Organize your jobs with Folders and Views

The goal of this exercise is to use Folders on the Jenkins instance.

Your group of developers will be split in 2 teams: **TeamA** and **TeamB**. They will work on the same application, but adapt it for different customers.

### IMPORTANT

Sadly, the staging environment is shared. We suppose that project managers of TeamA and TeamB are well synchronized :)

## Creating Folder

- Start by creating a new folder:
  - Use the **New Item** button on the left-menu
  - **Name:** TeamA
  - Kind: **Folder**
- Take time to review the Folder settings:
  - Enable Project security to only allow access to members of the `jenkins-admin` and `jenkins-developers` groups:

Name: TeamA

Display Name:

Description:

Views Tab Bar: Default Views TabBar

Icon: Default Icon

Health metrics

☒ Enable project-based security

User/group	Credentials			Job								Run			View			Promote	SCM				
	Create	Delete	Manage	Domains	Update	View	Build	Cancel	Configure	Create	Delete	Discover	Move	Read	Workspace	Delete	Replay	Update	Configure	Create	Delete	Read	Tag
jenkins-admin	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
jenkins-developers	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Anonymous	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

User/group to add:

Figure 41. New Folder for Team A

- The newly created folder is empty:



All



## Welcome to Jenkins!

Please **create new jobs** to get started.

Figure 42. Team A Folder is empty

- Browse to main Dashboard (use the **Up** button in left-menu)
- For **both** demoapp-build and demoapp-docker-builder:
  - Go the the Job page
  - Use the **Move** button (on the left-menu)
  - Move to the **TeamA** folder

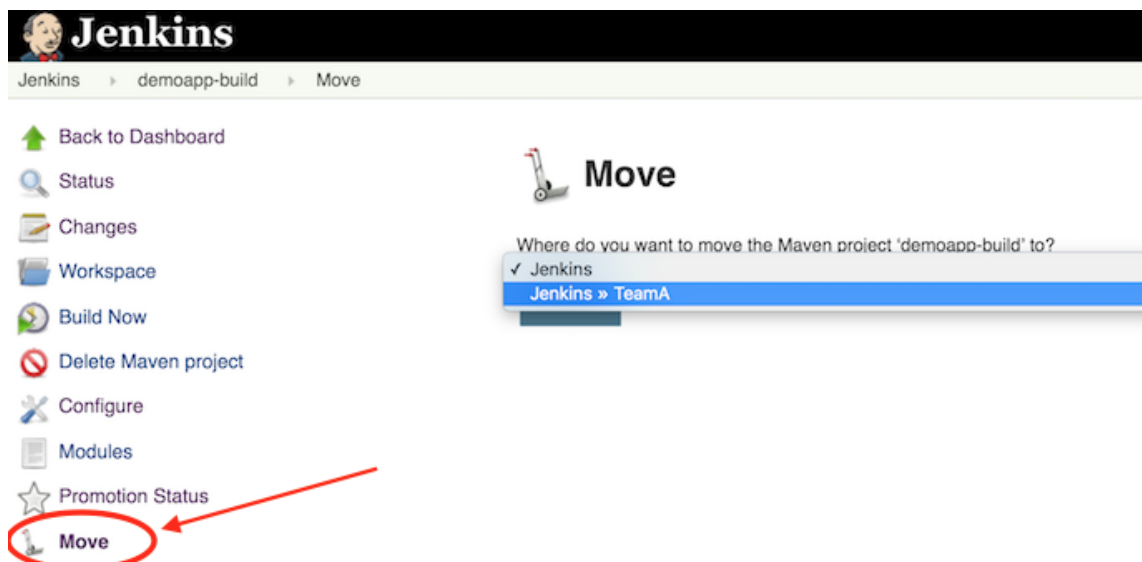


Figure 43. Move to the Team A folder

## Copy Folder

We now want to initialize the Team B folder, with the same settings as Team A.

- Create a **New Item**:

- **Name:** TeamB
- **Type:** **Copy existing Item**
- **Copy from:** TeamA
- Review the configuration: same as Team A !
- You now have a **TeamB** folder with a **copy** of the original content
  - Jobs have no **Build** button (yet), and their status is **Grey**

**IMPORTANT**

You should review the Job configurations **before** you try to launch a build from them.



All +		
S	W	Name ↓
		<a href="#">demoapp-build</a>
		<a href="#">demoapp-docker-builder</a>

Icon: [S](#) [M](#) [L](#)

Figure 44. TeamB folder with newly created jobs

- Review and **Save** the new job configurations and kick off builds of the demoapp-build for **both** folders
  - Observe that they behave as independent jobs

**TIP**

Even the upstream / downstream links have been adapted

**Jenkins**

Jenkins > TeamB > demoapp-docker-builder >

Up  
Status  
Changes  
Workspace  
Build with Parameters  
Delete Project  
Configure  
Move

**Build History** trend RSS for all RSS for failures

**Project demoapp-docker-builder**  
Full project name: TeamB/demoapp-docker-builder

Workspace  
Recent Changes

**Upstream Projects**  
TeamB » demoapp-build

**Downstream Projects**  
demoapp-staging-deployer

**Permalinks**

Figure 45. Upstream and Downstream Jobs adapted

## Folders on FileSystem

Folders provide **namespacing** for Jenkins objects. As you have seen, jobs have same names, but can still co-exist within their own folders.

- How is it behaving in the Jenkins home folder ?
- Open a Command Line on [Devbox](#)
- Spawn a **bash** shell on the Jenkins machine using `docker exec (...)`
- Browse to the `${JENKINS_HOME}/jobs` directory and check the file-system tree:

```
cloudbees-devbox $ docker exec -ti jenkins /bin/bash
bash-4.3$ cd /home/jenkins/jobs/
bash-4.3$ ls -l
# 1 Filesystem Dir for EACH TeamX object
TeamA
TeamB
admin-tasks
demoapp-staging-deployer
bash-4.3$ ls -l TeamA
# Each contains a config file and a sub `jobs` folder
config.xml
jobs
bash-4.3$ ls -l TeamA/jobs/
# Jobs are stored inside the sub-folder `jobs`
demoapp-build
demoapp-docker-builder
bash-4.3$ diff -u TeamA/config.xml TeamB/config.xml
# No differences at configuration level, or the one you may have introduced
```

That's all for this exercise!

## Track your artifacts and dependencies with fingerprinting

Now that we have a **shared** build ( demoapp-staging-deployer ), how can we sure, at a given time, which artifacts have been deployed in staging ?

We are going to use **fingerprints** to track our builds.

- Start by going to the configuration of TeamA's demoapp-build
  - Scroll down to the **Archive Artifact** post build action
  - Click the **Advanced** button
  - Enable the **Fingerprint all archived artifacts** checkbox

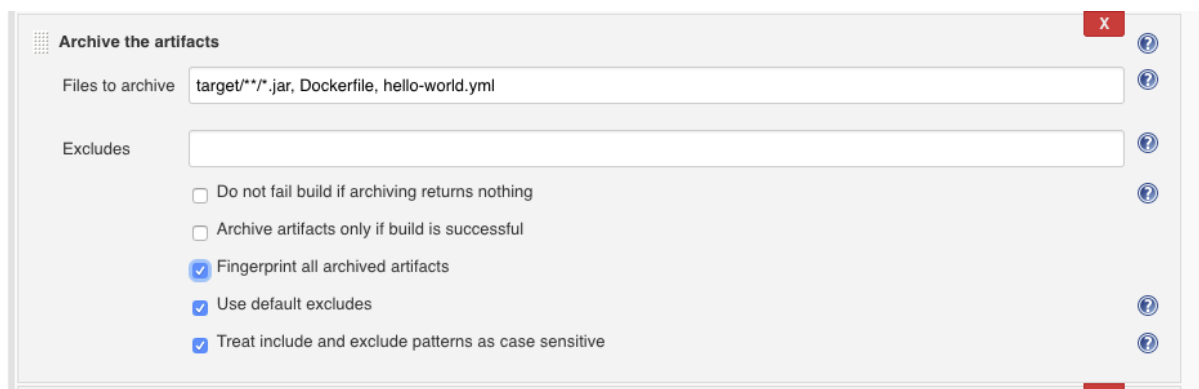


Figure 46. See latest success build fingerprints

- Save & Kick off a new build
- Once it is a success, browse to the latest build page,
  - Click the button **See Fingerprints** on the left menu

- You can see all archived artifacts and their tracking:
  - Where do they come from ? (**Original owner**)
  - Have they changed ?



## Recorded Fingerprints

File ↓	Original owner	Age	
demoapp.jar	this build	1 min 23 sec old	<a href="#">more details</a>
Dockerfile	<a href="#">TeamA/demoapp-build #3</a>	15 hr old	<a href="#">more details</a>
hello-world.yml	<a href="#">TeamA/demoapp-build #3</a>	15 hr old	<a href="#">more details</a>
target/demoapp.jar	this build	1 min 23 sec old	<a href="#">more details</a>
target/original-demoapp.jar	<a href="#">TeamA/demoapp-build/com.dduportal.jenkins.demoapp #32</a>	1 min 27 sec old	<a href="#">more details</a>

Figure 47. See latest success build fingerprints

- Try to check details of the `demoapp.jar` file
- You'll be able to view:
  - Builds using these artifacts
  - The **MD5** hashsum of each artifact



Figure 48. Details about an artifact

- Open a CLI in **Devbox**
- Spawn a **bash** shell on the Jenkins master with `docker exec`
- Browse to the `${JENKINS_HOME}/fingerprints` folder
- You'll see a specific tree that Jenkins maintains for traceability; this is the fingerprint **database**
- Try to find one of your artifacts:
  - Find the MD5sum from the Jenkins GUI. We'll use `3a66b07aa92b01c2267db94fe918ee05` in the snippets below
  - Use the `grep` command to find it:

```
cloudbees-devbox $ grep -rI 3a66b07aa92b01c2267db94fe918ee05 \
/home/jenkins/fingerprints/
/home/jenkins/fingerprints/3a/66/b07aa92b01c2267db94fe918ee05.xml :
<md5sum>3a66b07aa92b01c2267db94fe918ee05</md5sum>
```

- Check the content of the **XML** file found by `grep`:

b07aa92b01c2267db94fe918ee05.xml

```
<?xml version='1.0' encoding='UTF-8'?>
<fingerprint>
  <timestamp>XXXXXXXXXXXX UTC</timestamp>
  <original>
    <name>TeamA/demoapp-build</name>
    <number>32</number>
  </original>
  <md5sum>3a66b07aa92b01c2267db94fe918ee05</md5sum>
  <fileName>demoapp.jar</fileName>
  <usages>
    <entry>
      <string>TeamA/demoapp-build</string>
      <ranges>32</ranges>
    </entry>
    <entry>
      <string>TeamA/demoapp-docker-builder</string>
      <ranges>13</ranges>
    </entry>
  </usages>
  <facets/>
</fingerprint>
```

That's all for this exercise !

## Jenkins Command Line Interface

The goal of this exercise is to experiment with the Jenkins Command Line Interface (CLI) and its REST API.

### Requirements

JNLP port must be correctly configured to ensure that Jenkins CLI will work:

- Check this on the **Jenkins Global Security**:
  - **Enable security** must be enabled
  - The **TCP port for JNLP slave agents** must be **Fixed** to 50000

Our Jenkins instance is secured. Thus the **JENKINS CLI plugin** requires to use an SSH based authentication per user:

#### TIP

the flags `--username` and `--password` are still there for backward compatibility but won't work on our Jenkins >=1.419

- Browse to the User configuration page for butler:
  - Click on the top-right on the username
  - then click on the left-menu **Configure button**
  - OR use the direct URL: <http://localhost:5000/jenkins/user/butler/configure>

- In the field **SSH Public Keys**, add a new SSH *Public* key:
  - The public key is stored locally. Open the following link in a new tab of your web browser, and copy/paste its :name: value [http://localhost:5000/pub\\_key.txt](http://localhost:5000/pub_key.txt)
- **Save** the configuration

## Step 1. Download the client jar

- Start by browsing to the Jenkins Master, click **Manage Jenkins**, and then click **Jenkins CLI** to get to the Jenkins CLI web page.

**TIP** Alternatively, you can go directly to this page by navigating to <http://localhost:5000/jenkins/cli>

- You will find there a *quickstart* example, link to documentation and a comprehensive list of commands available
- It also gives you an URL to **jenkins-cli.jar**:

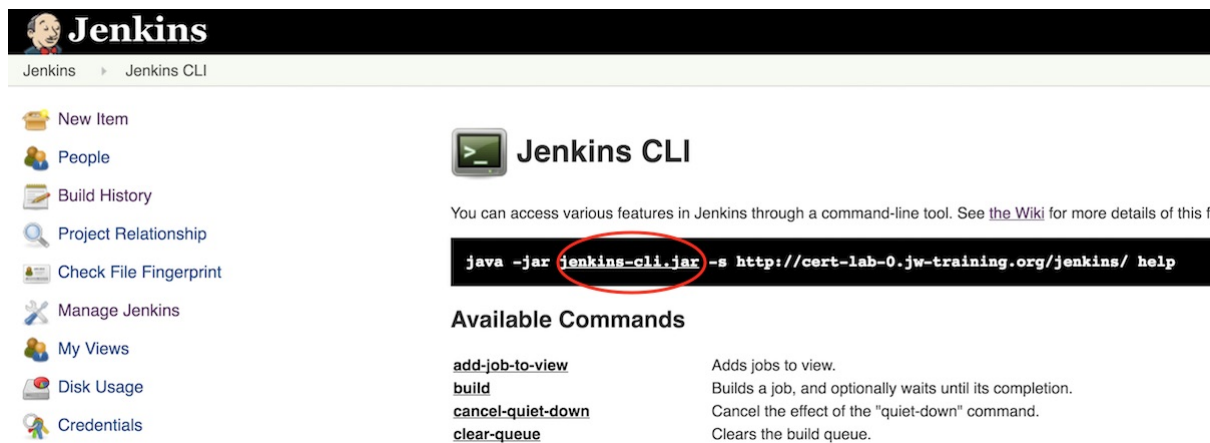


Figure 49. Link to download Jenkins CLI

- Open a **Devbox** command line, where you will download the `jenkins-cli.jar` file into `/tmp`:

### IMPORTANT

We will use Jenkins private URL, <http://jenkins:8080/jenkins> instead of the public one that Jenkins shows you. It allows to reach Jenkins directly instead of using the reverse-proxy.

```
cloudbees-devbox $ cd /tmp
cloudbees-devbox $ curl -LO \
  http://jenkins:8080/jenkins/jnlpJars/jenkins-cli.jar
...
cloudbees-devbox $ ls jenkins-cli.jar
jenkins-cli.jar
```

## Step 2. Run the help command

- Run the proposed command (`help`).
  - It should ask you for authentication against Jenkins master
  - Use the `-i` flag to specify the path to the **private** SSH key



```
cloudbees-devbox $ java -jar ./jenkins-cli.jar \
  -s http://jenkins:8080/jenkins help
You must authenticate to access this Jenkins.
Use --username/--password/--password-file parameters or login command.
...
cloudbees-devbox $ java -jar jenkins-cli.jar \
  -s http://jenkins:8080/jenkins \
  -i /ssh-keys/vagrant_insecure_key help
...
```

- Set Jenkins URL as the `${JENKINS_URL}` environment variable so that you don't have to keep retyping long URL.

It should produce the same output:

```
cloudbees-devbox $ export JENKINS_URL=http://jenkins:8080/jenkins
cloudbees-devbox $ java -jar jenkins-cli.jar \
  -i /ssh-keys/vagrant_insecure_key help
```

## Step 3. Run a build

- Now, let's launch a new "our CD pipeline" run by kicking off a new build of the job demoapp-build:

**TIP** The `-s` flag makes the command wait for the completion of the build.

```
cloudbees-devbox $ java -jar jenkins-cli.jar \
  -i /ssh-keys/vagrant_insecure_key build -s \
  TeamA/demoapp-build -v
```

For extra credit, try the command again, but this time manually abort a build while it's going on: CLI will report a failure.

Try also running with the `-c` option, and verify that the build is skipped.

## Step 4. Managing Agents with the CLI

- **Exercise:** Make the `ssh-agent` offline using the CLI
  - Reason to provide is `SSH agent maintenance`
  - Verify using the GUI that the agent is offline
  - Put it online manually from the **Node Management** page

**TIP** Use the `help` and `offline-node` command

## Step 5. Creating a 3rd agent with the CLI

- Start by fetching the current `ssh-agent` configuration

- Use the `get-node` command to retrieve the **XML** configuration

**TIP**

The login command would help to avoid the CLI asking you for a passphrase, if you add one.

```
# Try the command to study file content
cloudbees-devbox $ java -jar jenkins-cli.jar \
  -i /ssh-keys/vagrant_insecure_key \
  get-node ssh-agent
...
# Persist on a local file
cloudbees-devbox $ java -jar jenkins-cli.jar \
  -i /ssh-keys/vagrant_insecure_key \
  get-node ssh-agent > agent-config.xml
cloudbees-devbox $ cat agent-config.xml
...
```

- The new agent will be launched on the same SSH machine, as a separate JVM.
  - Its name will be: `ssh-agent-2`
  - Edit the XML file to adapt the name accordingly
  - Create the new agent using the `create-node` command

**IMPORTANT**

**Read** the command usage before !

```
# Change the name in the XML file
cloudbees-devbox $ sed -i \
  's#name>ssh-agent<#name>ssh-agent-2<#g' \
  ./agent-config.xml
# Check the change
cloudbees-devbox $ cat agent-config.xml
# Create the 3rd agent
cloudbees-devbox $ java -jar jenkins-cli.jar \
  -i /ssh-keys/vagrant_insecure_key \
  create-node < agent-config.xml
# Verify creation:
cloudbees-devbox $ java -jar jenkins-cli.jar \
  get-node ssh-agent-2
```

- Finally, use the Jenkins GUI to validate you have a 3rd agent

## Manage Jenkins using its REST API

The goal of this exercise is to experiment with the Jenkins REST API.

#### IMPORTANT

There are 2 API documentation "entrypoints":

- [Jenkins Website](#)
- For **each** Jenkins Object, append `/api` to the URL to see dynamic documentation. Example for the `demoapp-build`: <http://localhost:5000/jenkins/job/demoapp-build/api>

## Launch a Job with the REST API

To demonstrate a simple usage of the Jenkins API, we'll re-deploy the staging application using only the `curl` command.

- Start by opening a shell in the **Devbox**
- Take a minute to read the API "Entrypoint" on your Jenkins instance: <http://localhost:5000/jenkins/api/>
- Command anatomy:
  - `curl` is used to send a **POST** HTTP request
  - Authentication is handled by Curl HTTP auth, using the `-u` flag
  - Our job needs a parameter; it is encoded as JSON for this example. (It could instead be encoded as XML, python, or other language)

```
cloudbees-devbox $ curl -X POST \
  http://jenkins:8080/jenkins/job/demoapp-staging-deployer/build/api \
  -u butler:butler \
  --data-urlencode json='{ "parameter": [ { "name": "DOCKER_IMAGE", "value": "demo-
  app:latest" } ] }'
```

- browse back to the GUI: you should see your build started and the delivery done.

That's all for this exercise !

## Journey Summary

This lab covered:

- Enabling Security in Jenkins
- Making our Jenkins a distributed build cluster
- How to implement complex CD Pipeline with Job **Promotion**
- How to use Folders
- How to use Jenkins CLI and API to automate Jenkins operations