

# Deploy di Jenkins 2.91 per Mondadori

**Attenzione** Se ti trovi dietro a un proxy e installi Jenkins su una VM in localhost, nessun servizio Git\* sarà in grado di connettersi ad esso.

Se non ti trovi dietro un proxy ti basta sostituire "localhost" con il tuo **IP pubblico** nei vari webhook.

## Installazione VM

- Download di Centos v6.7 da [http://archive.kernel.org/centos-vault/6.7/isos/x86\\_64/](http://archive.kernel.org/centos-vault/6.7/isos/x86_64/)

## Creazione macchina virtuale su GCP o simile:

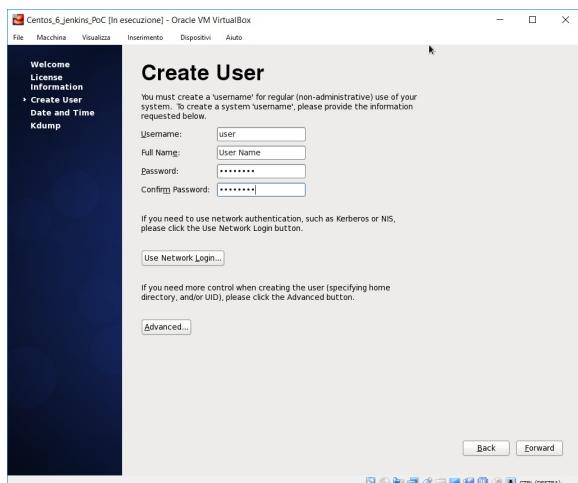
- Centos v6.7
- 8 GB RAM
- 20 GB Hdd

## Installazione di Centos 6.7

- Selezionare lingua inglese
- Selezionare lingua italiana per la tastiera
- Impostare l'orario su "Europe,Rome"
- Root Password : password



- Create a user : **user** con password : **password**



- Reboot
- Imposta il proxy (se sei in sede reply)

```

su
Password: *****

vi /etc/yum.conf

#aggiungi in fondo
proxy=http://proxy.reply.it:8080

vi /etc/profile

#aggiungi in fondo
export http_proxy=http://proxy.reply.it:8080
export https_proxy=https://proxy.reply.it:8080

source /etc/profile

```

- Imposta la connessione automatica al boot

```

vi /etc/sysconfig/network-scripts/ifcfg-eth0

#modifica
ONBOOT=yes

```

- Aggiorna il sistema

```
sudo yum update -y
```

- Se vuoi, installa le Virtual Box guest additions (io lo trovo molto comodo, michele le odia)

```

sudo yum update
sudo yum install gcc
sudo yum install kernel-headers
sudo yum install kernel-devel

```

## Installazione di jenkins

Per l'installazione di jenkins seguirò un mix tra questa [guida per Centos 7](#) e questa [guida specifica per Centos 6](#).

Ottenerne un utente NON root con privilegi di root

Il nostro utente *user* va più che bene a questo scopo, per garantirgli privilegi di root è necessario editare il file [/etc/sudoers](#) ma, mi raccomando, **NON FARLO MAI CON VIM**.

Usa invece:

```

su
Password: *****

visudo

```

Giusto per capire cosa andremo a fare: **root ALL=(ALL:ALL) ALL**

- *root* indica l'utente
- Il primo *ALL* indica che la regola si applica ad ogni host
- Il secondo *ALL* indica che l'utente root esegue comandi per ogni altro utente (tramite sudo)
- Il terzo *ALL* indica che può eseguire comandi per ogni gruppo
- L'ultimo *ALL* indica che la regola si applica a qualsiasi comando

Decomenta la riga

```
%wheel ALL=(ALL) ALL
```

In modo da garantire tutti i diritti agli utenti del gruppo wheel. Quindi aggiungi *user* al gruppo wheel

```
sudo usermod -aG wheel username  
reboot #per rendere valida la modifica
```

Puoi trovare maggiori informazioni su come gestire i diritti sudo a [questa pagina](#).

## Installazione di Jenkins da pacchetto RPM

Tutti i comandi dovranno essere fatti con l'utente **NON ROOT** tramite sudo.

### Installare java 1.8

Per prima cosa installiamo java, questa versione di jenkins richiede java 1.8

Connettiti alla pagina di [oracle](#), accetta la licenza e scarica **jdk-8u151-linux-x64.tar.gz**.

```
cd /opt  
wget http://download.oracle.com/otn-pub/java/jdk/8u151-b12/e758a0de34e24606bca991d704f6dcfb/jdk-  
8u151-linux-x64.rpm?AuthParam=1511780178_e971f91c277e5926c5a9b60179d896de  
  
sudo rpm -ivh jdk-8u151-linux-x64.rpm\?AuthParam\=1511780178_e971f91c277e5926c5a9b60179d896de  
  
java -version  
java version "1.8.0_151"  
Java(TM) SE Runtime Environment (build 1.8.0_151-b12)  
Java HotSpot(TM) 64-Bit Server VM (build 25.151-b12, mixed mode)  
  
# per salvare spazio  
sudo rm -rf jdk-8u151-linux-x64.rpm\?AuthParam\=1511780178_e971f91c277e5926c5a9b60179d896de
```

### Download del repo RPM

Scarica il pacchetto RPM di jenkins, puoi spostarti in [/etc/yum.repos.d](#) oppure eseguire direttamente

```
sudo wget -O /etc/yum.repos.d/jenkins.repo http://pkg.jenkins-ci.org/redhat/jenkins.repo
```

Importa il codice di verifica di validità del pacchetto

```
sudo rpm --import https://jenkins-ci.org/redhat/jenkins-ci.org.key
```

Se, per una qualche ragione, rpm non riesce a scaricare la chiave restituendo un errore 404 (come è successo a me):

```
sudo wget https://jenkins-ci.org/redhat/jenkins-ci.org.key
sudo rpm --import jenkins-ci.org.key
```

### Installazione con yum

Una volta fatto possiamo installare jenkins con yum.

```
sudo yum install jenkins
```

Terminata l'installazione, avviamo il servizio di jenkins e impostiamo l'avvio automatico

```
sudo service jenkins start
sudo chkconfig jenkins on
```

### File utili

E' stato creato un utente *jenkins* pensato per eseguire tutto quello che è legato al servizio, puoi modificare questo utente nel file di config , ma ricorda di cambiare i privilegi a `/var/log/jenkins, /var/lib/jenkins e /var/cache/jenkins.

Troverai i file di log in

Cosa	File
Config	/etc/sysconfig/jenkins
Log	/var/log/jenkins/jenkins.log

### Connessione al servizio e firewall

Usa `service jenkins start/stop/status` per gestire il servizio.

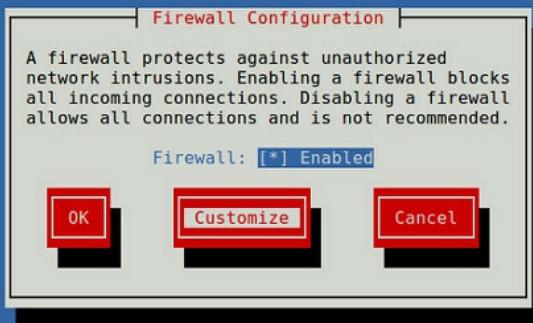
All'avvio del servizio, potrai interagire con jenkins tramite il tuo browser alla porta **8080**, il firewall potrebbe darti dei disagi, per disabilitarlo, in Centos 6.7 non possiamo usare `firewalld`, utilizzeremo la TUI di `system-config-firewall`

```
sudo iptables -L #per vedere le attuali regole
```

Per modificare permanentemente il firewall, in modo da aprire la porta **tcp:8080**

```
sudo system-config-firewall-tui
```

system-config-firewall



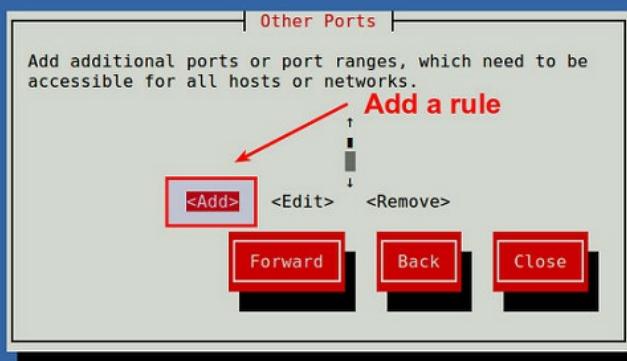
<Tab>/<Alt-Tab> between elements | <Space> selects | <F12> next screen

system-config-firewall



<Tab>/<Alt-Tab> between elements | <Space> selects | <F12> next screen

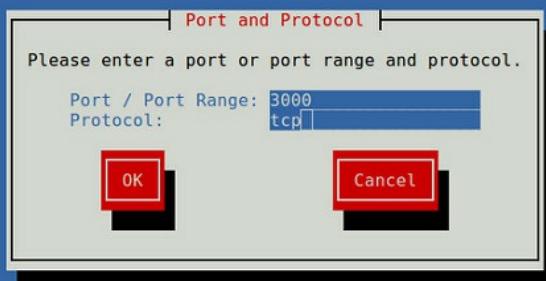
system-config-firewall



<Tab>/<Alt-Tab> between elements | <Space> selects | <F12> next screen

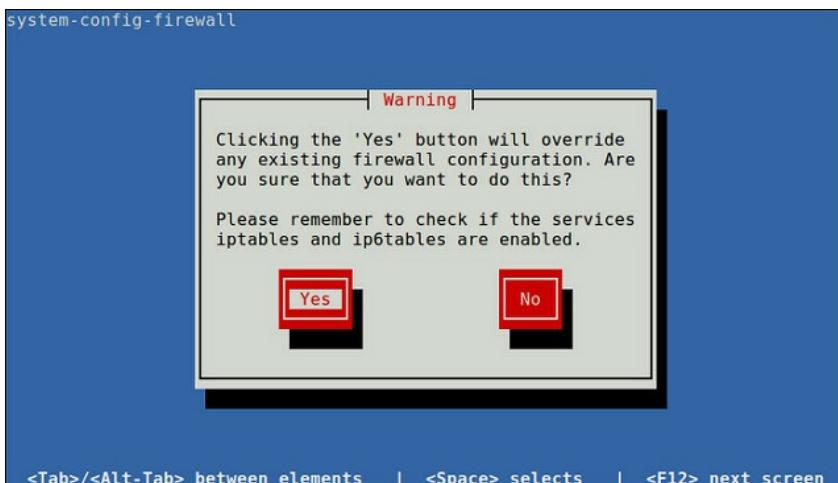
system-config-firewall

Ovviamente inserisci la porta 8080



<Tab>/<Alt-Tab> between elements | <Space> selects | <F12> next screen

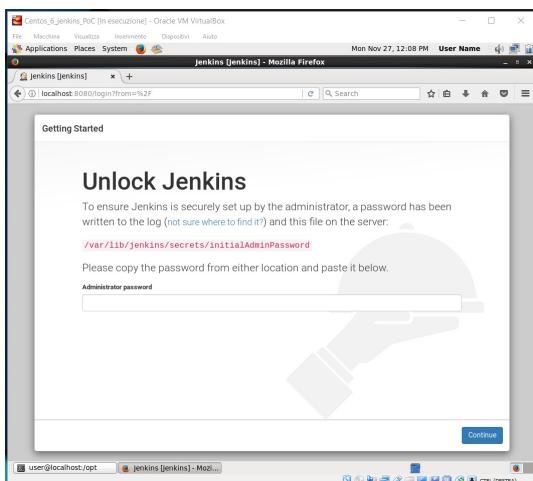
OK -> Back -> Close -> Ok



## Avvio e utilizzo di jenkins

```
sudo service jenkins start
Starting Jenkins
[ OK ]
```

E collegati alla pagina <http://35.196.238.203:8080>, dovrebbe aprirsi una pagina del genere.



Scopriamo la password autogenerata

```
sudo cat /var/lib/jenkins/secrets/initialAdminPassword
ab21de47d4cf4d70856f57d953fb84bf
```

## Installazione plugin

Quindi installiamo i plugin che ci servono (io devo fare una prova, ho installato quelli di default).



**Attenzione** se sei dietro a un proxy questa procedura fallirà malamente, questo perché jenkins ha bisogno di una configurazione proxy ma nessuno ci ha dato la possibilità di configurarlo.

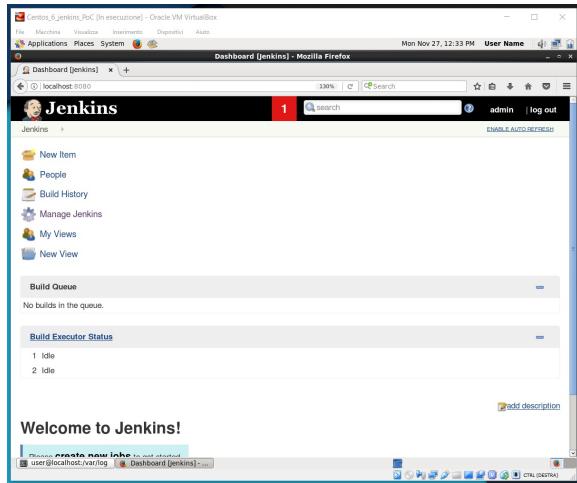
Non preoccuparti per ora, fai continua e ci pensiamo dopo.

## Creazione utente admin

Per comodità ho creato un utente admin default:

- Username = admin
- Password = admin

Se tutto va bene dovrebbe aprirsi l'homepage di jenkins.



## Configurare Jenkins

### Proxy

Manage Jenkins -> Manage Plugins -> "Advanced" tab. Qui puoi impostare il proxy reply.

A screenshot of the Jenkins 'HTTP Proxy Configuration' page. It features two input fields: 'Server' with the value 'proxy.reply.it' and 'Port' with the value '8080'. Both fields have a question mark icon in the top right corner. Above these fields is a navigation bar with tabs: 'Updates', 'Available', 'Installed', and 'Advanced', where 'Advanced' is highlighted.

### Enable Global Security

Manage Jenkins -> Configure Global Security

Se la spunta non è presente, spuntare *Enable Security*

//TODO Spuntare Matrix-based security e configura un utente admin (non c'è il checkbox) (<https://wiki.jenkins.io/display/JENKINS/Matrix-based+security>)

### Installare Plugin

Manage Jenkins -> Manage Plugins -> "Available" tab

Seleziona i plugin che desideri, per la mia Demo ho selezionato solo **GIT plugin** e **GitHub Plugin** dal momento che il progetto da gestire si trova su github. In ogni caso jenkins ha installato anche tutti i plugin che erano falliti nell'installazione iniziale.

Esegui l'installazione con restart e, al termine, torna alla Top Page.

### Configurare GIT

Se Git non è installato nell'host questo non funzionerà nemmeno su jenkins (grazie al cazzo).

```
sudo yum install git -y

# Per sapere dove è stato installato
sudo which git
/usr/bin/git
```

Inoltre dovremo configuralo in Jenkins, andiamo quindi in Manage Jenkins -> Configure System -> sezione Git Plugin e inseriamo nome utente e email dell'account usato nel repository.

Git plugin

Global Config user.name Value  
dpasquali

Global Config user.email Value  
d.pasquali@reply.it

Create new accounts based on author/committer's email

Non possiamo però ancora accedere al repository, la configurazione sarà completata all'inserimento del link (prossia sezione).

## Demo Git Hub

Nella homepage creiamo un *New Job* dal pulsante centrale se non ne abbiamo o da *New Item* a sinistra.

Jenkins

New Item

People

Build History

Manage Jenkins

My Views

Credentials

New View

Build Queue

No builds in the queue.

Build Executor Status

1 Idle

2 Idle

Welcome to Jenkins!

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

Metti il nome che preferisci e seleziona *Freestyle Project*

Enter an item name

Demo

» Required field

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

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

OK

Imposta la descrizione, spunta *GitHub project* e inserisci il link del repository (intendo l'**URL nella barra di navigazione**)

General Build Triggers Advanced Project Options Pipeline

Pipeline name: Demo

Description: Jenkins demo with GitHub

[Plain text] Preview

Discard old builds ?

Do not allow concurrent builds ?

GitHub project ?

Project url: <https://github.com/dpasqualiReply/JenkinsDemo> ?

[Advanced...](#)

Qui invece imposta il **link per clonare il repository** e le credenziali dell'account GitHub

### Source Code Management

None

Git

**Repositories**

Repository URL: <https://github.com/dpasqualiReply/JenkinsDemo.git> ?

Credentials: dpasqualiReply/\*\*\*\*\*\*\*\*\* Add Advanced...

**Branches to build**

Branch Specifier (blank for 'any'): \*/master X ?

**Repository browser** (Auto) ?

Imposta poi il build del job Jenkins in modo che venga eseguito ogni volta che viene fatto un commit git.

Inoltre puoi impostare una serie di comandi che vengono fatti durante la build, strumento potentissimo, **da imparare ad usare**.

### Build Triggers

Trigger builds remotely (e.g., from scripts) ?

Build after other projects are built ?

Build periodically ?

GitHub hook trigger for GITScm polling ?

Poll SCM ?

---

### Build

Execute shell

Command: echo "Build Done" X ?

## Configurazione Webhook GitHub

Un webhook è un collegamento che permette di scatenare un determinato evento tra due differenti servizi web. In questo caso, vogliamo fare in modo che, nel momento in cui viene fatto il commit di su GitHub, venga fatta una POST <http://35.196.238.203:8080/github-webhook>. Questa chiamata scatenerà build, test, deploy e compagnia bella.

La configurazioni deve essere fatta lato GitHub, al momento è implementata come un servizio, non come un webhook, andiamo quindi nella pagina **Integration & Services**

The screenshot shows the GitHub 'Installed GitHub Apps' page. On the left, there is a sidebar with options like 'Options', 'Collaborators', 'Webhooks', 'Integrations & services' (which is currently selected), and 'Deploy keys'. The main area is titled 'Installed GitHub Apps' and contains a section for 'Services'. A modal window titled 'Available Services' is open, showing a search bar with 'jenk' and a list of results. The 'Jenkins (GitHub plugin)' option is highlighted with a blue background.

Imposta il link del server su cui è presente jenkins.

The screenshot shows the Jenkins GitHub plugin configuration page. It includes a success message 'Okay, the test payload is on its way.' and a note about Jenkins being a popular CI server. Below this, there are 'Install Notes' and a 'Jenkins hook url' input field containing 'http://35.196.203.8080/github-webhook/'. There is also a checked 'Active' checkbox and buttons for 'Update service' and 'Delete service'.

## Demo GitLab

Seguirò questa [guida](#).

GitLab ha davvero un sacco di problemi con Jenkins, da quel che dice il web funziona solo con la versione enterprise di GitLab, oppure con il tool di CI proprietario di GitLab.

Al momento utilizzo:

- GitLab 9.5.4
- GitLab Plugin [1.5.2](#)

Per questa combinazione di versioni esiste una [Know Issue](#) e [issue2](#) per cui il bottone "Webhook" test di GitLab lancia una POST errata, vedremo al momento del test come gestirlo.

### Chiave SSH per il deploy

Per prima cosa è necessario configurare una chiave RSA per il deploy, questa procedura è comoda perché permette al server Jenkins di accedere ai repository di nostro interesse, senza la necessità di creare utenti "dummy" con diritti custom.

### Generare la chiave

```

su
Password: ****

cd /var/lib/jenkins/.ssh
sudo -u jenkins ssh-keygen

```

```

Generating public/private rsa key pair.
Enter file in which to save the key (/var/lib/jenkins/.ssh/id_rsa): id_rsa
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in id_rsa.
Your public key has been saved in id_rsa.pub.
The key fingerprint is:
SHA256:pELwRD6I+aWPm3XZ2mbZxh018dLNL4BBSm9/LPbzY jenkins@devops-worker
The keys randomart image is:
+---[RSA 2048]---+
| ..o      .=.   |
| o *      o + .  |
|o . *     .. + + o |
| . + . o  . =.. o|
| o . . S  .*o. o|
|   o . o o Bo= ..|
|   . o o o o *. .|
|   + .      . oEo|
|   o          =+|
+---[SHA256]---+

```

Una volta generata la chiave, copiala e inseriscila nel repository GitLab

```

cat id_rsa.pub
ssh-rsa
AAAAB3NzaC1yc2EAAAQABAAQCB9WxJsoJX11XpMQk2bAaRSFKzYjBq5NeNOKhzvCP2cnyKm0NK409IlWpDpTec5qOSaOsJ6e
0IGftKHXRXLnH39j0WZVGEbZ096pzZiyGZAMdq2xM+Azm6CpcS6V1/XGyer8gtsvKDj5kD5q8EUfALKV6y642V7dh6UiCacyQuL6
tF6xumaCLW+20ASUmkuCqmDQ9aSi08c8HsxIIAQmZs6XCS8ecCro08LyhLCLSguwwaMFtWpomINTOApH/6k9nQDWWM3K1/1+uUoo
mAyu7G1zj7gHAp4oxX05/rIZ4l0Ynvyw9ods8/NWmy6349/5nXnITuD0afrQLJnaj0Dv jenkins@devops-worker

```

## Inserire la chiave su GitLab

Apri il repository GitLab a cui vuoi applicare la Chiave di Deploy

The screenshot shows the 'Project' tab of the 'jenkins-CI-test' repository settings. It includes the repository name 'jenkins-CI-test', a description 'PoC of Jenkins CI with GitLab', and a URL 'http://jarvis.datareply.eu/d.pas'. Navigation links for Home, Activity, and Cycle Analytics are also visible.

The screenshot shows the main repository page for 'jenkins-CI-test'. A message at the top states 'The repository for this project is empty'. Below it, instructions for pushing files and creating initial files like README, LICENSE, or .gitignore are provided. A note at the bottom indicates that pushes require owner or master permission.

Settings -> Repository -> Deploy Keys

**Protected Branches****Expand**

Keep stable branches secure and force developers to use merge requests.

**Protected Tags****Expand**

Limit access to creating and updating tags.

**Deploy Keys****Expand**

Deploy keys allow read-only or read-write (if enabled) access to your repository. Deploy keys can be used for CI, staging or production servers. You can create a deploy key or add an existing one.

Dai un nome alla chiave, copiala e spunta *Write access allowed***Deploy Keys****Collapse**

Deploy keys allow read-only or read-write (if enabled) access to your repository. Deploy keys can be used for CI, staging or production servers. You can create a deploy key or add an existing one.

Create a new deploy key for this project

Title

jenkins-ci

Key

```
ssh-rsa
AAAAAB3NzaC1yc2EAAAQABAAQDiBtL5IBeSFxOGq9ieJsvEBCn4DIdXp/5D+qrIQP74m/hk29O9pDaE6nRP5bTgst6B3kqg6kboB
C+LbhnmOS5b0+Fc0zFBxuOl8NbtoH1814tJJ5k6j7LUnMqjZ2RoovyHskWmsdn08THBRNCNsGq5o5ld53GfNA1EPaAwSFukGKn7p/v
1gdkrTAv07tjxxkYxgh7G8hxLUKUOdBnOzSrzPw02E2wWkXYwjNidjOLBCik91n0dflyH50j36PcfOUoEnUvqZRbMoAHyPR4lEdhkZG1/T
GgNBjaDwQ92QThenBPk61fGiHeR7XsIUTZMmVbjh4EU8nE6DbR/t dario_pasquali93@devops-worker
```

Paste a machine public key here. Read more about how to generate it [here](#) **Write access allowed**

Allow this key to push to repository as well? (Default only allows pull access.)

**Add key****Inserire la chiave in jenkins**

La stessa chiave deve essere inserita in Jenkins, per farlo vai in Credentials -&gt; System -&gt; freccina nel dominio -&gt; Add Credentials

The screenshot shows the Jenkins System configuration page. On the left, there's a sidebar with links like Nuovo Elemento, Utenti, Cronologia compilazioni, Relazioni fra progetti, Controlla impronta file, Gestisci Jenkins, Le mie viste, and New View. The main area is titled "System" and has a table with two columns: "Domain" and "Description". The first row shows "Global credentials (unrestricted)" under "Domain" and "Credentials that should be available irrespective of domain specification to requirements matching." under "Description". A blue box highlights the "Add credentials" button in the "Domain" column. The status bar at the bottom right shows "admin | dis".

Domain	Description
Global credentials (unrestricted)	Credentials that should be available irrespective of domain specification to requirements matching.

Scope Global (Jenkins, nodes, items, all child items, etc)

Username jenkins

Private Key  Enter directly  
 From a file on Jenkins master  
File /var/lib/jenkins/.ssh/id\_rsa

From the Jenkins master ~/.ssh

Passphrase

ID cf832894-f3dc-4707-8f10-fd90ddcfe9e

Description

**Save**

## Creare un Access Token GitLab

Accedere alle configurazioni del profilo in GitLab -> Access Tokens

User Settings

Profile Account Applications Chat Access Tokens Emails Password Notifications SSH Keys GPG Keys Preferences Authentication

Public Avatar

You can upload an avatar here or change it at gravatar.com

Upload new avatar

Browse file... No file chosen

The maximum file size allowed is 200KB.

Main settings

This information will appear on your profile.

Name Dario Pasquali

User ID 119

Email d.pasquali@reply.it

Dai un nome al tuo token e seleziona una expiration date

### Personal Access Tokens

You can generate a personal access token for each application you use that needs access to the GitLab API.

You can also use personal access tokens to authenticate against Git over HTTP. They are the only accepted password when you have Two-Factor Authentication (2FA) enabled.

### Add a personal access Token

Pick a name for the application, and we'll give you a unique personal access Token.

#### Name

jenkins-access-token

#### Expires at

2018-10-31

#### Scopes

- api** Access your API
- read\_user** Read user information
- read\_registry** Read Registry

**Create personal access token**

Viene quindi generato e mostrato il tuo Personal Access Token super segretissimo, mi raccomando **SALVALO** perchè non te lo faranno vedere mai più.

Metti quindi il token in jenkins creando una nuova Credentials come prima.

Jenkins

Credentials > System > Global credentials (unrestricted) >

**Kind:** GitLab API token

**Scope:** Global (Jenkins, nodes, items, all child items, etc)

**API token:**

**ID:** gitlab1

**Description:** gitlab personal access token for test

**OK**

## Configurare la connessione tra Jenkins e GitLab

Torniamo in Manage Jenkins -> Configure System.

Sotto **Git Lab** sarà tutto rosso-incazzato, inserisci le informazioni richieste e sarà contento.

Fai **attenzione**, il \*Gitlab host URL non è quello del tuo progetto, ma la radice, cioè il server su cui gira Gitlab.

Premendo *Test Connection* potrai ottenere un dolcissimo e gratificante *Success*.

**Gitlab**

Enable authentication for '/project' end-point

GitLab connections

Connection name:	Mondadori GitLab
Gitlab host URL:	http://jarvis.datareply.eu
Credentials:	GitLab API token (gitlab personal access token for ! <input type="button" value="Add"/>
API-Level:	autodetect
Ignore SSL Certificate Errors:	<input type="checkbox"/>
Connection timeout (in seconds):	10
Read timeout (in seconds):	10
Success:	<input type="button" value="Test Connection"/>
<input type="button" value="Elimina"/>	
<input type="button" value="Aggiungi"/>	

## Creazione del Job Jenkins

Esattamente come per Github, notare che la *GitLab Connection* viene inserita da sola



Nella sezione *Gestione Codice Sorgente* inserisci

Config	Value
URL	url del repository Git Lab
Credentials	Le credenziali RSA configurate con l'utente jenkins
Name	origin
Refspec	+refs/heads/:refs/remotes/origin/ +refs/merge-requests//head:refs/remotes/origin/merge-requests/
Branch	origin/\${gitlabSourceBranch}

Impostiamo in modo che la compilazione avvenga ogni volta che avviene un qualche evento sul repository. Ovviamente configurare sulla base delle proprie esigenze.

**Attenzione** Segnati l'URL del servizio, nel mio caso <http://35.196.238.203:8080/project/GitLab-demo>.

**Trigger compilazione**

- Attiva una compilazione da remoto (ad esempio da uno script)
- Compila dopo aver compilato gli altri progetti
- Compila periodicamente
- Build when a change is pushed to GitLab. GitLab CI Service URL: http://35.196.238.203:8080/project/GitLab-demo

Enabled GitLab triggers	Push Events	<input checked="" type="checkbox"/>
Opened Merge Request Events	<input checked="" type="checkbox"/>	
Accepted Merge Request Events	<input checked="" type="checkbox"/>	
Closed Merge Request Events	<input checked="" type="checkbox"/>	
Rebuild open Merge Requests	Never	
Comments	<input checked="" type="checkbox"/>	
Comment (regex) for triggering a build	Jenkins please retry a build	
<a href="#">Avanzate...</a>		
<input type="checkbox"/> GitHub hook trigger for GITScm polling		
<input type="checkbox"/> Esegui polling sul sistema di controllo del codice sorgente		

Aggiungi i passi di compilazione utili: **QUI VERRANNO INSERITI I TEST DIREI**

**Compila**

Esegui shell

```
Command echo "Jenkins build Done"
echo "Let's do the deploy"
```

Si veda [l'elenco delle variabili d'ambiente disponibili](#)

[Avanzate...](#)

**Aggiungi passo di compilazione ▾**

- Esegui comando batch Windows
- Esegui shell
- Invoca target Maven principali
- Invoke Ant
- Invoke Gradle script
- Run with timeout
- Set build status to "pending" on GitHub commit

Aggiungi un'azione di post compilazione, possiamo per esempio inviare una email a destinatari custom, oppure, come in questo caso, mostrare sulla GUI di GitLab lo stato del commit.

**Azioni post compilazione**

Publish build status to GitLab commit (GitLab 8.1+ required)

Build name

Mark unstable builds as success

**Aggiungi azione post compilazione ▾**

## Creazione webhook lato GitLab

Lato GitLab è necessario configurare il Webhook in modo che venga fatta una POST in risposta a determinati eventi. Ovviamente configura il webhook in modo consistente con la configurazione degli eventi in jenkins.

The screenshot shows the 'Integrations' section of the GitLab 'Settings' page. At the top, there are tabs for General, Integrations, Repository, and Pipelines. The 'Integrations' tab is selected. Below it, there's a heading 'Webhooks can be used for binding events when something is happening within the project.' A URL input field contains 'http://35.196.238.203:8080/project/GitLab-demo'. A 'Secret Token' input field is present below it. A note says, 'Use this token to validate received payloads. It will be sent with the request in the X-Gitlab-Token HTTP header.' Under the 'Trigger' section, several event types are listed with checkboxes: Push events (checked), Tag push events, Comments, Issues events, Confidential Issues events, Merge Request events, Job events, Pipeline events, and Wiki Page events. Most checkboxes are checked. At the bottom, there are 'SSL verification' and 'Enable SSL verification' checkboxes, both unchecked. A green 'Add webhook' button is at the bottom right.

## Test

Ok, dovrebbe essere tutto a posto, ora fai qualche test.

Come già detto, esiste una know issue per cui il bottone "Webhook Test" di Git Lab (nella pagina di configurazione del webhook) invia una chiamata POST con un header errato. Per testare che tutto funzioni possiamo quindi effettuare un normale commit oppure usare un tool come [Postman](#) per simulare la chiamata REST.

### Postman

#### URL

```
http://35.196.238.203:8080/project/GitLab-demo
```

#### Headers

```
Content-Type:application/json
X-Gitlab-Event:Push Hook
X-Gitlab-Token:d5188a5c982e87a9a9207e13bcb94677
```

#### Body

```
{
  "object_kind": "push",
  "event_name": "push",
  "before": "e35bfb275c5953be8acf19284618061749684de8",
  "after": "062380a3f6421bc376568f19b0445385fbdeaf2c",
  "ref": "refs/heads/master",
  "checkout_sha": "062380a3f6421bc376568f19b0445385fbdeaf2c",
  "message": null,
  "user_id": 119,
  "user_name": "Dario Pasquali",
  "user_username": "d.pasquali",
  "user_email": "d.pasquali@reply.it",
  "user_avatar": "http://www.gravatar.com/avatar/3fd07fe896722b6563f7ef62824d9bec?s=80&d=identicon",
  "project_id": 147,
  "project": {
    "name": "jenkins-CI-test",
    "description": "PoC of Jenkins CI with GitLab",
    "web_url": "http://jarvis.datareply.eu/d.pasquali/jenkins-CI-test",
  }
}
```

```

"avatar_url": null,
"git_ssh_url": "git@jarvis.datareply.eu:d.pasquali/jenkins-CI-test.git",
"git_http_url": "http://jarvis.datareply.eu/d.pasquali/jenkins-CI-test.git",
"namespace": "d.pasquali",
"visibility_level": 20,
"path_with_namespace": "d.pasquali/jenkins-CI-test",
"default_branch": "master",
"ci_config_path": null,
"homepage": "http://jarvis.datareply.eu/d.pasquali/jenkins-CI-test",
"url": "git@jarvis.datareply.eu:d.pasquali/jenkins-CI-test.git",
"ssh_url": "git@jarvis.datareply.eu:d.pasquali/jenkins-CI-test.git",
"http_url": "http://jarvis.datareply.eu/d.pasquali/jenkins-CI-test.git"
},
"commits": [
{
  "id": "062380a3f6421bc376568f19b0445385fbdeaf2c",
  "message": "test\n",
  "timestamp": "2017-11-27T19:18:17+01:00",
  "url": "http://jarvis.datareply.eu/d.pasquali/jenkins-CI-
test/commit/062380a3f6421bc376568f19b0445385fbdeaf2c",
  "author": {
    "name": "d.pasquali",
    "email": "d.pasquali@reply.it"
  },
  "added": [
    ],
  "modified": [
    "prova.sh"
  ],
  "removed": [
    ]
},
{
  "id": "e35bfb275c5953be8acf19284618061749684de8",
  "message": "initial commit\n",
  "timestamp": "2017-11-27T19:15:12+01:00",
  "url": "http://jarvis.datareply.eu/d.pasquali/jenkins-CI-
test/commit/e35bfb275c5953be8acf19284618061749684de8",
  "author": {
    "name": "d.pasquali",
    "email": "d.pasquali@reply.it"
  },
  "added": [
    "prova.sh"
  ],
  "modified": [
    ],
  "removed": [
    ]
}
],
"total_commits_count": 2,
"repository": {
  "name": "jenkins-CI-test",
  "url": "git@jarvis.datareply.eu:d.pasquali/jenkins-CI-test.git",
  "description": "PoC of Jenkins CI with GitLab",
  "homepage": "http://jarvis.datareply.eu/d.pasquali/jenkins-CI-test",
  "git_http_url": "http://jarvis.datareply.eu/d.pasquali/jenkins-CI-test.git",
  "git_ssh_url": "git@jarvis.datareply.eu:d.pasquali/jenkins-CI-test.git",
  "visibility_level": 20
}
}

```

}

Non avremo alcun feedback su Postman ma, aprendo la dashboard jenkins, vedrmo che la build è in corso...

Tutte +

S	M	Nome ↓	Ultima compilazione riuscita
		<a href="#">GitHubDemo</a>	22 h - #4
		<a href="#">gitlab</a>	23 min - #17

Ico In corso Legenda RSS per tu

...e poi viene aggiornato il conteggio dall'ultima compilazione

Tutte +

S	M	Nome ↓	Ultima compilazione riuscita
		<a href="#">GitHubDemo</a>	22 h - #4
		<a href="#">gitlab</a>	7.9 s - #19

Icona: [S](#) [M](#) [L](#) Legenda E

Cliccando sul job possiamo vedere il report di build, in questo caso essendo una push di test, non avremo nessun cambiamento.

## Compilazione #19 (28-nov-2017 14.12.09)

Started by GitLab push by Dario Pasquali



Nessun cambiamenti.



Started by GitLab push by Dario Pasquali



Revisione: 062380a3f6421bc376568f19b0445385fbdeaf2c

- origin/master

Commit & Push

```

MINGW64:/c/Users/d.pasquali/Desktop/DevOps/MondadoriCI/jenkins-CI-test
d.pasquali@DAT18226 MINGW64 ~/Desktop/DevOps/MondadoriCI/jenkins-CI-test (master)
$ git add --all
d.pasquali@DAT18226 MINGW64 ~/Desktop/DevOps/MondadoriCI/jenkins-CI-test (master)
$ git commit -m "tutorial GitLab CI"
[master fafb8bb] tutorial GitLab CI
 2 files changed, 25 insertions(+)
 create mode 100644 gitlabCI-tutorial/img/ci_pipeline.png
 create mode 100644 gitlabCI-tutorial/tutorial.md

d.pasquali@DAT18226 MINGW64 ~/Desktop/DevOps/MondadoriCI/jenkins-CI-test (master)
$ git push

```

Ripetiamo il test facendo un commit & push.

Non appena viene fatto il push parte il build sulla dashboard Jenkins.

The Jenkins dashboard shows two builds listed under the 'Tutte' tab:

S	M	Nome ↓	Ultima compilazione riuscita
		GitHubDemo	22 h - #4
		gitlab	8 min 27 s - #19

Below the main dashboard, there are two sections:

- Elenco compilazioni**: Shows 'Nessuna compilazione in coda.'
- Stato esecutore compilazione**: Shows two entries: 'Inattivo' and 'gitlab' (status bar indicates '#20').

Al termine del quale potremo ammirare il nuovo cambiamento avvenuto con successo.

## Compilazione #20 (28-nov-2017 14.20.34)

Started by GitLab push by Dario Pasquali

The job details page for compilation #20 shows the following information:

- Cambiamenti**: A list containing '1. tutorial GitLab CI (dettaglio)'.
- Started by GitLab push by Dario Pasquali**
- Revisione**: fafb8bbf9402d4503da7b9bbcec9b0836b5f6c83
  - origin/master

E il report console, dove possiamo notare i nostri passi di compilazione.

## Output terminale

```
Started by GitLab push by Dario Pasquali
Building in workspace /var/lib/jenkins/workspace/gitlab
> git rev-parse --is-inside-work-tree # timeout=10
Fetching changes from the remote Git repository
> git config remote.origin.url git@jarvis.datareply.eu:d.pasquali/jenkins-CI-test.git # timeout=10
Fetching upstream changes from git@jarvis.datareply.eu:d.pasquali/jenkins-CI-test.git
> git --version # timeout=10
using GIT_SSH to set credentials
> git fetch --tags --progress git@jarvis.datareply.eu:d.pasquali/jenkins-CI-test.git +refs/heads/*:refs/remotes/origin/* +refs/heads:refs/remotes/origin/merge-requests/*
> git rev-parse remotes/origin/master^{commit} # timeout=10
> git branch -a -v --no-abbrev --contains fafb8bbf9402d4503da7b9bbcec9b0836b5f6c83 # timeout=10
Checking out Revision fafb8bbf9402d4503da7b9bbcec9b0836b5f6c83 (origin/master)
> git config core.sparsecheckout # timeout=10
> git checkout -f fafb8bbf9402d4503da7b9bbcec9b0836b5f6c83
Commit message: "tutorial Gitlab CI"
> git rev-list 062380a3f6421bc376568f19b0445385fbdeaf2c # timeout=10
[gitlab] $ /bin/sh -xe /tmp/jenkins4403415033581496660.sh
+ echo 'Jenkins build Done'
Jenkins build Done
+ echo 'Let's do the deploy'
Let's do the deploy
Finished: SUCCESS
```

## Fallimento

Si noti che, in caso di fallimento, sia jenkins che GitLab mostreranno il l'evento. Per testare questa casistica ho inserito una **istruzione\_inesistente** tra i passi di build, tutto il job fallisce se anche uno solo dei passi fallisce (anche se jenkins mostra la percentuale di completamento).

## Output terminale

```
Started by GitLab push by Dario Pasquali
Building in workspace /var/lib/jenkins/workspace/gitlab
> git rev-parse --is-inside-work-tree # timeout=10
Fetching changes from the remote Git repository
> git config remote.origin.url git@jarvis.datareply.eu:d.pasquali/jenkins-CI-test.git # timeout=10
Fetching upstream changes from git@jarvis.datareply.eu:d.pasquali/jenkins-CI-test.git
> git --version # timeout=10
using GIT_SSH to set credentials
> git fetch --tags --progress git@jarvis.datareply.eu:d.pasquali/jenkins-CI-test.git +refs/heads/*:refs/
requests/*:head:refs/remotes/origin/merge-requests/*
> git rev-parse remotes/origin/master^{commit} # timeout=10
> git branch -a -v --no-abbrev --contains 11af30d95bee5824ff0f4fd70931f91f6f9cf0d9 # timeout=10
Checking out Revision 11af30d95bee5824ff0f4fd70931f91f6f9cf0d9 (origin/master)
> git config core.sparsecheckout # timeout=10
> git checkout -f 11af30d95bee5824ff0f4fd70931f91f6f9cf0d9
Commit message: "test fail"
> git rev-list fafb8bbf9402d4503da7b9bbcec9b0836b5f6c83 # timeout=10
[gitlab] $ /bin/sh -xe /tmp/jenkins4261353647124123631.sh
+ echo 'Jenkins build Done'
Jenkins build Done
+ istruzione_inesistente
/tmp/jenkins4261353647124123631.sh: line 3: istruzione_inesistente: command not found
Build step 'Execute shell' marked build as failure
Finished: FAILURE
```

Su GitLab possiamo osservare la differenza tra un push terminato con successo o fallimento.



The screenshot shows the GitLab commit history for the 'jenkins-CI-test' branch. It displays two successful commits from Dario Pasquali and one failed commit from 'test'. The failed commit has a red status indicator and is labeled 'Commit: failed'. The commit message 'test fail' is visible, along with the error message 'istruzione\_inesistente: command not found'. The commit history also shows a successful commit from 'tutorial GitLab CI' and another from 'test'.

Commit	Author	Date	Status	Message	Actions
master	jenkins-CI-test				Filter by commit message
28 Nov, 2017 2 commits					
test fail	Dario Pasquali	committed 3 minutes ago	Commit: failed	11af30d9	Browse Files
tutorial GitLab CI	Dario Pasquali	committed 13 minutes ago	Success	faf8bbf	Browse Files
27 Nov, 2017 2 commits					
test	Dario Pasquali	committed about 20 hours ago	Success	062380a3	Browse Files
initial commit	Dario Pasquali	committed about 20 hours ago	Success	e35bfb27	Browse Files