FREE eBook

LEARNING gitlab

#gitlab

Table of Contents

About 1

Chapter 1: Getting started with gitlab 2

Remarks 2

Versions 2

Examples 3

Installation or Setup 3

Chapter 2: Android CI Configuration 5

Examples 5

Build Tools 24.0.0 - Android 5

Chapter 3: Continuous integration 6

Introduction 6

Remarks 6

Examples 6

Runner installation 6

Debian, Ubuntu and CentOS 6

Windows 7

Runner configuration 7

Setup Gitlab CI to allow cloning other private repositories 8

Chapter 4: Google Cloud SDK CI Configuration 10

Examples 10

Open JDK 8 Docker Example 10

Credits 11

About

Chapter 1: Getting started with gitlab

Remarks

GitLab is a web-based version control software based on git and adds additional features such as branch management bug tracking and continuous integration. It is developed in Ruby.

GitLab Community Edition (CE) is being developed open-source and uses the MIT-License. Since August 2013 GitLab Inc. is providing the enterprise-grade *Enterprise Edition (EE)* which includes more features than the Community Edition.

GitLab.com is the Software as a Service alternative and runs itself on a Enterprise Edition. It is free of charge and supports private and public repositories.

Versions

Version	Release Date
8.15	2016-12-22
8.14	2016-11-22
8.13	2016-10-22
8.12	2016-09-22
8.11	2016-08-22
8.10	2016-07-22
8.9	2016-06-22
8.8	2016-05-22
8.7	2016-04-22
8.6	2016-03-22
8.5	2016-02-22
8.4	2016-01-22
8.3	2015-12-22
8.2	2015-11-22
8.1	2015-10-22

Version	Release Date
8.0	2015-09-22

Examples

Installation or Setup

This is a short summary of the GitLab guide on Install a GitLab CE Omnibus package.

Requirements

In order to install the GitLab Community Edition on your server, you should read the requirements page. To make it brief, the recommended requirements are:

- **OS:** Ubuntu, Debian, CentOS, RHEL
- **Ruby version:** Ruby (MRI) 2.1.x, currently does not work with versions 2.2 or 2.3.
- **CPU:** 2 cores (supports up to 500 users)
- **Memory:** 2 GB (supports up to 100 users)
- **Database:** PostgreSQL

Installation

The recommended method is to install the Omnibus package, which is fast to install. It contains GitLab and all its dependencies (Ruby, PostgreSQL, Redis, Nginx, Unicorn, etc.). For other methods check out the GitLab's installation options

With Ubuntu 16.04 as the recommended OS, this guide describes the installation steps on Debian based distributions. For CentOS, RHEL, Oracle Linux and Scientific Linux, please refer to the original guides:

- CentOS 6 (and RedHat/Oracle/Scientific Linux 6)
- CentOS 7 (and RedHat/Oracle/Scientific Linux 7)

Ubuntu, Debian, Raspberrian

Install necessary dependencies. If you use Postfix select 'Internet Site' during setup

```
sudo apt-get install curl openssh-server ca-certificates postfix apt-transport-https
curl https://packages.gitlab.com/gpg.key | sudo apt-key add -
```

Add the Gitlab package server and install the package

```
sudo curl -sS https://packages.gitlab.com/install/repositories/gitlab/raspberry-
pi2/script.deb.sh | sudo bash
sudo apt-get install gitlab-ce
```

If you do not want to install the repository through a piped script, download the package manually and install it using

```
dpkg -i gitlab-ce_<version>.deb
```

Now configure and start GitLab

```
sudo gitlab-ctl reconfigure
```

Finally browse to the hostname and login. At first you will be redirected to provide a password for the initial administrator account. After that you're able to login in. The **default administrator account username** is **root**.

Read Getting started with gitlab online: https://riptutorial.com/gitlab/topic/2046/getting-started-with-gitlab

Chapter 2: Android CI Configuration

Examples

Build Tools 24.0.0 - Android

```
image: jangrewe/gitlab-ci-android
before_script:
  - apt-get --quiet update --yes
  - apt-get --quiet install --yes wget tar unzip lib32stdc++6 lib32z1 openjdk-8-jdk
  - echo y | ${ANDROID_HOME}/tools/android --silent update sdk --no-ui --all --filter android-
24
  - echo y | ${ANDROID_HOME}/tools/android --silent update sdk --no-ui --all --filter
platform-tools
  - echo y | ${ANDROID_HOME}/tools/android --silent update sdk --no-ui --all --filter build-
tools-24.0.0
  - echo y | ${ANDROID_HOME}/tools/android --silent update sdk --no-ui --all --filter extra-
android-m2repository
  - echo y | ${ANDROID_HOME}/tools/android --silent update sdk --no-ui --all --filter extra-
google-google_play_services
  - echo y | ${ANDROID_HOME}/tools/android --silent update sdk --no-ui --all --filter extra-
google-m2repository
  - chmod +x gradlew
build:
  script:
    - ./gradlew assembleDebug
  artifacts:
    paths:
    - app/build/outputs/apk/app-debug.apk
```

Change the build tools number to your compile target, and fork the docker image if you don't want to install everything every time.

Read Android CI Configuration online: https://riptutorial.com/gitlab/topic/6952/android-ci-configuration

Chapter 3: Continuous integration

Introduction

The GitLab CI runs build jobs based on a checked in `.gitlab-ci.yml`. Jobs are run on a remote server in it's own docker container.

The CI server itself is configured with a `config.toml`.

Remarks

- A build will fail if any lines in a job return an exit code != 0.

Examples

Runner installation

Debian, Ubuntu and CentOS

1. Add the official repository

Debian/Ubuntu

```
curl -L https://packages.gitlab.com/install/repositories/runner/gitlab-ci-multi-
runner/script.deb.sh | sudo bash
```

CentOS

```
curl -L https://packages.gitlab.com/install/repositories/runner/gitlab-ci-multi-
runner/script.rpm.sh | sudo bash
```

2. Install the `gitlab-ci-multi-runner` package

Debian/Ubuntu

```
sudo apt-get install gitlab-ci-multi-runner
```

CentOS

```
sudo yum install gitlab-ci-multi-runner
```

3. Register the runner

```
sudo gitlab-ci-multi-runner register
```

- Enter the URL to your GitLab CI. It should look like this `http://example.com/ci`
- Enter the registration token. If this is a project specfic runner you can find the token in `Project settings -> Runners`. If it is a shared runner go to `Admin area -> Runners` and find the registration token there.
- Now give your runner a descriptive name.
- Select the executor which you want to use. Valid executors are: `shell` (These can be later configured to use sh or bash),`docker`,`docker-ssh`,`ssh`,`parallels`,`virtualbox`,`docker+machine` or `docker-ssh+machine`. For more detail information on executors check the official documentation.

Windows

1. Download the runner binary and place it somewhere appropriate on your system.
2. Open a command prompt as Administrator
3. Register the runner

```
<runner-binary> register
```

- Enter the URL to your GitLab CI. It should look like this `http://example.com/ci`
- Enter the registration token. If this is a project specfic runner you can find the token in `Project settings -> Runners`. If it is a shared runner go to `Admin area -> Runners` and find the registration token there.
- Now give your runner a descriptive name.
- Select the executor which you want to use. Valid executors are: `shell`(Can be later configured to use cmd or powershell),`ssh`,`parallels` or `virtualbox`. For more detail information on executors check the official documentation.

4. (Optional) Register runner as service

```
<runner-binary> install --user <username> --password <password>
```

5. Start the runner

```
<runner-binary> start
```

Runner configuration

The config location for your runner is:

Debian/Ubuntu/CentOS

`/etc/gitlab-runner/config.toml`

if run as root

`~/.gitlab-runner/config.toml` if run as non-root

Windows

`config.toml` where your binary is located

A minimal `config.toml` can look like this:

```
concurrent = 1
[[runners]]
  name = "ExampleRunner"
  url = "https://example.com/ci"
  token = "f3058595ca4b2d217726466b1feed9"
  executor = "shell"
  shell = "bash"
```

For advanced configuration please check the official documentation.

Setup Gitlab CI to allow cloning other private repositories

Some projects like GoLang might need to clone other dependent GitLab repositories during build. To get this working you can add a Deploy Key to dependent repositories and put the private key (without password) into the origin repository.

Create and check-in a SSH key inside the Git Repository that depends on some other repository during build time:

```
ssh-keygen -t rsa -b 4096 -C "My CI Deploykey"

# In the following promt name the key "deploykey" and leave the passphrase empty
Generating public/private rsa key pair.
Enter file in which to save the key (/home/user/.ssh/id_rsa): deploykey
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in deploykey.
Your public key has been saved in deploykey.pub.

# check-in both files

```

Use the `deploykey.pub` to configure a deploykey in the dependent repository. You can find an Deploykey page in the GitLab Project Settings.

Now add the following to you `.gitlab-ci.yml`

```
before_script:
  # Git and SSH setup to clone private repos
  # Needs the deploykey file to be installed in all dependent repositories
  - git config --global url."git@gitlab.com:".insteadOf "https://gitlab.com/"
  # Add gitlab to known_hosts
  - mkdir -p ~/.ssh && chmod 700 ~/.ssh
  - ssh-keyscan -H gitlab.com >> ~/.ssh/known_hosts
  # Start the ssh agent and add the deploykey
  - chmod 400 deploykey
  - eval $(ssh-agent -s)
  - ssh-add deploykey
```

Now any call to `git clone` inside your build should work. Even if it's via some other tools like `go get`, `govendor sync`, or whatever you are using.

Read Continuous integration online: https://riptutorial.com/gitlab/topic/6258/continuous-integration

Chapter 4: Google Cloud SDK CI Configuration

Examples

Open JDK 8 Docker Example

```
  image: openjdk:8-jdk

  before_script:
       - curl https://dl.google.com/dl/cloudsdk/release/google-cloud-sdk.tar.gz > /tmp/google-
cloud-sdk.tar.gz
       - mkdir -p /usr/local/gcloud
       - tar -C /usr/local/gcloud -xvf /tmp/google-cloud-sdk.tar.gz
       - echo y |/usr/local/gcloud/google-cloud-sdk/install.sh
       - chmod +x ./gradlew

    build:
      script:
        - ./gradlew build
      artifacts:
        paths:
        - app/build/outputs/
```

Read Google Cloud SDK CI Configuration online: https://riptutorial.com/gitlab/topic/9094/google-cloud-sdk-ci-configuration

Credits

S. No	Chapters	Contributors
1	Getting started with gitlab	Alessandro Trinca Tornidor, Community, Connor Shea, Fairy, Greg Dubicki, jim
2	Android CI Configuration	ReverseCold
3	Continuous integration	Fairy, Tarion
4	Google Cloud SDK CI Configuration	Michael Meyer