Kostenloses eBook

LERNEN gitlab

Free unaffiliated eBook created from
Stack Overflow contributors.

#gitlab

Inhaltsverzeichnis

Über

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: gitlab

It is an unofficial and free gitlab ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official gitlab.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Kapitel 1: Erste Schritte mit Gitlab

Bemerkungen

GitLab ist eine webbasierte Versionskontrollsoftware, die auf git basiert und um zusätzliche Funktionen wie Fehlerverfolgung der Zweigstellenverwaltung und kontinuierliche Integration ergänzt. Es wird in Ruby entwickelt.

GitLab Community Edition (CE) wird Open Source entwickelt und verwendet die MIT-License. Seit August 2013 bietet GitLab Inc. die *Enterprise Edition (Enterprise Edition, EE) für Unternehmen an,* die mehr Funktionen bietet als die Community Edition.

GitLab.com ist die Software as a Service-Alternative und läuft auf einer Enterprise Edition. Es ist kostenlos und unterstützt private und öffentliche Endlager.

Versionen

Ausführung	Veröffentlichungsdatum
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

Ausführung	Veröffentlichungsdatum
8,0	2015-09-22

Examples

Installation oder Setup

Dies ist eine kurze Zusammenfassung des GitLab-Handbuchs zum Installieren eines GitLab CE Omnibus-Pakets .

Bedarf

Um die GitLab Community Edition auf Ihrem Server zu installieren, sollten Sie die Anforderungsseite lesen. Um es kurz zu machen, die empfohlenen Anforderungen sind:

- **OS:** Ubuntu, Debian, CentOS, RHEL
- **Ruby-Version:** Ruby (MRI) 2.1.x funktioniert derzeit nicht mit den Versionen 2.2 oder 2.3.
- **CPU:** 2 Kerne (unterstützt bis zu 500 Benutzer)
- **Speicher:** 2 GB (unterstützt bis zu 100 Benutzer)
- **Datenbank:** PostgreSQL

Installation

Die empfohlene Methode ist die Installation des Omnibus-Pakets, das schnell installiert werden kann. Es enthält GitLab und alle seine Abhängigkeiten (Ruby, PostgreSQL, Redis, Nginx, Unicorn usw.). Für andere Methoden schauen Sie sich die Installationsoptionen von GitLab an

Mit Ubuntu 16.04 als empfohlenem Betriebssystem beschreibt dieses Handbuch die Installationsschritte auf Debian-basierten Distributionen. Informationen zu CentOS, RHEL, Oracle Linux und Scientific Linux finden Sie in den Originalhandbüchern:

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

Ubuntu, Debian, Raspberrian

Installieren Sie die notwendigen Abhängigkeiten. Wenn Sie Postfix verwenden, wählen Sie während des Setups 'Internet Site'

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

Fügen Sie den Gitlab Package Server hinzu und installieren Sie das Paket

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

Wenn Sie das Repository nicht über ein Pipe-Skript installieren möchten, laden Sie das Paket manuell herunter und installieren Sie es mithilfe von

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

Nun konfigurieren und starten Sie GitLab

```
sudo gitlab-ctl reconfigure
```

Navigieren Sie schließlich zum Hostnamen und melden Sie sich an. Zunächst werden Sie umgeleitet, um ein Kennwort für das ursprüngliche Administratorkonto anzugeben. Danach können Sie sich anmelden. Der **Standard-Benutzername des Administratorkontos** ist **root** .

Erste Schritte mit Gitlab online lesen: https://riptutorial.com/de/gitlab/topic/2046/erste-schritte-mit-gitlab

Kapitel 2: Android CI-Konfiguration

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
```

Ändern Sie die Build-Tools-Nummer in Ihr Kompilierungsziel und verzweigen Sie das Docker-Image, wenn Sie nicht jedes Mal alles installieren möchten.

Android CI-Konfiguration online lesen: https://riptutorial.com/de/gitlab/topic/6952/android-ci-konfiguration

Kapitel 3: Google Cloud SDK-CI-Konfiguration

Examples

Öffnen Sie das JDK 8 Docker-Beispiel

```
  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/
```

Google Cloud SDK-CI-Konfiguration online lesen:
https://riptutorial.com/de/gitlab/topic/9094/google-cloud-sdk-ci-konfiguration

Kapitel 4: Kontinuierliche Integration

Einführung

Das GitLab-CI führt Build-Jobs basierend auf einem eingecheckten `.gitlab-ci.yml` . Jobs werden auf einem Remote-Server in einem eigenen Docker-Container ausgeführt.

Der CI-Server selbst ist mit einer `config.toml` konfiguriert.

Bemerkungen

- Ein Build schlägt fehl, wenn eine Zeile in einem Job einen Exit-Code zurückgibt! = 0.

Examples

Runner-Installation

Debian, Ubuntu und CentOS

1. Fügen Sie das offizielle Repository hinzu

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. Installieren Sie das `gitlab-ci-multi-runner`

Debian / Ubuntu

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

CentOS

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

3. Registrieren Sie den Läufer

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

- Geben Sie die URL zu Ihrer GitLab-CI ein. Es sollte so aussehen `http://example.com/ci`
- Geben Sie das Registrierungstoken ein. Wenn dies ein projektspezifischer Läufer ist, finden Sie das Token unter `Project settings -> Runners` . Wenn es sich um einen gemeinsam genutzten Läufer handelt, gehen Sie zum `Admin area -> Runners` und suchen Sie dort das Registrierungstoken.
- Nun gib deinem Läufer einen beschreibenden Namen.
- Wählen Sie den Executor aus, den Sie verwenden möchten. Gültige Vollstrecker sind: `shell` (Diese können später SH oder bash verwenden konfiguriert werden), `docker` , `docker-ssh` , `ssh` , `parallels` , `virtualbox` , `docker+machine` oder `docker-ssh+machine` . Weitere Informationen zu Executoren finden Sie in der offiziellen Dokumentation .

Windows

1. Laden Sie die Runner-Binärdatei herunter und platzieren Sie sie an einem geeigneten Ort in Ihrem System.
2. Öffnen Sie eine Eingabeaufforderung als Administrator
3. Registrieren Sie den Läufer

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

- Geben Sie die URL zu Ihrer GitLab-CI ein. Es sollte so aussehen `http://example.com/ci`
- Geben Sie das Registrierungstoken ein. Wenn dies ein projektspezifischer Läufer ist, finden Sie das Token unter `Project settings -> Runners` . Wenn es sich um einen gemeinsam genutzten Läufer handelt, gehen Sie zum `Admin area -> Runners` und suchen Sie dort das Registrierungstoken.
- Nun gib deinem Läufer einen beschreibenden Namen.
- Wählen Sie den Executor aus, den Sie verwenden möchten. Gültige Executoren sind: `shell` (Kann später für die Verwendung von cmd oder powershell konfiguriert werden), `ssh` , `parallels` oder `virtualbox` . Weitere Informationen zu Executoren finden Sie in der offiziellen Dokumentation .

4. (Optional) Registrieren Sie den Läufer als Dienst

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

5. Starten Sie den Läufer

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

Runner-Konfiguration

Der Konfigurationsort für Ihren Läufer lautet:

Debian / Ubuntu / CentOS

`/etc/gitlab-runner/config.toml` wenn als root ausgeführt

`~/.gitlab-runner/config.toml` falls nicht als root ausgeführt

Windows

`config.toml` wo sich Ihre Binärdatei befindet

Ein minimales `config.toml` kann so aussehen:

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

Für die erweiterte Konfiguration lesen Sie bitte die offizielle Dokumentation .

Richten Sie das Gitlab-CI ein, um das Klonen anderer privater Repositorys zu ermöglichen

Einige Projekte wie GoLang müssen möglicherweise während des Builds andere abhängige GitLab-Repositorys klonen. Damit dies funktioniert, können Sie einen Deploy-Schlüssel zu abhängigen Repositorys hinzufügen und den privaten Schlüssel (ohne Kennwort) in das Ursprungs-Repository einfügen.

Erstellen und checken Sie einen SSH-Schlüssel im Git-Repository ein, der während der Erstellungszeit von einem anderen Repository abhängig ist:

```
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
```

Verwenden Sie " `deploykey.pub` , um einen deploykey im abhängigen Repository zu konfigurieren. Sie finden eine Deploykey-Seite in den GitLab-Projekteinstellungen.

`.gitlab-ci.yml` Sie `.gitlab-ci.yml` Folgendes hinzu: `.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
```

Nun sollte jeder Aufruf zum `git clone` in Ihrem Build funktionieren. Auch wenn es über andere Tools wie `go get` , `govendor sync` oder was auch immer Sie verwenden, `govendor sync` .

Kontinuierliche Integration online lesen: https://riptutorial.com/de/gitlab/topic/6258/kontinuierliche-integration

Credits

S. No	Kapitel	Contributors
1	Erste Schritte mit Gitlab	Alessandro Trinca Tornidor, Community, Connor Shea, Fairy, Greg Dubicki, jim
2	Android CI-Konfiguration	ReverseCold
3	Google Cloud SDK-CI-Konfiguration	Michael Meyer
4	Kontinuierliche Integration	Fairy, Tarion