

[Self Hosted Git]

Alberto Donati

`alberto.donati6@studio.unibo.it`

Alessia Rocco

`alessia.rocco@studio.unibo.it`

19 luglio 2021

Il progetto ha come obiettivo quello di studiare le possibili realizzazioni per una implementazione self-hosted che gestisca i repository Git di studenti e ricercatori dell'università. L'idea generale è quella di ospitare nei server universitari i progetti interni, attraverso la soluzione ritenuta più appropriata tra le varie proposte durante questa trattazione.

Indice

1	Introduzione	4
1.1	Git	4
1.2	Panoramica soluzioni proposte	5
2	GitLab	6
2.1	Funzioni	6
2.2	Soluzioni implementative	8
2.2.1	GitLab Docker hosted	8
2.2.2	GitLab Azure hosted	19
2.3	Runner	21
2.3.1	Runner Linux	21
2.3.2	Runner Docker	23
3	Gogs	26
3.1	Funzioni	26
3.2	Installazione tramite Ubuntu	28
4	Gitea	31
4.1	Funzioni	31
4.2	Installazione tramite Docker	33
5	GitBucket	35
5.1	Funzioni	35
5.2	Installazione tramite Java	37
6	GitPrep	42
6.1	Funzioni	42
6.2	Installazione tramite Ubuntu	44
7	Kallithea	48
7.1	Funzioni	48
7.2	Installazione tramite Ubuntu	50
8	Tuleap	55
8.1	Funzioni	55
8.2	Installazione tramite Docker	56
9	Phabricator	59
9.1	Funzioni	59
9.2	Installazione (non testata)	60
10	Confronto	61
10.1	Confronto funzionalità	61

10.2	Confronto installazione	61
10.3	Confronto sicurezza	62
10.4	Soluzione scelta	62
11	Prova di deploy	63
11.1	Configurazione	63
11.2	Configurazione HTTPS	66
12	Conclusioni	67
12.1	Lavori futuri	67
12.2	Cosa abbiamo imparato	67
	Glossario	68

1 Introduzione

1.1 Git

Git è un sistema per il versionamento distribuito del codice gratuito e open source che consente di gestire e tracciare piccoli e grandi progetti. Esso è nato nel 2005 per sostituire BitKeeper, sviluppato inizialmente da Linus Torvalds e orientato verso sistema Unix, ancora adesso infatti il tracciamento dei file e i relativi permessi vengono gestiti Unix-oriented.

Git offre molte funzionalità per la gestione dei progetti tracciati in repository, come:

- *Repository*: una collezione di metadati contenenti la storia di un progetto;
- *Branching*: la possibilità di avere multipli branch locali indipendenti su cui lavorare in remoto o in modo condiviso;
- *Backup multipli*: ogni utente anche se lavora in locale ha una copia backup integrale del main server. Ognuna di queste copie possono essere pushate o sostituite in caso di eventi di crash o corruzioni;
- *Staging Area*: è l'area intermedia che consente di personalizzare, riguardare e formattare i commit prima di essere eseguiti. La staging area infatti consente di aggiungere o togliere file modificati dai commit, che considerano solo i file aggiunti alla staging area;
- *Data Assurance*: ogni commit di un progetto ha un ID unico che è impossibile da modificare, questo assicura l'integrità dei dati e dei progetti dato che ogni modifica è rintracciabile da un nuovo commit.

Di seguito riportiamo una panoramica dei comandi base per utilizzare Git:

- `git init`: inizializzazione di un repository;
- `git clone`: crea una copia locale di un repository remoto;
- `git add`: aggiunge uno o più file alla staging area;
- `git reset`: rimuove uno o più file dalla staging area;
- `git commit`: permette di tracciare le modifiche locali;
- `git push`: invia i commit al repository remoto;
- `git pull`: scarica l'ultima versione del repository remoto;
- `git branch`: controlla la creazione, visualizzazione e eliminazione dei branch;
- `git checkout`: permette di spostarsi tra i branch o di crearne di nuovi;
- `git merge`: incorpora i cambiamenti dal commit citato nel branch corrente.

1.2 Panoramica soluzioni proposte

Molte sono le piattaforme per la gestione di repository Git che permettono una soluzione self-hosted come per esempio GitLab, Gogs, Gitea, Phabricator, GitBucket, Tuleap, GitPrep e Kallithea. Esse offrono tutte di base la compatibilità con le funzionalità predefinite di Git e poi ulteriori features e grafiche a seconda della soluzione considerata, tutte quante analizzate in dettaglio nel seguito.[17][2]

I principali ambienti di esecuzione per l'utilizzo self-hosted delle varie piattaforme sono Docker e Linux.

E' stata quindi proposta una tabella che riassume tutte le principali feature riscontrate dallo studio di ogni soluzione git self-hosted. Come si può notare successivamente, l'implementazione più completa è GitLab ma sono comunque valide anche le altre proposte, come ad esempio Gogs e GitBucket, per implementazioni più leggere in termini di risorse, adatte ad usi privati e meno intensivi.

	GitLab	Gogs	Gitea	GitBucket	GitPrep	Kallithea
<i>migrazione</i>	✓ (avanzato)	✓	✓	✓	✗	✓
<i>issues</i>	✓	✓	✓	✓	✓	✗
<i>ToDo list</i>	✓	✗	✗	✗	✗	✗
<i>Dev-Ops PipeLine</i>	✓	✗	✗	✗	✗	✗
<i>gestione gruppi</i>	✓	✓ (organization)	✓ (organization)	✓	✗	✓
<i>gestione utenti da admin</i>	✓	✓	✓	✓	✓	✓
<i>gestione repository</i>	✓	✓	✓	✓	✓	✓
<i>branches</i>	✓	✓	✓	✓	✓	✓
<i>fork</i>	✓	✓	✓	✓	✓	✓
<i>web Integrated Development Environment</i>	✓	✓	✓	✓	✗	✓
<i>built-in container register</i>	✓	✗	✗	✗	✗	✗
<i>activity</i>	✓	✓ (dashboard)	✓ (dashboard)	✓ (news feed)	✗	✓ (journal)
<i>milestones</i>	✓	✓	✓	✓	✗	✗
<i>controllo avanzato degli accessi</i>	✓	✓	✓	✓	✗	✓
<i>webhook</i>	✓	✓	✓	✓	✗	✗
<i>wiki</i>	✓	✓	✓	✓	✓	✗
<i>snippet</i>	✓	✗	✗	✓	✗	✓ (gists)
<i>deploy keys</i>	✓	✓	✓	✓	✗	✗
<i>aggiunta di chiavi crittografiche</i>	✓	✓	✓	✓	✓	✗
<i>supporto multilingua</i>	✓	✓	✓	✗	✗	✗
<i>autenticazione esterna</i>	✓	✓	✓	✓	✗	✗
<i>feed</i>	✓	✗	✗	✓	✗	✓
<i>barra di ricerca</i>	✓	✓ (explore)	✓ (explore)	✓	✓	✓
<i>ricerca avanzata</i>	✓	✗	✗	✗	✗	✓
<i>pull requests</i>	✓ (merge requests)	✓	✓	✓	✓	✓
<i>2FA</i>	✓	✗	✓	✗	✗	✗

Tabella 1: Tabella panoramica funzionalità

2 GitLab

GitLab[4] è un'ottima alternativa in quanto è open source, attualmente molto utilizzata e ben documentata da molti forum¹. Integra oltre alle normali funzioni Git anche moltissimi strumenti DevOps.

Di GitLab esiste la versione cloud, la versione self-hosted e la Development-Kit² dedicata ai programmatori che vogliono contribuire alla manutenzione di GitLab. A noi interessa particolarmente la versione self-hosted che permette ad un utente di usare GitLab su un proprio server privato.

Inoltre, sia per la versione cloud che per quanto riguarda la versione self-hosted, esistono 3 tipi di licenze:

- *Free*: ha features illimitate, gitlab runner, free static website e 400 minuti al mese di CI/CD;
- *Premium*: ha tutto della licenza free con in aggiunta della revisione del codice, management dei progetti, supporto tecnico 24/7 e 10000 minuti al mese di CI/CD;
- *Ultimate*: ha tutto della licenza premium con in più sicurezza di test avanzata, utenti guest gratuiti e 50000 minuti al mese di CI/CD.

Considerando che l'ambito di utilizzo è universitario, dovrebbero esistere delle licenze gratuite illimitate. Come viene infatti specificato dal sito ufficiale³ emerge che:

"Beyond providing free, unlimited licenses of our top-tier functionality (SaaS or self-managed), we want to be a resource for you to learn more about DevOps and connect with others who share your interest."

In qualsiasi caso, la versione free per eseguire dei test è più che sufficiente e ricca di funzionalità. Una versione self-hosted community edition non cambia rispetto ad una versione enterprise con licenza non attivata.

2.1 Funzioni

Tutte le funzionalità di GitLab vengono specificate nella documentazione ufficiale[7] e di seguito sono riportate solo le più rilevanti per questo studio, come da tabella 1.

¹<https://forum.gitlab.com/>

²<https://gitlab.com/gitlab-org/gitlab-development-kit>

³<https://about.gitlab.com/solutions/education/>

	Gitlab
<i>migrazione</i>	✓ (avanzato)
<i>issues</i>	✓
<i>ToDo list</i>	✓
<i>Dev-Ops PipeLine</i>	✓
<i>gestione gruppi</i>	✓
<i>gestione utenti da admin</i>	✓
<i>gestione repository</i>	✓
<i>branches</i>	✓
<i>fork</i>	✓
<i>web Integrated Development Environment</i>	✓
<i>built-in container register</i>	✓
<i>activity</i>	✓
<i>milestones</i>	✓
<i>controllo avanzato degli accessi</i>	✓
<i>webhook</i>	✓
<i>wiki</i>	✓
<i>snippet</i>	✓
<i>deploy keys</i>	✓
<i>aggiunta di chiavi crittografiche</i>	✓
<i>supporto multilingua</i>	✓
<i>autenticazione esterna</i>	✓
<i>feed</i>	✓
<i>barra di ricerca</i>	✓
<i>ricerca avanzata</i>	✓
<i>pull requests</i>	✓ (merge requests)
<i>2FA</i>	✓

La principale funzionalità che ha GitLab a differenza di tutte le altre soluzioni studiate sono le Dev-Ops Pipeline: un set di istruzioni che consentono di determinare delle esecuzioni automatiche, eseguite nei Runner, che ad esempio possono scattare al momento della push di un repository. Questo permette quindi di aggiungere una serie di controlli al codice di un repo legati alla sua esecuzione. Si tratta perciò di un aspetto essenziale nel caso in cui l'utilizzo di GitLab self-hosted sia legato non solo al tracciamento o scambio di progetti universitari ma anche alla costruzione e deploy di applicazioni. GitLab ha inoltre integrato un registro dei container, per salvare tutte le immagini Docker legate ad un progetto; funzionalità di migrazione dei repository, non solo da una singolo progetto ma anche da tutti i progetti di un utente e un avanzato controllo degli accessi, attraverso il filtraggio dei domini e vari user-authentication provider esterni.

2.2 Soluzioni implementative

L'implementazione di GitLab in versione self-hosted è stata testata sia in ambiente Azure che con Docker, abbiamo redatto una semplice e sintetica guida per replicare le installazioni sulla falsa riga di quelle proposte da GitLab stesso[8].

Un'altra soluzione da noi non testata e non ufficiale basata su Docker, con una serie di migliorie e consigli sulla sua installazione e configurazione, è stata proposta dalla risorsa web Oramind[21].

2.2.1 GitLab Docker hosted

Per l'installazione di GitLab ospitato tramite *Docker* abbiamo creato una virtual machine con sistema operativo Lubuntu 20.04 su cui esegue *Docker*.

Installazione

Installazione di *Docker* tramite il comando:

```
1 sudo apt-get install docker.io
```

Successivamente, seguiamo le seguenti indicazioni prese dalla documentazione ufficiale di GitLab[6].

Come primo step scarichiamo l'immagine Docker di GitLab dalla fonte ufficiale (DockerHub) con il comando:

```
1 sudo docker pull gitlab/gitlab-ee:latest
```

Il sistema scaricherà e decomprimerà l'immagine Docker.

Successivamente settiamo la variabile globale di sistema di GitLab tramite:

```
1 export GITLAB_HOME=/srv/gitlab
```

Infine per avviare il container si lancia il seguente comando:

```
1 sudo docker run --detach \  
2 --hostname gitlab.example.com \  

```

```
3 --publish 443:443 --publish 80:80 --publish 22:22 \  
4 --name gitlab \  
5 --restart always \  
6 --volume $GITLAB_HOME/config:/etc/gitlab \  
7 --volume $GITLAB_HOME/logs:/var/log/gitlab \  
8 --volume $GITLAB_HOME/data:/var/opt/gitlab \  
9 gitlab/gitlab-ee:latest
```

Impostando come `hostname` il nome di dominio con cui potersi collegare e come `publish` la configurazione delle porte tra il contenitore Docker e l'esterno di esso.

Nel nostro caso decidiamo di modificare il comando sopra non assegnando `hostname`, dato che l'utilizzo per le prove è solo locale, e omettendo `-restart always` per evitare che il contenitore si riavvii ogni volta che il sistema riparte.

```
1 sudo docker run --detach \  
2 --publish 443:443 --publish 80:80 --publish 22:22 \  
3 --name gitlab \  
4 --volume $GITLAB_HOME/config:/etc/gitlab \  
5 --volume $GITLAB_HOME/logs:/var/log/gitlab \  
6 --volume $GITLAB_HOME/data:/var/opt/gitlab \  
7 gitlab/gitlab-ee:latest
```

Tramite il comando:

```
1 sudo docker logs -f gitlab
```

Ci si può agganciare al contenitore per visualizzare i `log` per capire come sta andando la situazione. Dai `log` si possono vedere il comportamento del contenitore e le varie chiamate effettuate al server GitLab.

Una volta che tutti i servizi sono partiti e caricati correttamente si vedrà la pagina iniziale di GitLab all'indirizzo web specificato in `hostname`, che in tal caso è `localhost` o `127.0.0.1`. Al primo accesso alla pagina, comparirà il login che chiede di resettare la password dell'utente **root** (cioè l'amministratore).

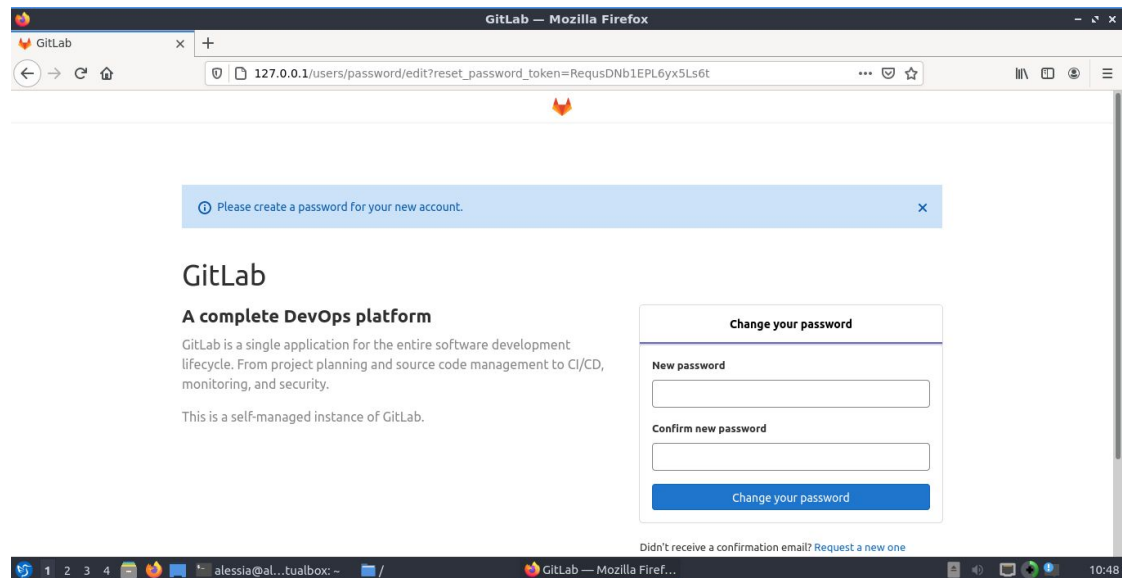


Figura 1: GitLab - Schermata iniziale di reimpostazione password root

Nel caso si voglia reindirizzare su altre porte si può ad esempio inserire alla creazione del container Docker:

```
1 --publish 1443:443 --publish 1080:80 --publish 1022:22
```

Al link http://127.0.0.1/help/instance_configuration si può vedere la nostra configurazione.

Infine, se si volesse aggiornare il sistema (come consigliato dall'immagine seguente), eseguire i seguenti comandi che fanno riferimento alle indicazioni fornite nella documentazione ufficiale⁴.

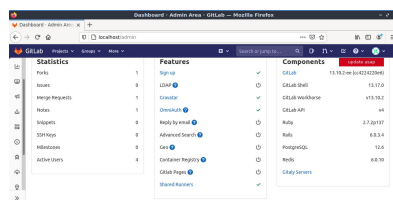


Figura 2: GitLab - Non aggiornato

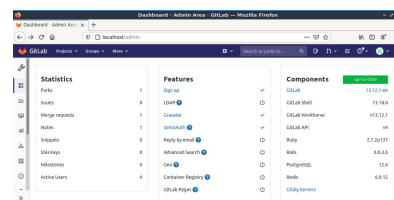


Figura 3: GitLab - Aggiornato

Per stoppare ed eliminare l'immagine Docker di GitLab:

```
1 sudo docker stop gitlab
2 sudo docker rm gitlab
```

⁴<https://docs.gitlab.com/omnibus/docker/README.html>

Rilanciare poi i comandi dell'installazione, presenti anche nel file bash fornito. Il sistema ora è aggiornato, come si può constatare dall'immagine soprastante.

Funzionalità

Si possono creare nuovi utenti o accedere tramite la pagina principale.

Entrando come **root** si hanno a disposizione diverse possibili opzioni, tutte mostrate nel menù presente a sinistra. Un elemento importante è la dashboard, da qui possiamo vedere le diverse componenti che costituiscono il sistema e se la versione di GitLab è aggiornata.

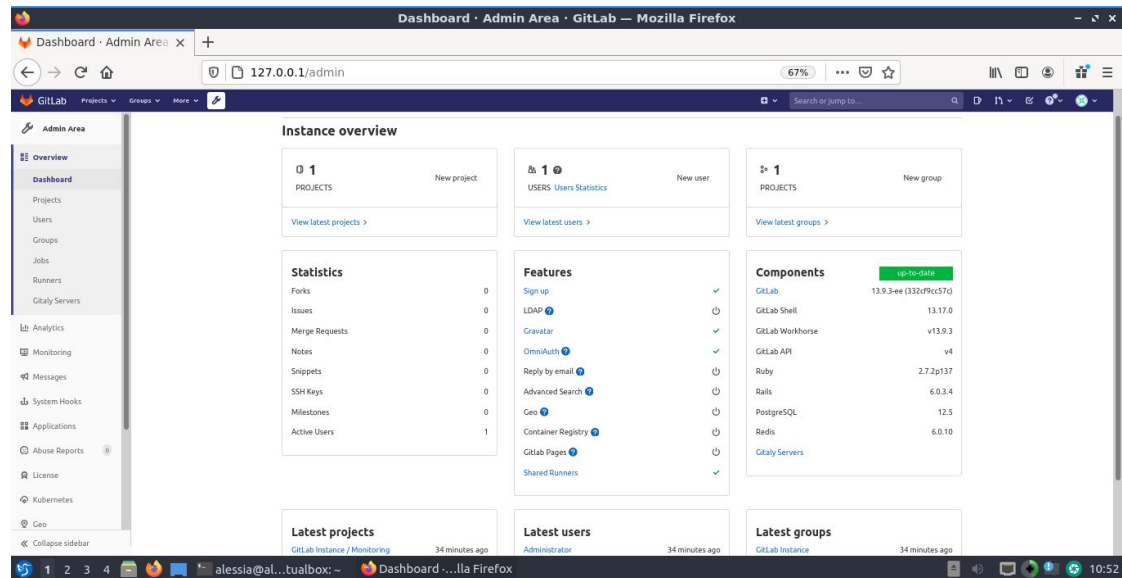


Figura 4: GitLab - Schermata di dashboard utente root

Altre funzioni sono Deploy Key (per concedere degli accessi tramite chiavi) o GitLab servers (per vedere la versione Git installata e a quale versione).

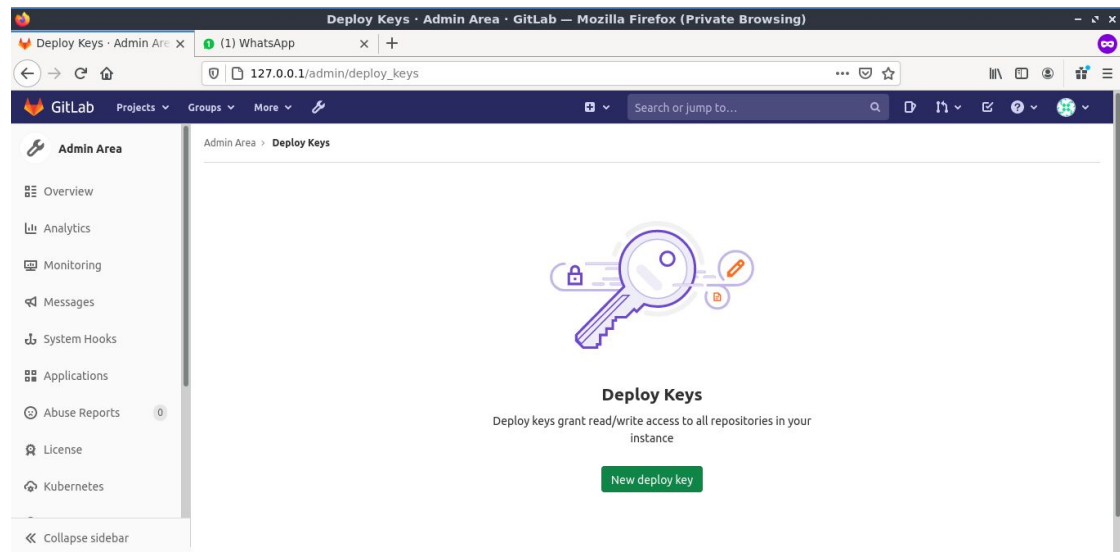


Figura 5: GitLab - Schermata deploy keys

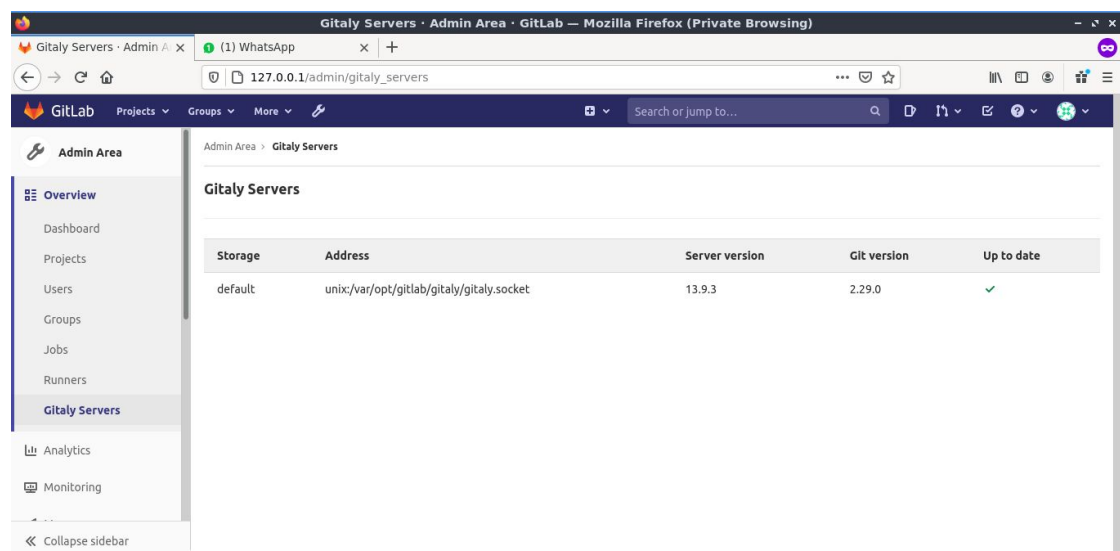


Figura 6: GitLab - Schermata Gitaly Servers

Sempre per quanto riguarda le funzioni di amministratore si ha la completa gestione degli utenti: si possono cancellare, bloccare, eliminare, ecc.

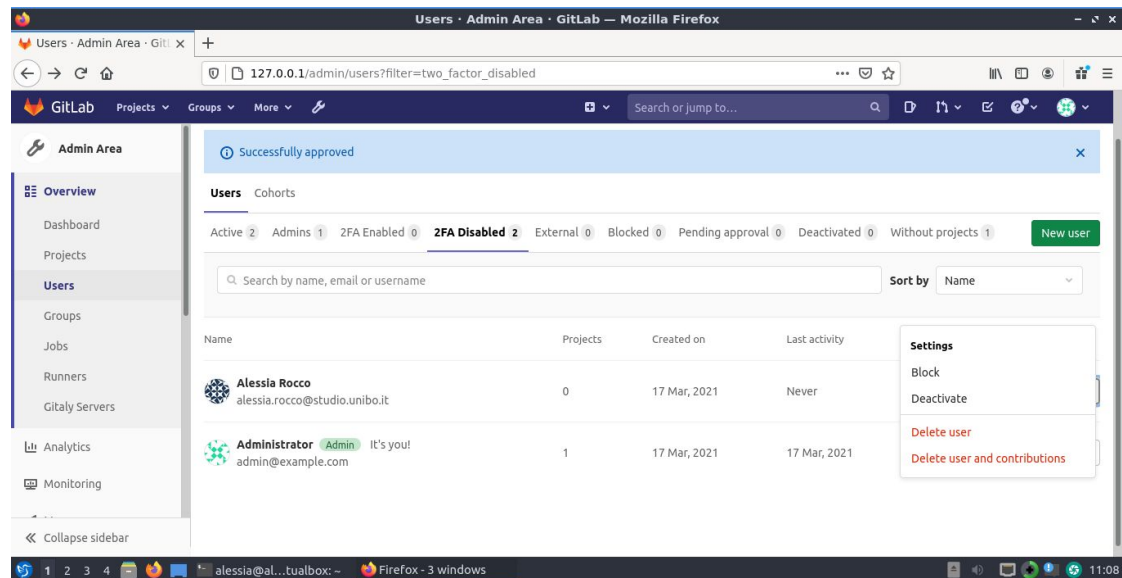


Figura 7: GitLab - Schermata gestione utenti da root

Si può inoltre decidere di creare nuovi gruppi, nuovi progetti, nuovi snippet ecc. Questa è una funzione sia degli utenti amministratori che di quelli senza privilegi o standard.

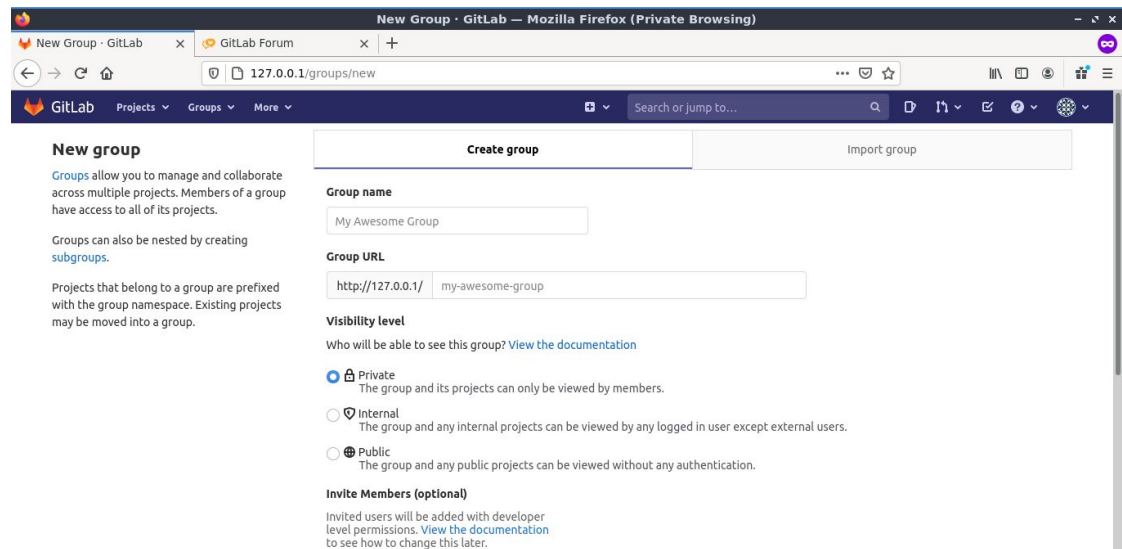


Figura 8: GitLab - Schermata creazione nuovo gruppo

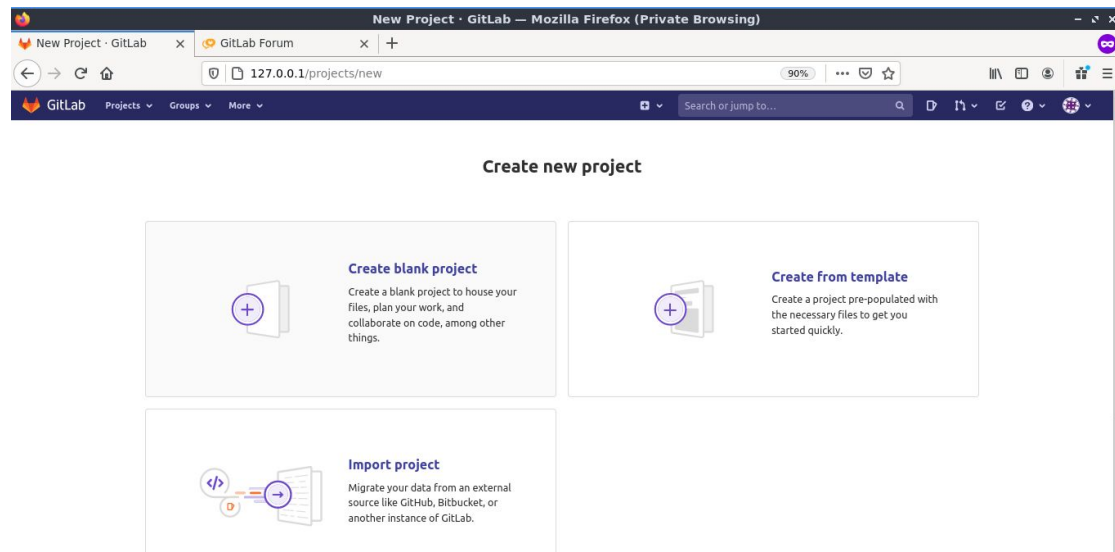


Figura 9: GitLab - Schermata creazione nuovo progetto

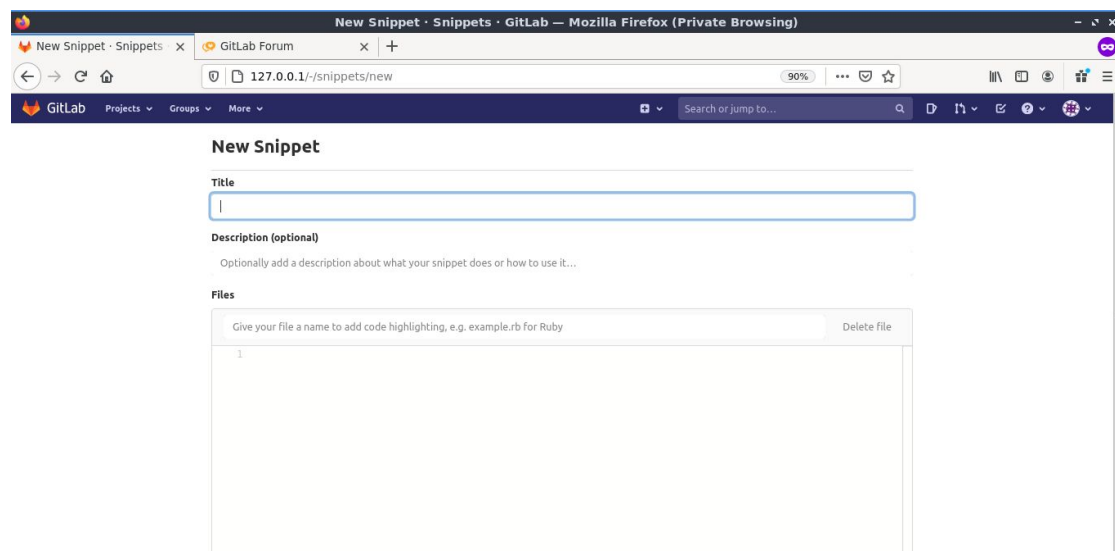


Figura 10: GitLab - Schermata creazione nuovo snippet

E' possibile aggiungere una licenza, nel nostro caso non abbiamo attivato nessuna prova gratuita ma essendo un progetto universitario dovrebbero esserci licenze illimitate gratuite (vedi paragrafo 2).

Inoltre, GitLab self-hosted permette di:

- personalizzare i *Termini di Servizio*;
- implementare un controllo degli accessi personalizzato;

- gestire la visibilità, gli accessi e le pull request dei progetti;
- gestire l'invio di mail in determinate situazioni;
- inviare comunicazioni agli utenti.

Molte delle configurazioni principali possono essere gestite direttamente dall'interfaccia Web dell'utente **admin** e sono tutte descritte in modo dettagliato dalla documentazione ufficiale[5].

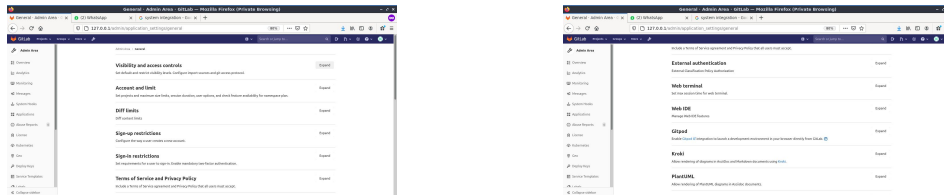


Figura 11: GitLab - Schermata settings admin

GitLab infine, permette di personalizzare la grafica del proprio ambiente, modificando ad esempio header, footer, favicon, testi di homepage ecc.

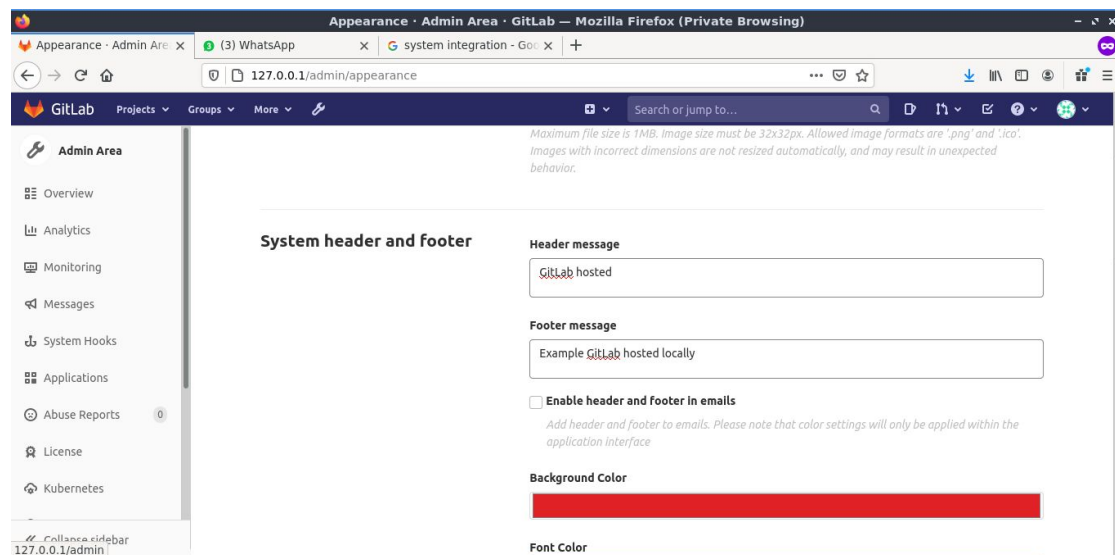


Figura 12: GitLab - Schermata modifica appearance

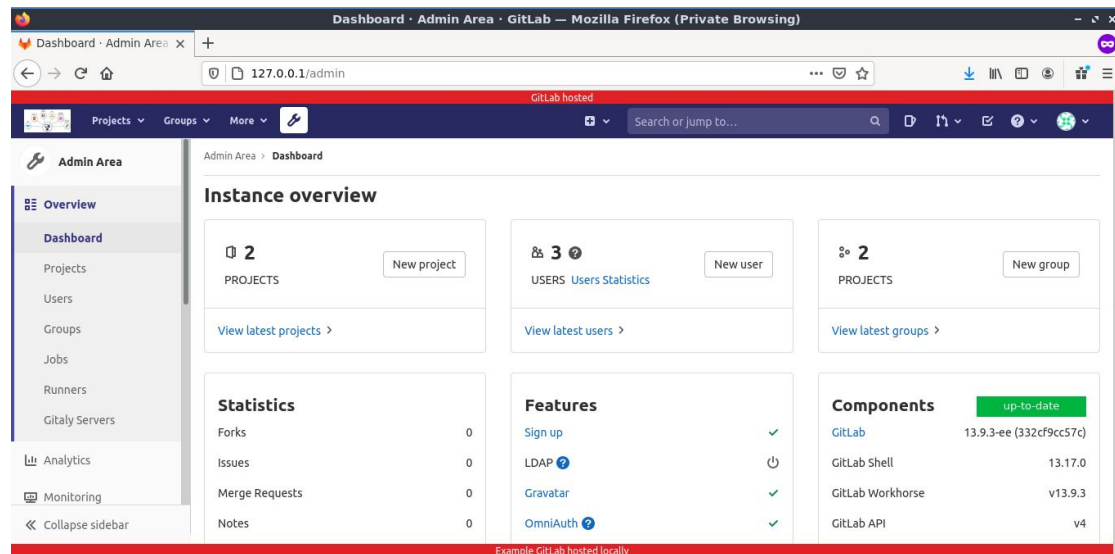


Figura 13: GitLab - Schermata appearance modificata

Test base funzionalità

Di seguito alcuni test di alcune funzionalità base testate da terminale: clone, add, commit, push, pull, merge, branch, checkout.

Abbiamo fatto inoltre dei test da browser che sono stati svolti attraverso la comunicazione con un server GitLab self-hosted creato in un container Docker.

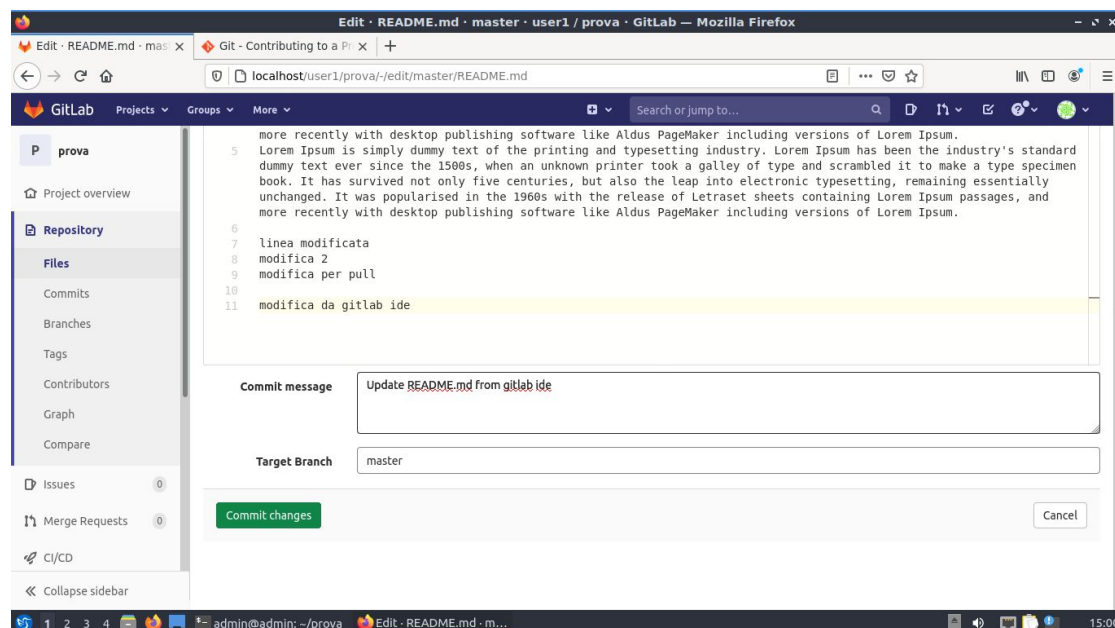


Figura 14: GitLab - commit

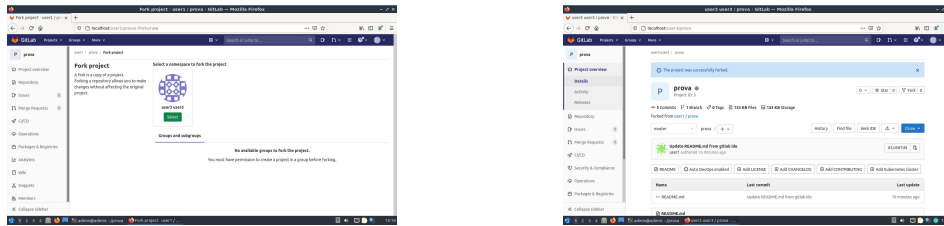


Figura 15: GitLab - fork

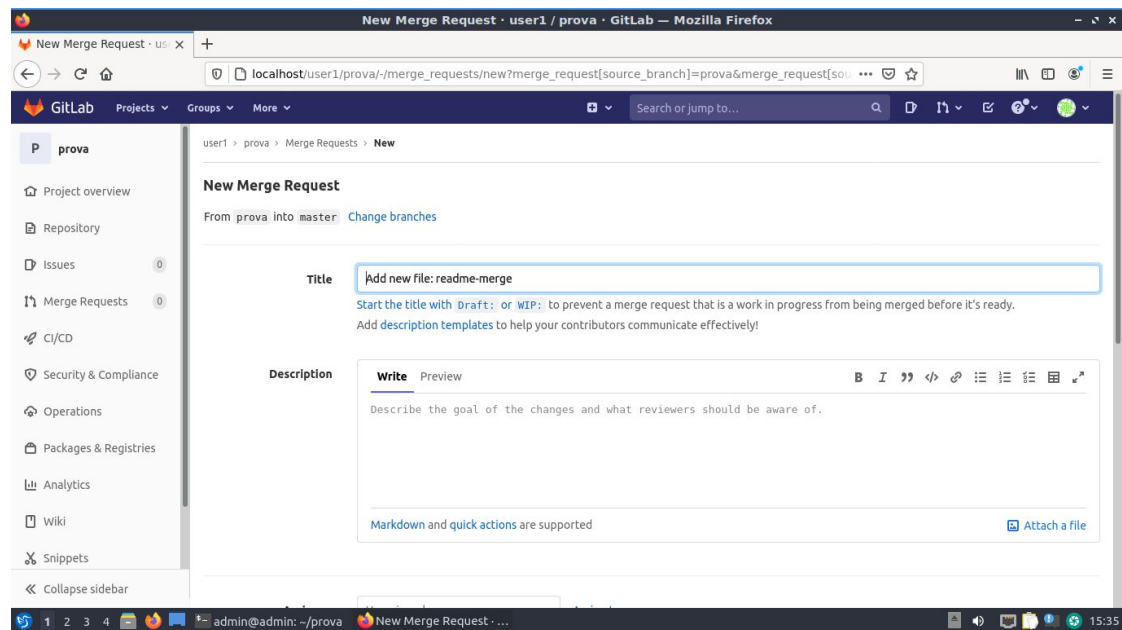


Figura 16: GitLab - merge request

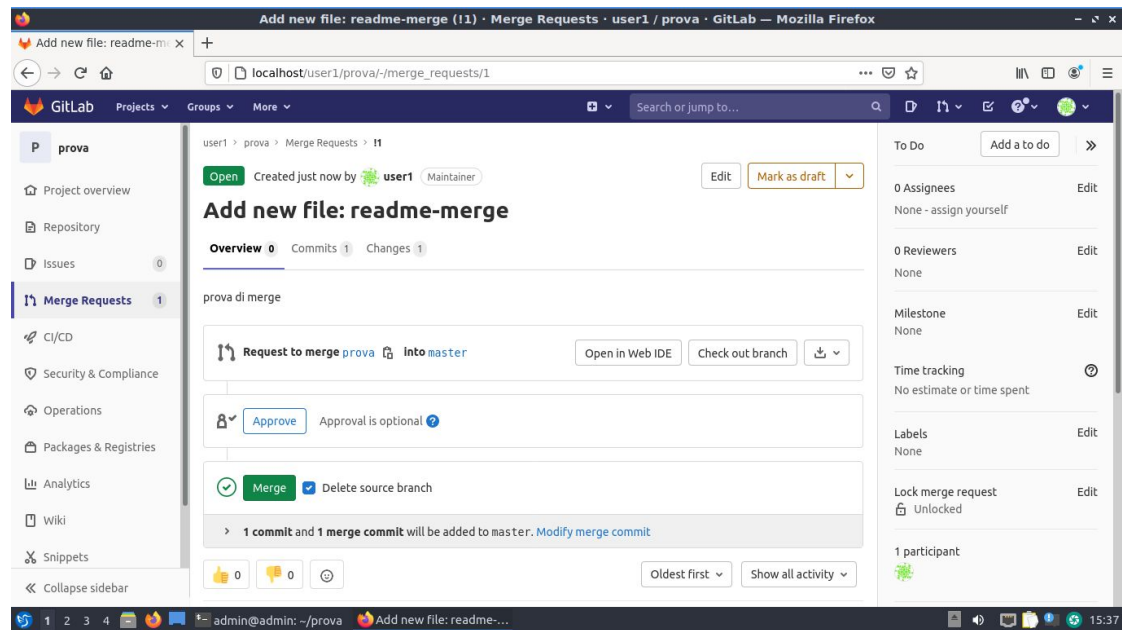


Figura 17: GitLab - merge esecuzione

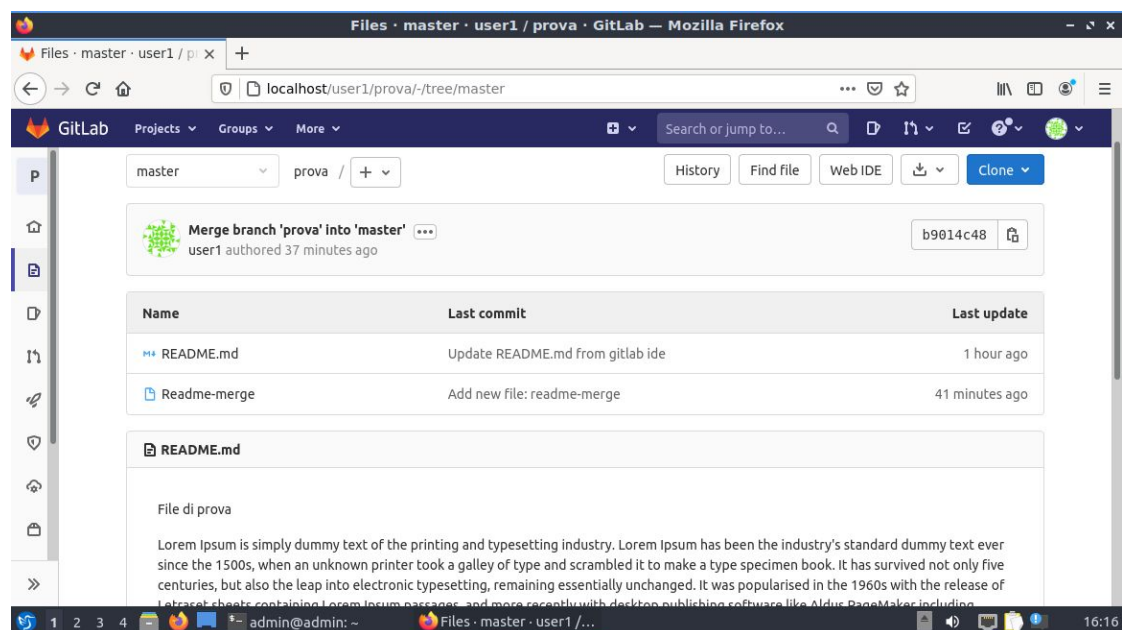


Figura 18: GitLab - merge request eseguita

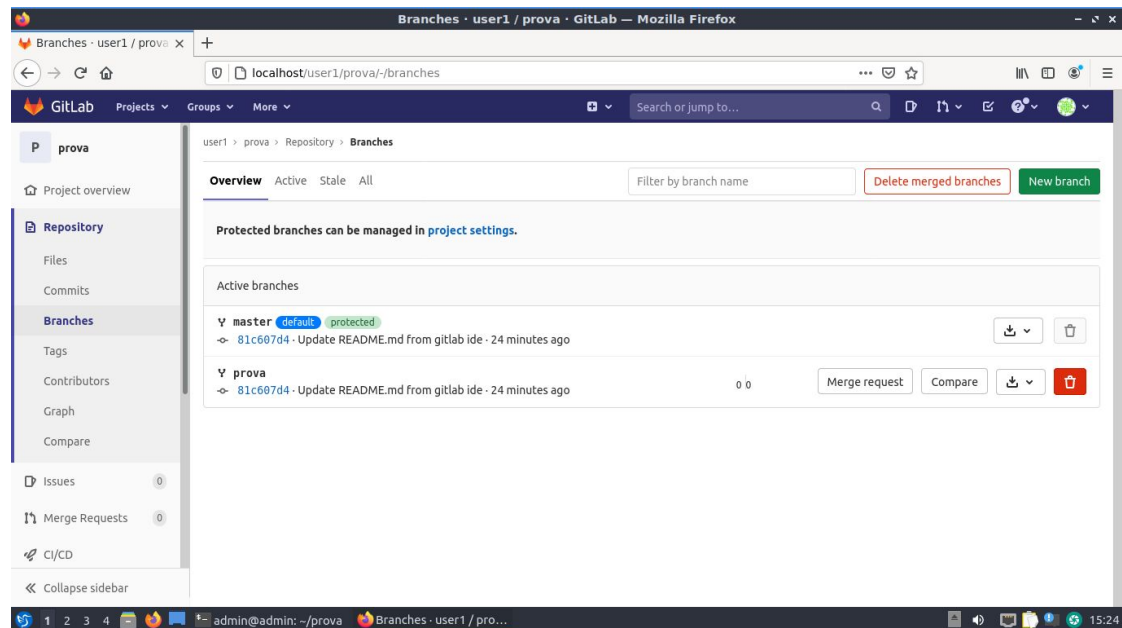


Figura 19: GitLab - branch

2.2.2 GitLab Azure hosted

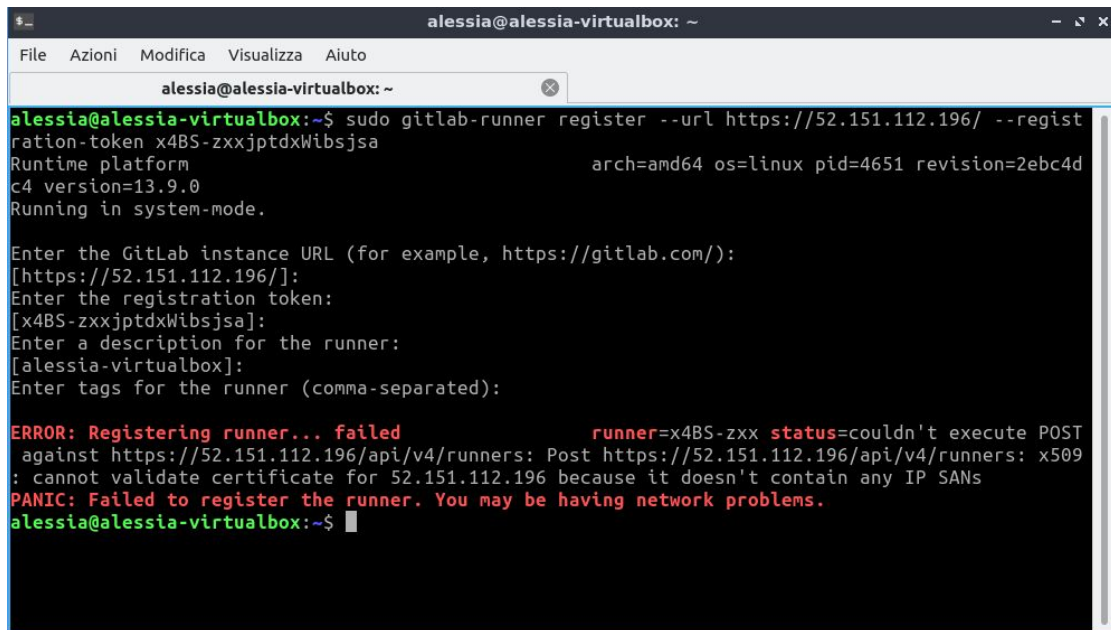
Per provare a testare su una macchina più performante, rispetto alle nostre disponibili, abbiamo creato una macchina virtuale con un'immagine presa dal Marketplace di Azure. Questa immagine è un'immagine *Bitnami* di Azure⁵.

Bitnami offre sia la virtualizzazione in ambienti proprietari come Amazon, Azure, etc..., così come su macchine virtuali su pc, cluster o server.

Primo accesso Nel caso di primo accesso, è presente un file con le credenziali per accedere all'interno della macchina GitLab di Bitnami. Questa informazione si ricava con le istruzioni presenti a questo link <https://bitnami.com/stack/gitlab/virtual-machine> → sezione "Credentials".

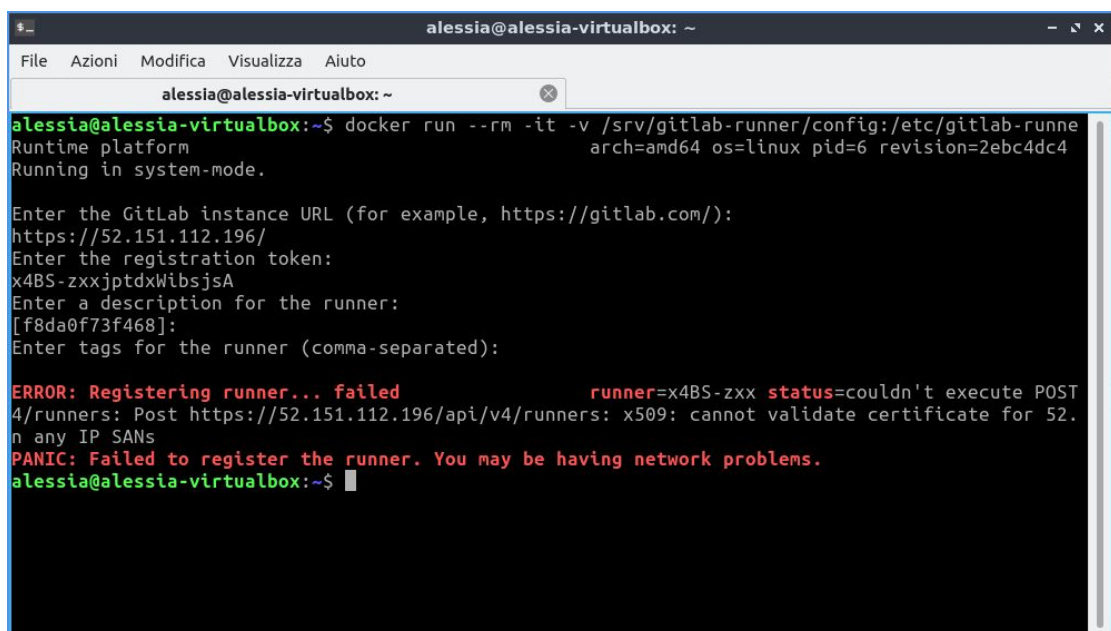
Problema Certificati SSL La prima problematica riscontrata è legata alla possibilità di creare pipeline e Runner alla condizione di avere dei certificati SSL. Provando a creare un Runner sia tramite Linux che tramite Docker, collegato in seguito alla macchina virtuale su Azure, il Runner viene creato con successo ma non si riesce a registrare a causa dell'assenza di certificazione per una connessione sicura.

⁵<https://bitnami.com/stack/gitlab>



```
alessia@alessia-virtualbox: ~  
File Azioni Modifica Visualizza Aiuto  
alessia@alessia-virtualbox: ~  
alessia@alessia-virtualbox:~$ sudo gitlab-runner register --url https://52.151.112.196/ --regist  
ration-token x4BS-zxxjptdxWibsjsa  
Runtime platform arch=amd64 os=linux pid=4651 revision=2ebc4d  
c4 version=13.9.0  
Running in system-mode.  
  
Enter the GitLab instance URL (for example, https://gitlab.com/):  
[https://52.151.112.196/]:  
Enter the registration token:  
[x4BS-zxxjptdxWibsjsa]:  
Enter a description for the runner:  
[alessia-virtualbox]:  
Enter tags for the runner (comma-separated):  
  
ERROR: Registering runner... failed runner=x4BS-zxx status=couldn't execute POST  
against https://52.151.112.196/api/v4/runners: Post https://52.151.112.196/api/v4/runners: x509  
: cannot validate certificate for 52.151.112.196 because it doesn't contain any IP SANs  
PANIC: Failed to register the runner. You may be having network problems.  
alessia@alessia-virtualbox:~$
```

Figura 20: GitLab - Errore presentato con Runner in Linux



```
alessia@alessia-virtualbox: ~  
File Azioni Modifica Visualizza Aiuto  
alessia@alessia-virtualbox: ~  
alessia@alessia-virtualbox:~$ docker run --rm -it -v /srv/gitlab-runner/config:/etc/gitlab-runne  
Runtime platform arch=amd64 os=linux pid=6 revision=2ebc4dc4  
Running in system-mode.  
  
Enter the GitLab instance URL (for example, https://gitlab.com/):  
https://52.151.112.196/  
Enter the registration token:  
x4BS-zxxjptdxWibsjsA  
Enter a description for the runner:  
[f8da0f73f468]:  
Enter tags for the runner (comma-separated):  
  
ERROR: Registering runner... failed runner=x4BS-zxx status=couldn't execute POST  
4/runners: Post https://52.151.112.196/api/v4/runners: x509: cannot validate certificate for 52.  
n any IP SANs  
PANIC: Failed to register the runner. You may be having network problems.  
alessia@alessia-virtualbox:~$
```

Figura 21: GitLab - Errore presentato con Runner in Docker

Le soluzioni a questo problema possono essere:

- ottenimento di un certificato custom gratuito tramite un qualsiasi provider di certificati SSL⁶;
- ottenimento di un'autocertificazione richiesta direttamente al Server GitLab⁷;
- ottenimento di un certificato direttamente dall'immagine Bitnami di Azure⁸.

2.3 Runner

I Runner fanno parte degli strumenti di C.I./C.D. che GitLab permette di usare. I Runner possono essere:

- *Shared*: esegue i jobs da tutti i progetti non assegnati;
- *Group*: esegue i jobs da tutti i progetti non assegnati nel proprio gruppo;
- *Specific*: esegue i jobs da tutti i progetti assegnati;
- *Locked*: non può essere assegnato ad altri progetti;
- *Paused*: non disponibile ad eseguire dei jobs.

I Runner possono essere creati per vari sistemi e in modalità diverse ma in questa trattazione le prove svolte sono state sottomesse solo tramite Runner installati in ambiente Linux e Docker[9].

2.3.1 Runner Linux

Installazione tramite i binari per sistema operativo Linux (Ubuntu):

```

1 # Download the binary for your system
2 sudo curl -L --output /usr/local/bin/gitlab-runner https://gitlab-runner-downloads.s3.
   amazonaws.com/latest/binaries/gitlab-runner-linux-amd64
3
4 # Give it permissions to execute
5 sudo chmod +x /usr/local/bin/gitlab-runner
6
7 # Create a GitLab CI user
8 sudo useradd --comment 'GitLab Runner' --create-home gitlab-runner --shell /bin/bash
9
10 # Install and run as service
11 sudo gitlab-runner install --user=gitlab-runner --working-directory=/home/gitlab-runner
12 sudo gitlab-runner start
13
14 # Register Runner Method
15 sudo gitlab-runner register --url <url del proprio progetto> --registration-token <
   registration token>

```

⁶<https://app.zerossll.com/certificate/verify/ab065ea33108ad1a8365825bee896ef2>

⁷<https://docs.gitlab.com/runner/configuration/tls-self-signed.html#supported-options-for-self-signed-certificates-targeting-the-gitlab-server>

⁸<https://docs.bitnami.com/general/apps/gitlab/administration/generate-configure-certificate-letsencrypt/>

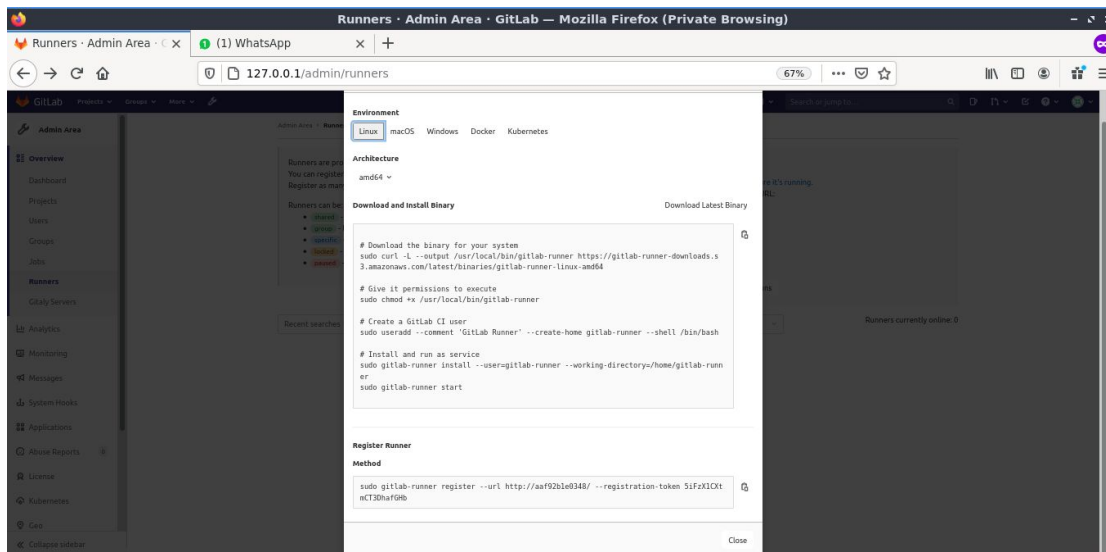


Figura 22: GitLab - Schermata installazione Runner con Linux

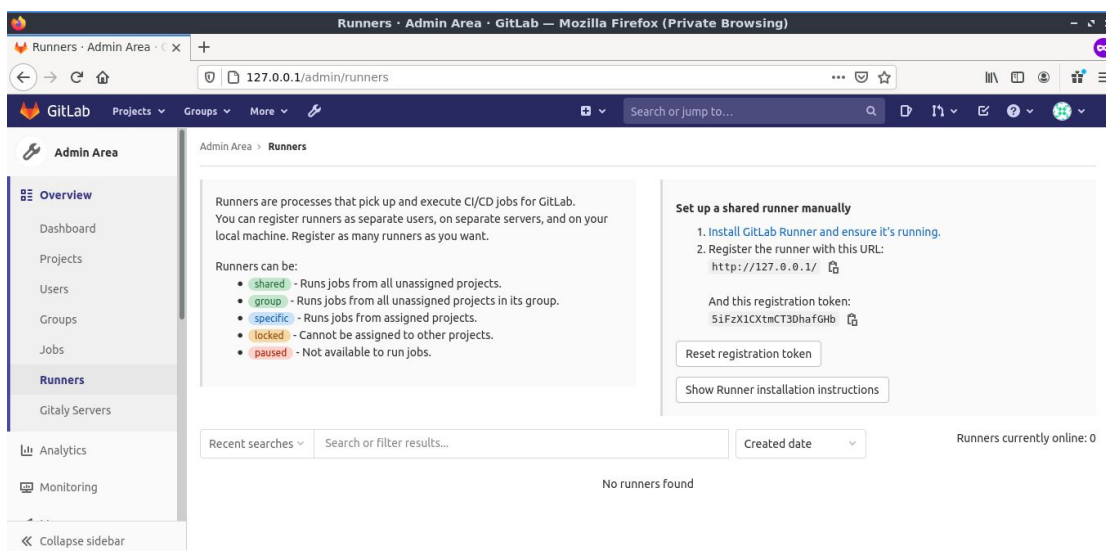


Figura 23: GitLab - Schermata informazioni per Runner utente root

Il sistema chiederà di inserire le informazioni relative al Runner. Le informazioni principali da inserire sono il dominio e il token che si possono trovare nella sezione **admin** → **overview** → **Runners**. In questa sezione inoltre si possono settare tutte le impostazioni per associare Runner a progetti o renderli pubblici, crearne dei nuovi o eliminare degli esistenti.

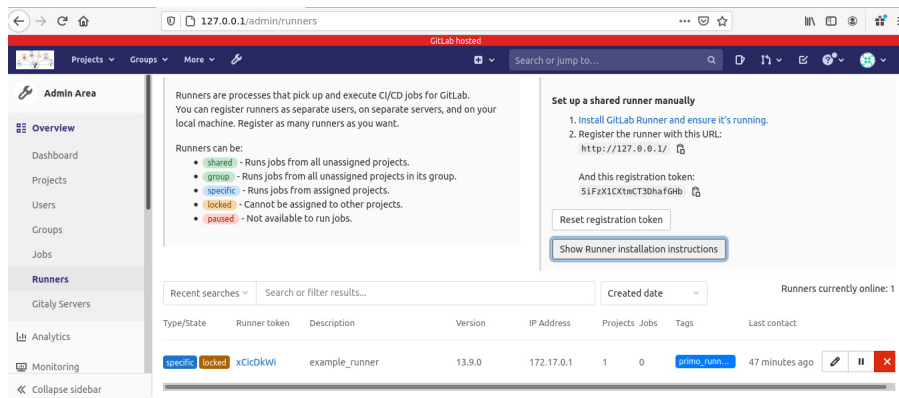


Figura 24: GitLab - Schermata di creazione avvenuta con successo di un Runner

Nel nostro caso lasciamo il dominio e il token già inseriti, inseriamo una descrizione del Runner, i tag del Runner, il tipo di Runner e la versione di ruby.

Il nostro Runner sarà quindi operativo e funzionante.

Una volta configurato e associato al progetto desiderato si deve aggiungere il file `.gitlab.ci.yml` contenente la determinazione delle operazioni da fare ad ogni push al progetto associato oppure al branch del progetto specificato. La presenza del file infatti consente in automatico a GitLab di lanciare le istanze del Runner ed eseguire le operazioni scritte[10].

2.3.2 Runner Docker

Per lanciare un Runner attraverso Docker si può utilizzare il mount di sistema locale o il volume di Docker.

Per la creazione di gitlab-runner tramite Docker sono disponibili due opzioni.

Opzione 1: usando local system volume mounts

```
1 sudo docker run -d --name gitlab-runner --restart always \
2 -v /srv/gitlab-runner/config:/etc/gitlab-runner \
3 -v /var/run/docker.sock:/var/run/docker.sock \
4 gitlab/gitlab-runner:latest
```

Opzione 2: usando Docker volumes

```
1 sudo docker volume create gitlab-runner-config
2 docker run -d --name gitlab-runner --restart always \
3 -v /var/run/docker.sock:/var/run/docker.sock \
4 -v gitlab-runner-config:/etc/gitlab-runner \
5 gitlab/gitlab-runner:latest
```

La prova fatta riguarda lo start del Runner `gitlab-runner` attraverso l'uso del volume di mount di sistema (Opzione 1). In seguito si è registrato il Runner appena creato all'immagine Docker di GitLab con il seguente comando:

```
1 sudo docker run --rm -it -v /srv/gitlab-runner/config:/etc/gitlab-runner gitlab/gitlab-
runner register
```

Verranno richiesti URL e registration token che si possono trovare o alla sezione Runner dell'admin area (<http://localhost/admin/runners>), se si sta creando il Runner come admin, oppure nei settings di un progetto, nel caso si voglia creare un Runner anche non essendo amministratore ma utente standard.

Come si può vedere dall'immagine seguente però si è occorsi in un errore fatale in quanto il dominio inserito non è un dominio https valido ma solo locale:

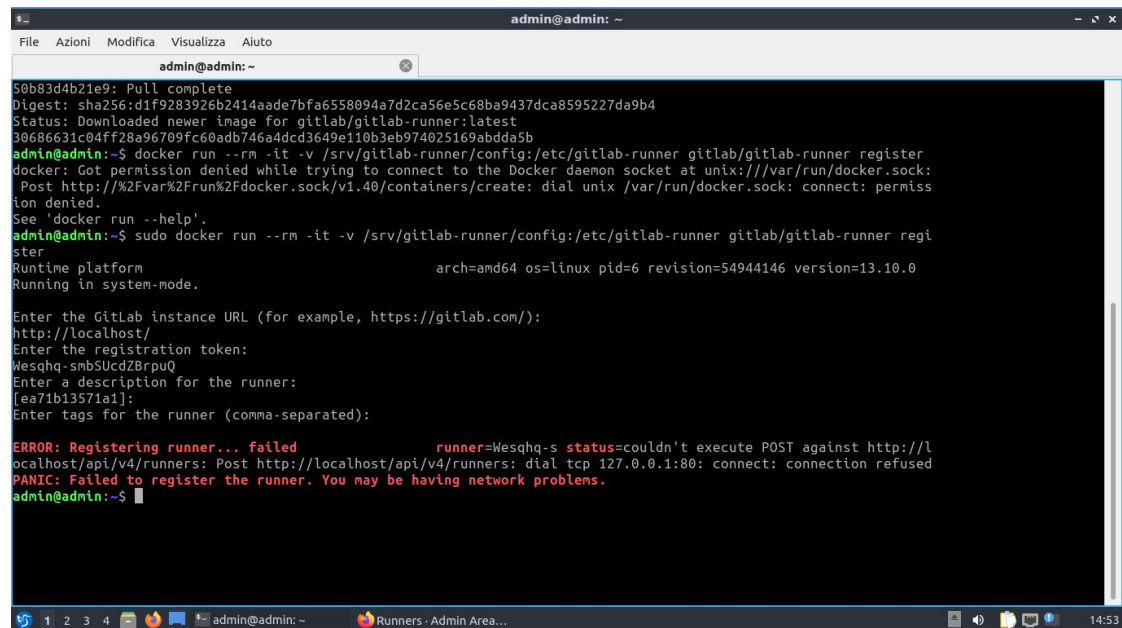


Figura 25: GitLab - Errore fatale registrazione nuovo Runner

Di seguito una possibile opzione per risolvere il problema attraverso la creazione di una rete interna per la simulazione di un dominio esterno.

Creiamo una rete interna a Docker:

```
1 sudo docker network create gitlabnet
```

Aggiungiamo GitLab alla rete appena creata:

```
1 sudo docker network connect gitlabnet gitlab
```

Creiamo un Runner che sia collegato a quella rete, inserendo come URL <http://gitlab>, il token e gli altri parametri:

```
1 sudo docker run --network gitlabnet --rm -it -v /srv/gitlab-runner/config:/etc/gitlab-
runner gitlab/gitlab-runner register
```

Per la risoluzione del problema riscontrato in precedenza ci siamo affidati ad una serie di risorse online quali Oramind[21], issue di GitHub[25] e forum di GitLab[26].

Nel caso infine si abbia a disposizione un'installazione di GitLab hostato in un server correttamente configurato ad un dominio pubblico HTTPS attivo, basterà inserire il proprio sito https e il token fornito dal proprio GitLab senza incorrere in alcun errore fatale.

3 Gogs

Gogs è un servizio self-hosted come buona alternativa di GitHub. Si tratta di un servizio multi-piattaforma in quanto si basa sul linguaggio di programmazione open-source Go⁹ e quindi è utilizzabile in qualsiasi macchina in grado di eseguire Go[13].

Il sito ufficiale di Gogs permette anche di sperimentare le sue funzionalità tramite una demo, di cui abbiamo riportato uno screenshot di seguito:

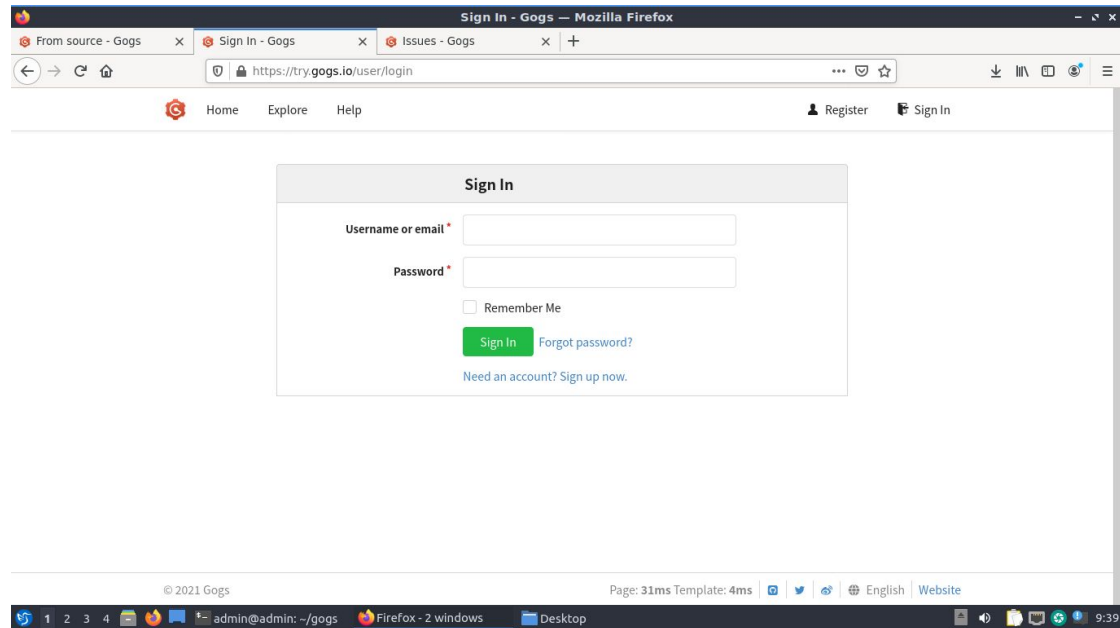


Figura 26: Gogs - Schermata di prova fornita da Gogs.io

3.1 Funzioni

Le principali funzioni di Gogs, alcune descritte nella documentazione ufficiale¹⁰ e altre dedotte dall'installazione di Gogs, sono riassunte nella tabella seguente:

⁹[https://it.wikipedia.org/wiki/Go_\(linguaggio_di_programmazione\)](https://it.wikipedia.org/wiki/Go_(linguaggio_di_programmazione))

¹⁰https://gogs.io/docs/features/custom_template

	Gogs
<i>migrazione</i>	✓
<i>issues</i>	✓
<i>ToDo list</i>	✗
<i>Dev-Ops PipeLine</i>	✗
<i>gestione gruppi</i>	✓ (organization)
<i>gestione utenti da admin</i>	✓
<i>gestione repository</i>	✓
<i>branches</i>	✓
<i>fork</i>	✓
<i>web Integrated Development Environment</i>	✓
<i>built-in container register</i>	✗
<i>activity</i>	✓ (dashboard)
<i>milestones</i>	✓
<i>controllo avanzato degli accessi</i>	✓
<i>webhook</i>	✓
<i>wiki</i>	✓
<i>snippet</i>	✗
<i>deploy keys</i>	✓
<i>aggiunta di chiavi crittografiche</i>	✓
<i>supporto multilingua</i>	✓
<i>autenticazione esterna</i>	✓
<i>feed</i>	✗
<i>barra di ricerca</i>	✓ (explore)
<i>ricerca avanzata</i>	✗
<i>pull requests</i>	✓
<i>2FA</i>	✗

3.2 Installazione tramite Ubuntu

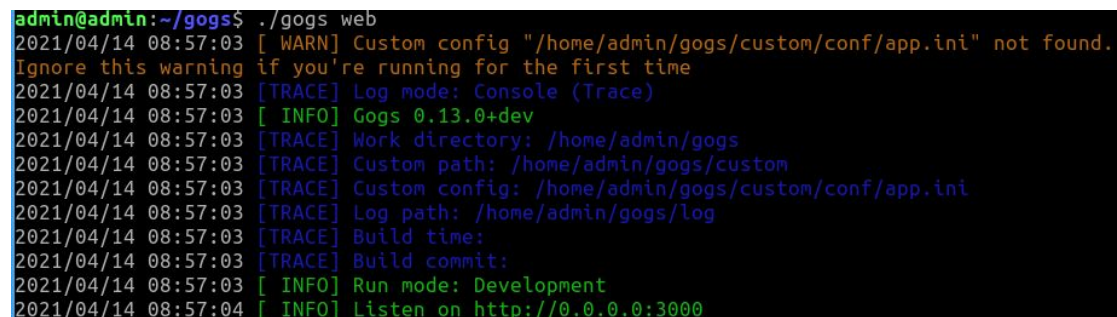
Essendo Gogs basato sul linguaggio di programmazione Go, esso è necessario per l'installazione e il funzionamento di Gogs. Come prima cosa quindi è necessario scaricare e installare Go dal sito ufficiale <https://golang.org/doc/install>, almeno la versione 1.14. Le prove che abbiamo svolto da qui in poi sono state effettuate su una macchina Ubuntu 21.04 e Lubuntu, ma le linee guida fornite dalla documentazione di installazione Gogs è molto simile anche per ambienti OS e Windows[14].

Successivamente abbiamo eseguito i seguenti comandi per creare un nuovo utente in cui eseguire e installare Gogs attraverso il repository offerto tramite GitHub:

```
1 sudo adduser --disabled-login --gecos 'Gogs' git
2 git clone --depth 1 https://github.com/gogs/gogs.git gogs
3 cd gogs
4 go build -o gogs
5 ./gogs web
```

Nel caso si incorra in qualche problema durante la build di Gogs (`go build -o gogs`) lanciare: `sudo apt-get install build-essential` che consente di installare i comandi essenziali del compilatore GCC necessario alla sua compilazione.

Una volta che l'installazione è andata a buon fine, collegarsi all'indirizzo suggerito dalle stampe risultanti dall'esecuzione di `./gogs web` come si può vedere dallo screen successivo. Nel nostro caso è in ascolto all'indirizzo 0.0.0.0/3000 che ci ha rimandato all'indirizzo: 0.0.0.0/3000/install che permette l'installazione personalizzata di Gogs.



```
admin@admin:~/gogs$ ./gogs web
2021/04/14 08:57:03 [ WARN] Custom config "/home/admin/gogs/custom/conf/app.ini" not found.
Ignore this warning if you're running for the first time
2021/04/14 08:57:03 [TRACE] Log mode: Console (Trace)
2021/04/14 08:57:03 [ INFO] Gogs 0.13.0+dev
2021/04/14 08:57:03 [TRACE] Work directory: /home/admin/gogs
2021/04/14 08:57:03 [TRACE] Custom path: /home/admin/gogs/custom
2021/04/14 08:57:03 [TRACE] Custom config: /home/admin/gogs/custom/conf/app.ini
2021/04/14 08:57:03 [TRACE] Log path: /home/admin/gogs/log
2021/04/14 08:57:03 [TRACE] Build time:
2021/04/14 08:57:03 [TRACE] Build commit:
2021/04/14 08:57:03 [ INFO] Run mode: Development
2021/04/14 08:57:04 [ INFO] Listen on http://0.0.0.0:3000
```

Figura 27: Gogs - Schermata installazione funzionante

Abbiamo impostato la configurazione corrente che nel nostro caso ha come *Database Type* SQLite3 mentre per il resto delle impostazioni abbiamo lasciato quelle di default.

Install Steps For First-time Run

If you're running Gogs inside Docker, please read [Guidelines](#) carefully before you change anything in this page!

Database Settings

Gogs requires MySQL, PostgreSQL, SQLite3 or TiDB (via MySQL protocol).

Database Type *

Path *

The file path of SQLite3 database.
Please use absolute path when you start as service.

Application General Settings

Application Name *

Put your organization name here huge and loud!

Repository Root Path *

All Git remote repositories will be saved to this directory.

Run User *

The user must have access to Repository Root Path and run Gogs.

Domain *

This affects SSH clone URLs.

SSH Port *

Port number which your SSH server is using, leave it empty to disable SSH feature.

☐ Use Builtin SSH Server

HTTP Port *

Port number which application will listen on.

Application URL *

This affects HTTP/HTTPS clone URL and somewhere in email.

Log Path *

Directory to write log files to.

☐ Enable Console Mode

Optional Settings

- ▶ Email Service Settings
- ▶ Server and Other Services Settings
- ▶ Admin Account Settings

[Install Gogs](#)

Figura 28: Gogs - Schermata configurazione

A questo punto Gogs è installato e in esecuzione all'indirizzo specificato nella configurazione. La creazione di un nuovo utente, per esempio, risulterà l'admin del sistema.

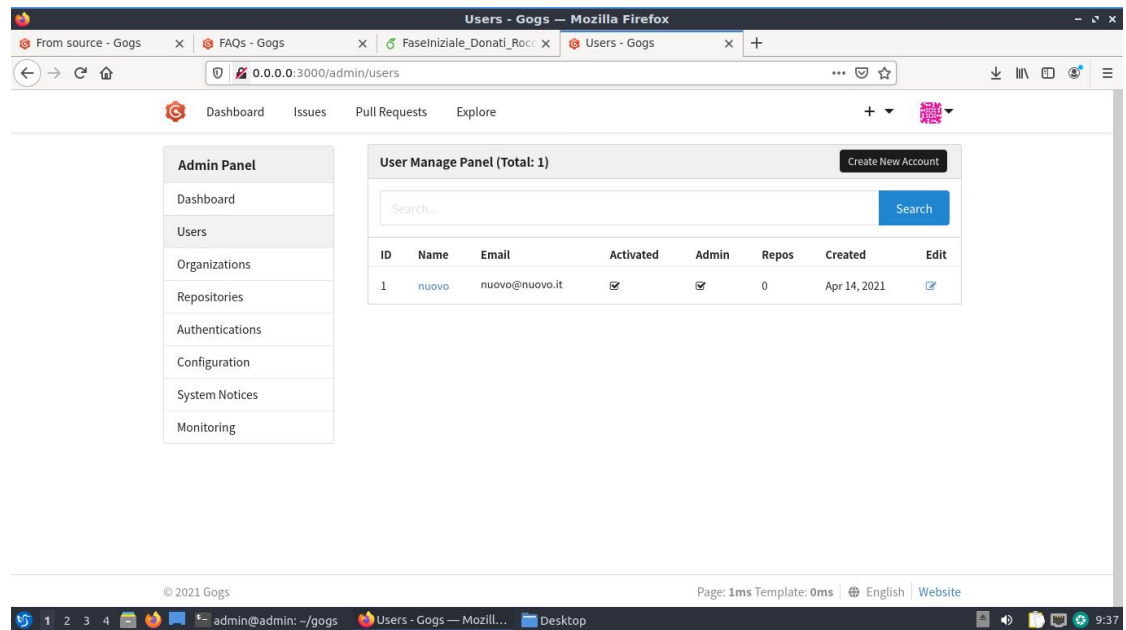


Figura 29: Gogs - Schermata gestione utenti da amministratore

4 Gitea

Gitea è un servizio progettato e forkato da Gogs che si basa sempre sul linguaggio Go. Dalla pagina ufficiale si può sia testare online l'ambiente di lavoro messo a disposizione da Gitea¹¹ che seguire le direttive di installazione per la realizzazione del proprio self-hosted Gitea[3].

4.1 Funzioni

In generale Gitea (come si può notare dal focus della tabella sottostante) presenta tutte le funzionalità di Gogs con in aggiunta dei supporti in più analizzati in questa sezione. Come con Gogs però non fornisce le funzionalità per le *Built-in CI/CD*.

Dalla documentazione emerge poi che Gitea è molto leggero e versatile adatto ad essere installato anche sui Raspberry Pi o comunque su sistemi o server che dispongono di poca RAM in quanto i requisiti sufficienti sono *2CPUcores* e *1GBRAM*.

¹¹<https://try.gitea.io/>

	Gitea
<i>migrazione</i>	✓
<i>issues</i>	✓
<i>ToDo list</i>	✗
<i>Dev-Ops PipeLine</i>	✗
<i>gestione gruppi</i>	✓ (organization)
<i>gestione utenti da admin</i>	✓
<i>gestione repository</i>	✓
<i>branches</i>	✓
<i>fork</i>	✓
<i>web Integrated Development Environment</i>	✓
<i>built-in container register</i>	✗
<i>activity</i>	✓ (dashboard)
<i>milestones</i>	✓
<i>controllo avanzato degli accessi</i>	✓
<i>webhook</i>	✓
<i>wiki</i>	✓
<i>snippet</i>	✗
<i>deploy keys</i>	✓
<i>aggiunta di chiavi crittografiche</i>	✓
<i>supporto multilingua</i>	✓
<i>autenticazione esterna</i>	✓
<i>feed</i>	✗
<i>barra di ricerca</i>	✓ (explore)
<i>ricerca avanzata</i>	✗
<i>pull requests</i>	✓
<i>2FA</i>	✓

4.2 Installazione tramite Docker

Per poter testare Gitea abbiamo utilizzato docker-compose seguendo la guida fornita dalla documentazione Gitea¹².

Prima di tutto creiamo una cartella “gitea”:

```
1 mkdir gitea
2 cd ./gitea
```

All'interno della cartella appena creata, inserire il file `docker-compose.yml` che permette di definire la macchina Docker con la determinazione della versione e delle porte per l'installazione e l'avvio di Gitea.

```
1 touch docker-compose.yml
```

```
1 version: "3"
2
3 networks:
4   gitea:
5     external: false
6
7 services:
8   server:
9     image: gitea/gitea:1.14.0
10    container_name: gitea2
11    environment:
12      - USER_UID=1000
13      - USER_GID=1000
14    restart: always
15    networks:
16      - gitea
17    volumes:
18      - ./gitea:/data
19      - /etc/timezone:/etc/timezone:ro
20      - /etc/localtime:/etc/localtime:ro
21    ports:
22      - "3000:3000"
23      - "222:22"
```

Una volta configurato il `docker-compose.yml` e lanciato il comando

```
1 sudo docker-compose up -d
```

Gitea in `localhost:3000` fa partire l'installazione proprio come avveniva in Gogs.

Dato che si tratta di un'immagine Docker per fermarla e rilanciarla basterà utilizzare i comandi “stop” e “restart” di Docker.

A fine installazione Gitea si presenterà così:

¹²<https://docs.gitea.io/en-us/install-with-docker/>

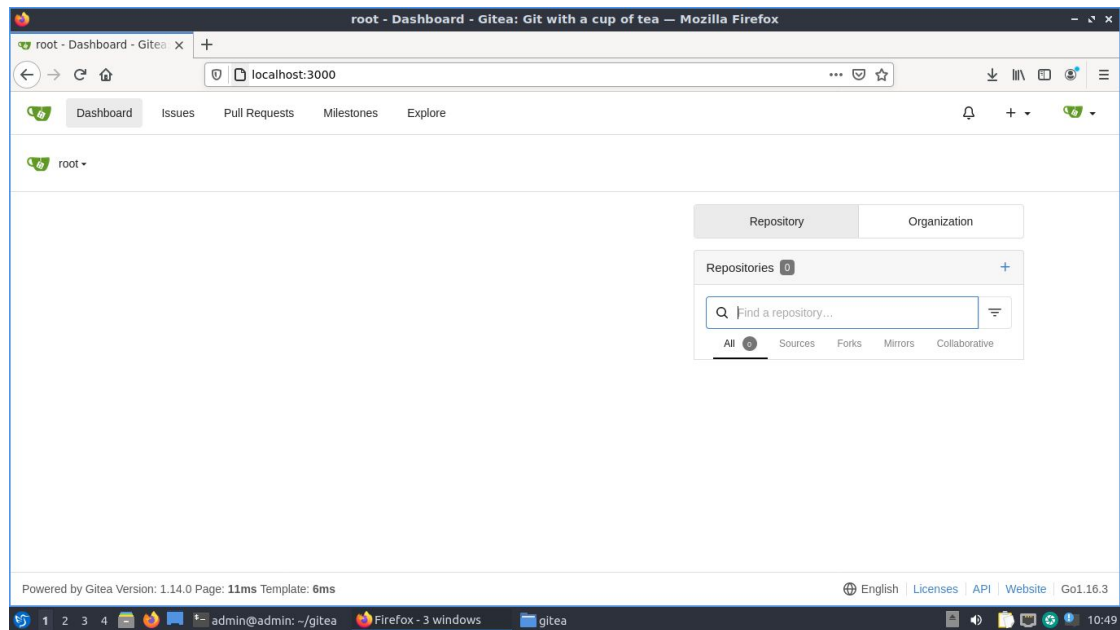


Figura 30: Gitea - Schermata principale admin area

Da come si può vedere infine dall'immagine lo stile e la configurazione di base è identica a quella di Gogs, quello che cambia sono le funzionalità in più come per esempio i supporti a Orgmode, CSV e l'autenticazione a due fattori.

5 GitBucket

GitBucket è una piattaforma Git open-source basata sulla JVM e fornita da Scala. La soluzione self-hosted può essere direttamente scaricata come archivio *.war* eseguibile attraverso Java 8 dal repository GitHub ufficiale oppure può essere testata attraverso una demo utilizzando come username e password "root", sempre attraverso il sito ufficiale[1].

5.1 Funzioni

Di seguito vengono riportate le funzionalità principali di GitBucket, dalle quali si può notare l'assenza delle DevOps-Pipeline e del supporto dei built-in container register. Per tale ragione l'utilizzo di GitBucket è limitato alla semplice gestione dei repository e degli utenti. Non presenta la possibilità di integrazione di controlli avanzati del codice risultando più limitato per un utilizzo accademico, ma è più leggero da scaricare e più facile da installare e configurare dato che viene il tutto delegato al compilatore Java.

	GitBucket
<i>migrazione</i>	✓
<i>issues</i>	✓
<i>ToDo list</i>	✗
<i>Dev-Ops PipeLine</i>	✗
<i>gestione gruppi</i>	✓
<i>gestione utenti da admin</i>	✓
<i>gestione repository</i>	✓
<i>branches</i>	✓
<i>fork</i>	✓
<i>web Integrated Development Environment</i>	✓
<i>built-in container register</i>	✗
<i>activity</i>	✓ (news feed)
<i>milestones</i>	✓
<i>controllo avanzato degli accessi</i>	✓
<i>webhook</i>	✓
<i>wiki</i>	✓
<i>snippet</i>	✓
<i>deploy keys</i>	✓
<i>aggiunta di chiavi crittografiche</i>	✓
<i>supporto multilingua</i>	✗
<i>autenticazione esterna</i>	✓
<i>feed</i>	✓
<i>barra di ricerca</i>	✓
<i>ricerca avanzata</i>	✗
<i>pull requests</i>	✓
<i>2FA</i>	✗

5.2 Installazione tramite Java

Per l'uso di GitBucket è sufficiente aver installato Java nella versione minima 8, scaricare il file `gitbucket.war` da <https://github.com/gitbucket/gitbucket/releases> ed eseguire il comando:

```
1 java -jar gitbucket.war
```

Si accede al server tramite browser all'indirizzo: `http://localhost:8080`

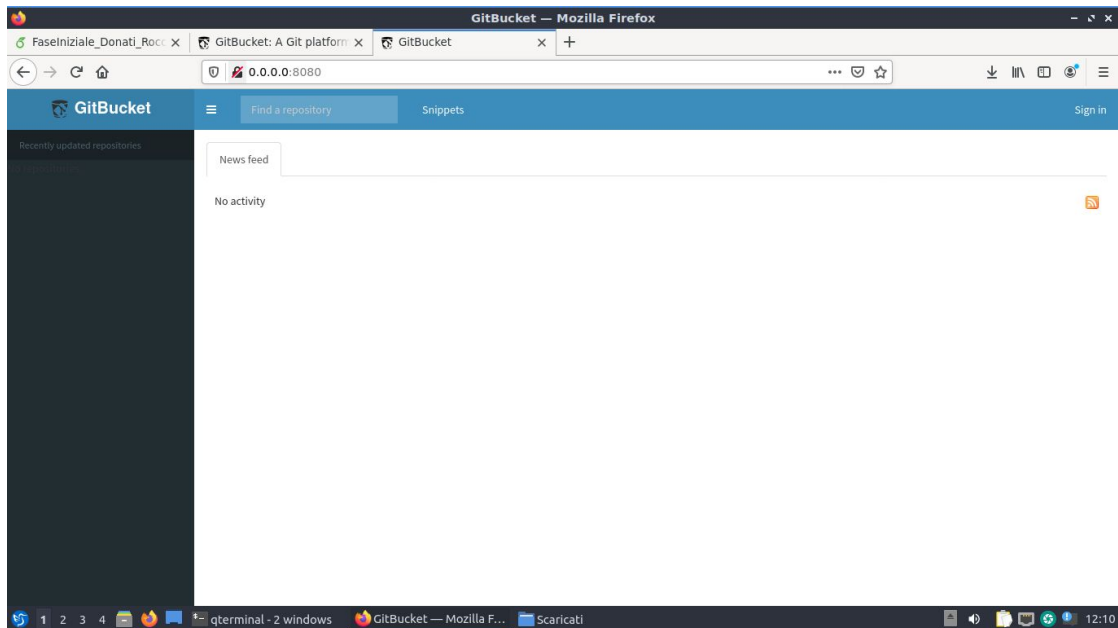


Figura 31: GitBucket - Schermata iniziale

Si può accedere con l'utente amministratore attraverso le credenziali user: "root" e password "root".

Esempio schermate relative alle issue:

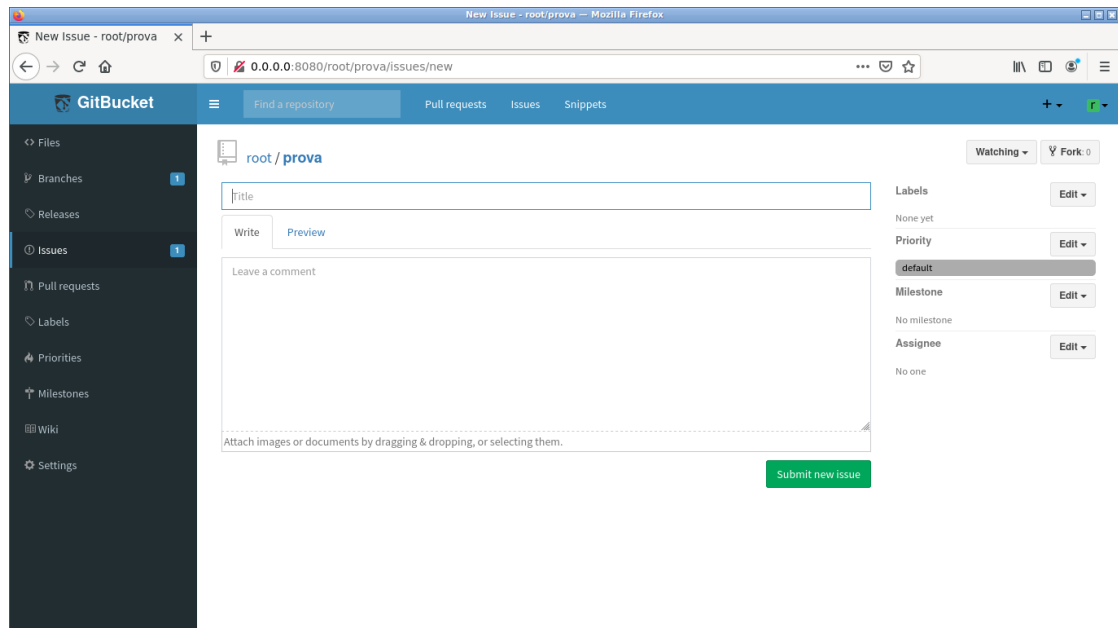


Figura 32: GitBucket - Schermata creazione issue

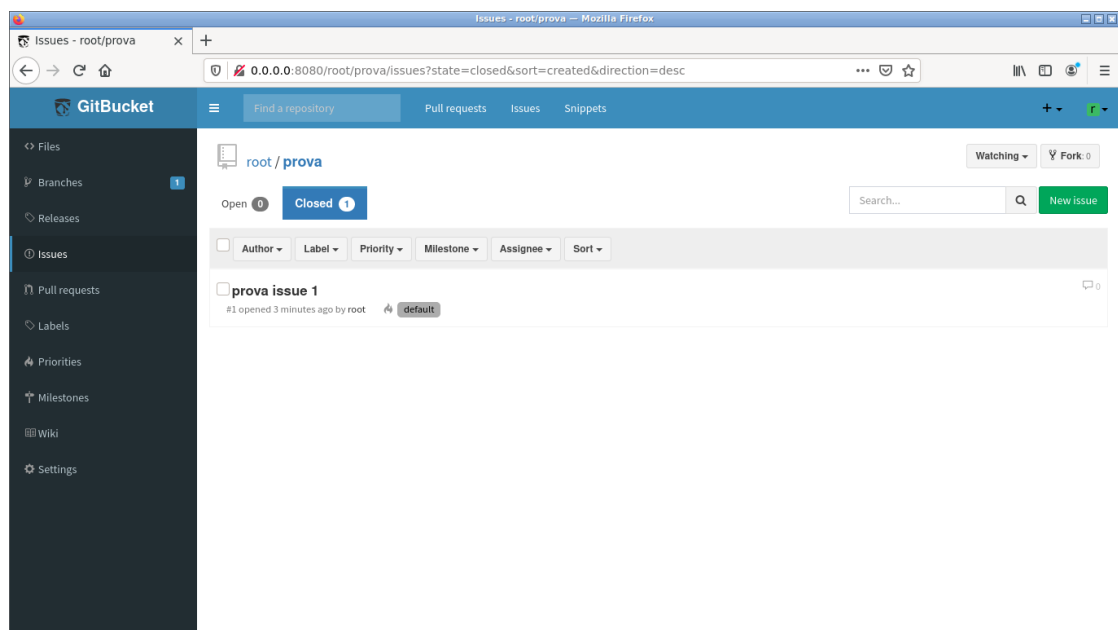


Figura 33: GitBucket - Schermata issue chiuse

Esempio schermate relative alle milestones:

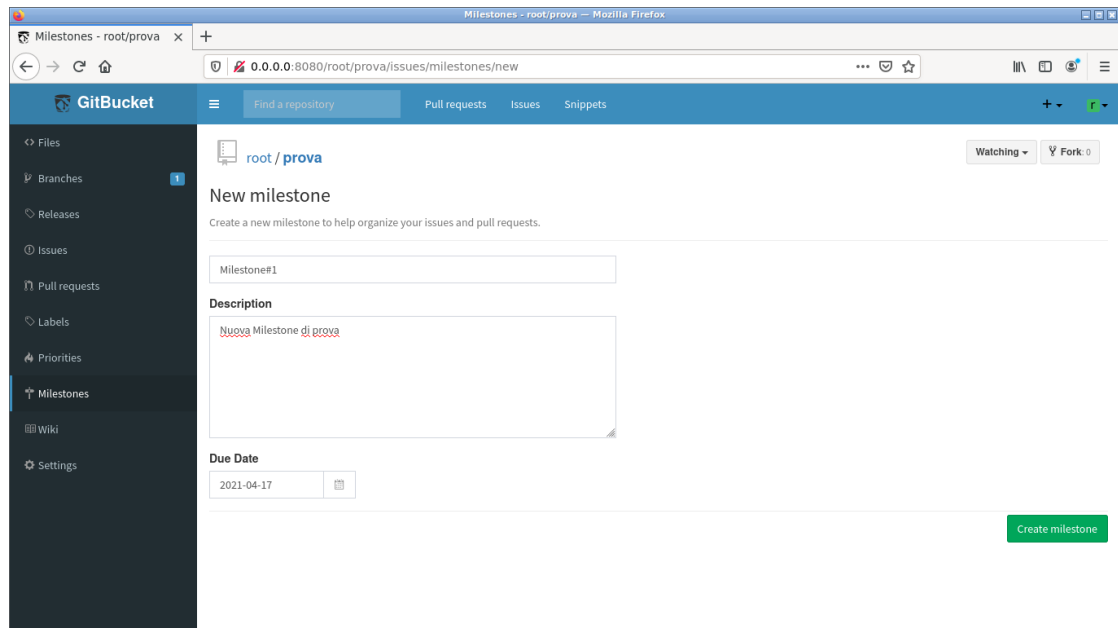


Figura 34: GitBucket - Schermata creazione milestone

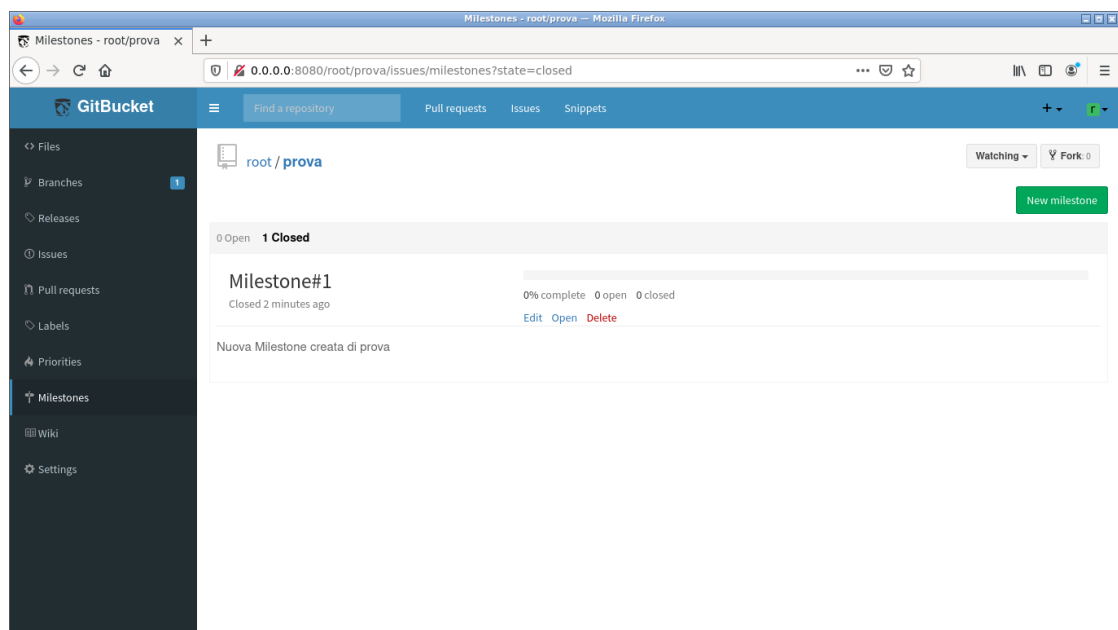


Figura 35: GitBucket - Schermata chiusura milestone

Di seguito alcuni screenshot delle funzionalità disponibili come amministratore:

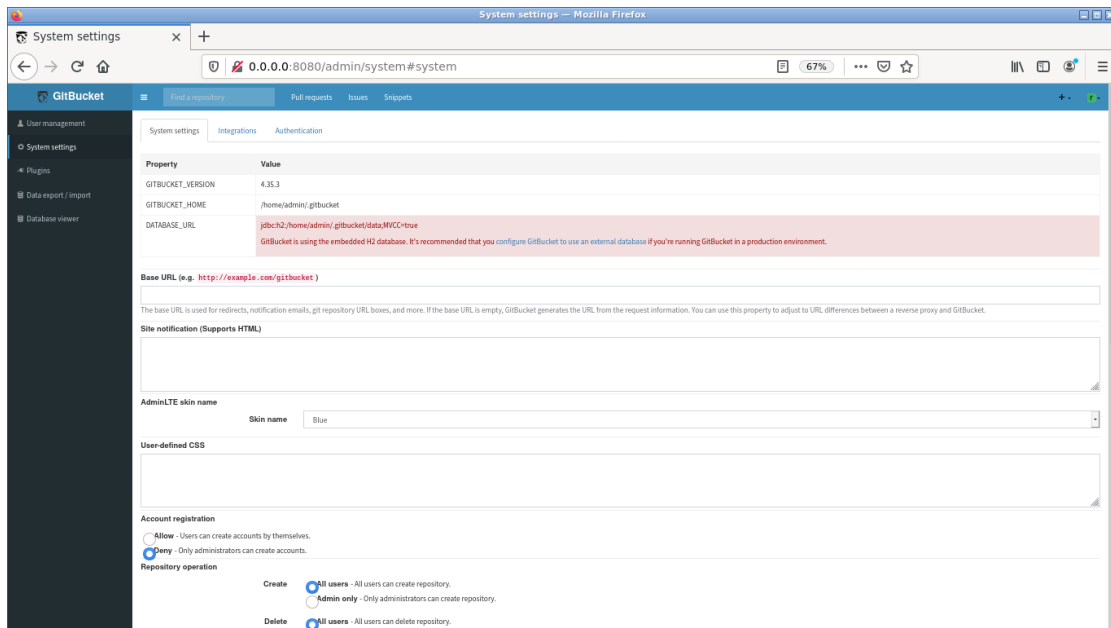


Figura 36: GitBucket - Schermata impostazioni di sistema

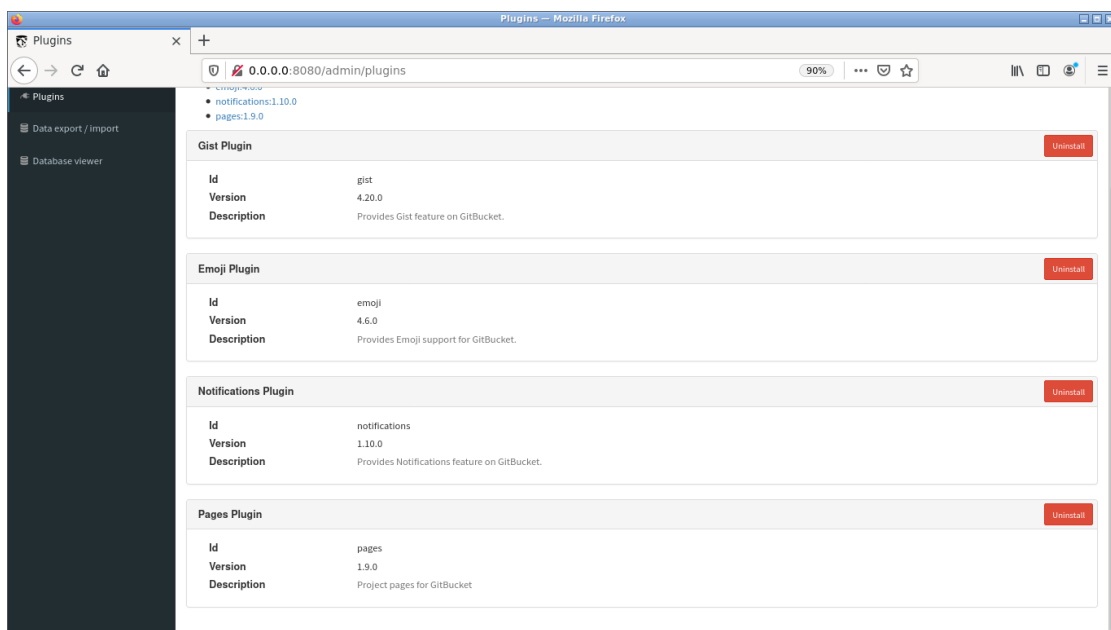


Figura 37: GitBucket - Schermata gestione plugins

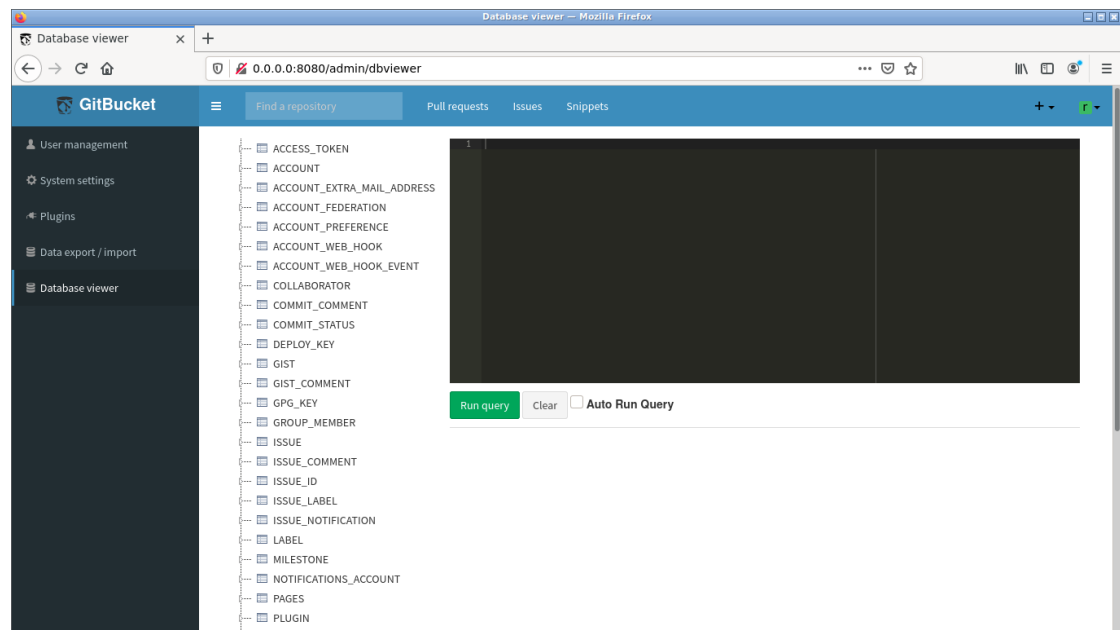


Figura 38: GitBucket - Schermata gestione database

6 GitPrep

GitPrep è un clone di GitHub disponibile con una licenza gratuita illimitata. Dalla community GitPrep sono messe a disposizione sia il download dell'ultima versione pronta all'installazione che il codice sorgente del progetto open-source [12].

6.1 Funzioni

Rispetto ad altre soluzioni esaminate le funzioni di GitPrep sono risultate più limitate, mancano infatti tutti i supporti alla continuous integration, al controllo dello sviluppo dei repository come milestones e la possibilità di migrare altri repository.

Le funzioni ritrovate durante lo studio e utilizzo di GitPrep sono state riportate nella tabella riassuntiva di seguito.

	GitPrep
<i>migrazione</i>	×
<i>issues</i>	✓
<i>ToDo list</i>	×
<i>Dev-Ops PipeLine</i>	×
<i>gestione gruppi</i>	×
<i>gestione utenti da admin</i>	✓
<i>gestione repository</i>	✓
<i>branches</i>	✓
<i>fork</i>	✓
<i>web Integrated Development Environment</i>	×
<i>built-in container register</i>	×
<i>activity</i>	×
<i>milestones</i>	×
<i>controllo avanzato degli accessi</i>	×
<i>webhook</i>	×
<i>wiki</i>	✓
<i>snippet</i>	×
<i>deploy keys</i>	×
<i>aggiunta di chiavi crittografiche</i>	✓
<i>supporto multilingua</i>	×
<i>autenticazione esterna</i>	×
<i>feed</i>	×
<i>barra di ricerca</i>	✓
<i>ricerca avanzata</i>	×
<i>pull requests</i>	✓
<i>2FA</i>	×

6.2 Installazione tramite Lubuntu

GitPrep si può installare in diverse modalità, una delle più semplici è l'installazione tramite pacchetto per bash Linux.

Iniziamo creando un nuovo utente **SuperUser** seguendo le indicazioni fornite nel repository ufficiale. Questo utente sarà utile per inserirci all'interno GitPrep, si tratta però di una pratica consigliata ma non obbligatoria.

Abbiamo inizialmente riscontrato un problema nella creazione del nuovo utente, il comando consigliato è `useradd`, ma da quel che sembra leggendo da vari forum è stato deprecato per `adduser`[20].

Creazione del nuovo utente SuperUser gitprep (facoltativa):

```
1 sudo adduser gitprep
2 sudo su - gitprep
```

La presente installazione necessita di Curl, nel caso non sia già installato lanciare:

```
1 sudo apt install curl
```

Controllo installazione Git e Perl(v.5.10.1+ necessaria):

```
1 git --version
2 perl -v
```

Installazione facoltativa di Git, se necessaria:

```
1 sudo apt install git
2 git --version
```

Installazione facoltativa di Perl, se necessaria (tramite perlbrew):

```
1 sudo apt install perlbrew
2 perlbrew install perl-5.16.3
3 perlbrew switch perl-5.16.3
4 perl -v
```

Scaricamento e spaccettamento del repository GitPrep nella giusta posizione:

```
1 curl -kL https://github.com/yuki-kimoto/gitprep/archive/latest.tar.gz > gitprep-latest.tar.gz
2 tar xf gitprep-latest.tar.gz
3 mv gitprep-latest gitprep
4 cd gitprep
```

Installazione:

```
1 ./setup_module
```

Test dell'installazione:

```
1 prove t
```

L'installazione del package potrebbe incorrere in un problema probabilmente causato da Perl. Questo problema definito come

```
1 build error in Text::Markdown::Hoedown
```

si risolve tramite il seguente comando trovato nella sezione Issue relativa alla repository ufficiale del progetto GitPrep¹³.

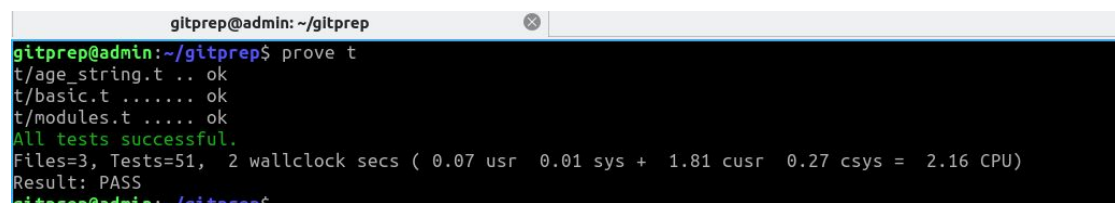
La soluzione consiste nel comando:

```
1 export PERL\_USE\_UNSAFE\_INC=1
```

Infine rieseguire i comandi precedenti:

```
1 ./setup_module
2 prove t
```

Il test finale andato a buon fine dovrebbe risultare simile a questa schermata:



```
gitprep@admin: ~/gitprep
gitprep@admin:~/gitprep$ prove t
t/age_string.t .. ok
t/basic.t ..... ok
t/modules.t ..... ok
All tests successful.
Files=3, Tests=51, 2 wallclock secs ( 0.07 usr 0.01 sys + 1.81 cusr 0.27 csys = 2.16 CPU)
Result: PASS
gitprep@admin:~/gitprep$
```

Figura 39: GitPrep - Schermata test con successo

Installazione del database:

```
1 ./setup_database
```

Start del database:

```
1 ./gitprep
```

Il server sarà disponibile all'indirizzo: <http://localhost:10020>

Per lo stop del server (dentro la cartella `gitprep` contenente tutti i sorgenti dell'esecuzione):

```
1 sudo ./gitprep --stop
```

Dopo aver scelto la password dell'admin compare la seguente schermata:

¹³<https://github.com/yuki-kimoto/gitprep/issues/134>

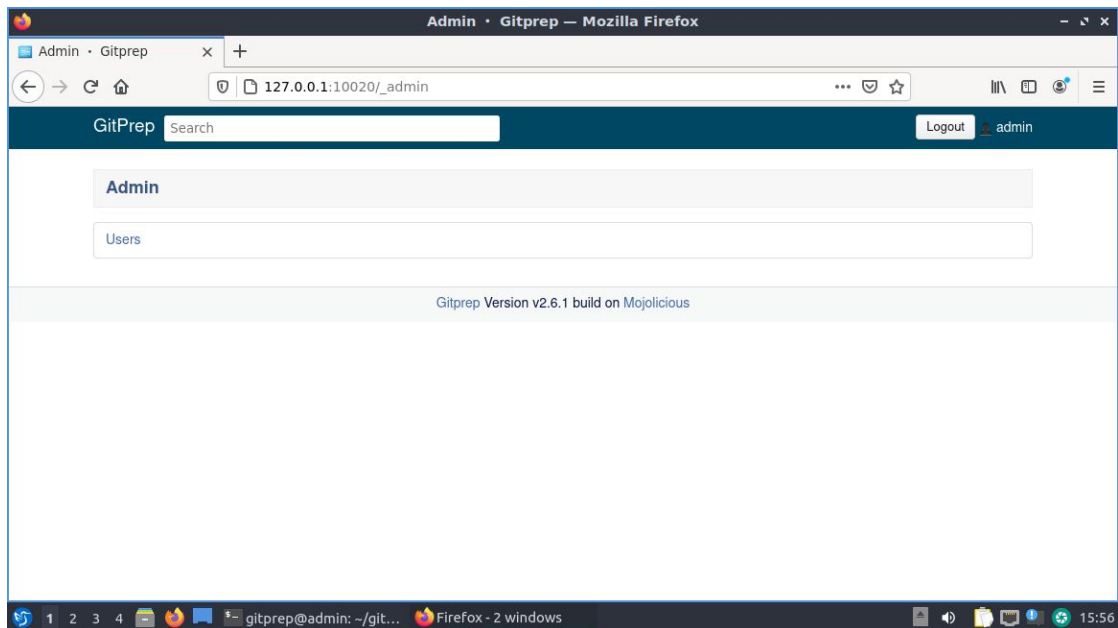


Figura 40: GitPrep - Schermata iniziale come utente admin

Creiamo un utente standard di prova:

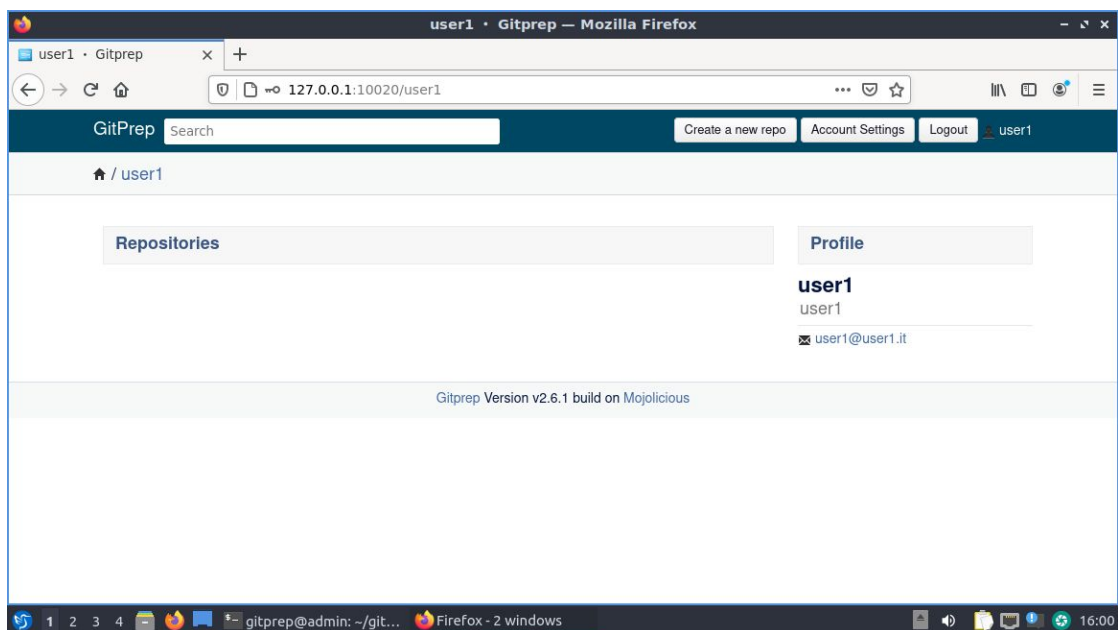


Figura 41: GitPrep - Schermata iniziale come utente standard

E' presente qualche funzione relativa alla gestione dei repository:

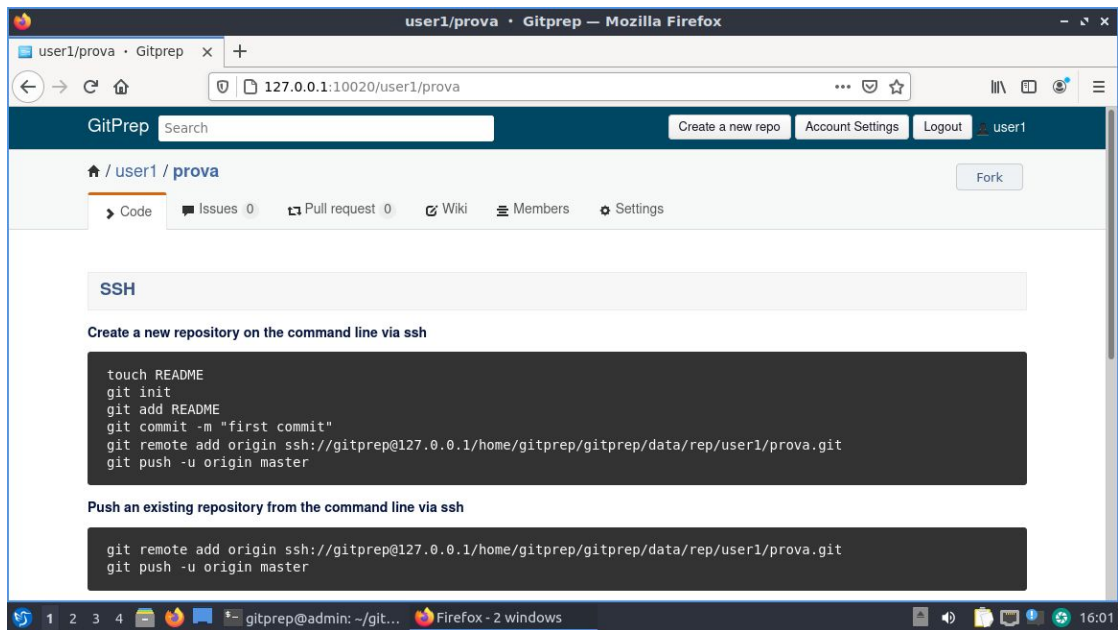


Figura 42: GitPrep - Schermata di management di un repository

7 Kallithea

Kallithea[15] è un progetto sotto licenza GPLv3 free e open source per la gestione del codice che supporta due sistemi di versionamento: Git e Mercurial. Ha un'interfaccia semplice da usare sia per gli utenti che per gli amministratori, che è possibile testare online tramite <https://kallithea-scm.org/repos/>.

7.1 Funzioni

Le principali funzionalità¹⁴ di Kallithea sono state definite dai creatori con nomi e sfumature diverse rispetto a quelle presentate fino ad ora. Per esempio, la sezione di activity è chiamata "journal" e riporta un vero e proprio storico pubblico di tutte le azioni intraprese dagli utenti iscritti alla piattaforma; mentre la possibilità di condivisione di snap di codice (snippet) è chiamata "gists" e permette di condividere codice sia in modo pubblico che privato.

Infine, non presenta nessun supporto per le autenticazioni esterne e l'utilizzo della crittografia simmetrica per l'accesso attraverso chiavi. Questo potrebbe pertanto essere un problema dal punto di vista della sicurezza e più limitante di altre soluzioni per un utilizzo accademico.

¹⁴<https://kallithea.readthedocs.io/en/default/readme.html#kallithea-features>

	Kallithea
<i>migrazione</i>	✓
<i>issues</i>	×
<i>ToDo list</i>	×
<i>Dev-Ops PipeLine</i>	×
<i>gestione gruppi</i>	✓
<i>gestione utenti da admin</i>	✓
<i>gestione repository</i>	✓
<i>branches</i>	✓
<i>fork</i>	✓
<i>web Integrated Development Environment</i>	✓
<i>built-in container register</i>	×
<i>activity</i>	✓ (journal)
<i>milestones</i>	×
<i>controllo avanzato degli accessi</i>	✓
<i>webhook</i>	×
<i>wiki</i>	×
<i>snippet</i>	✓ (gists)
<i>deploy keys</i>	×
<i>aggiunta di chiavi crittografiche</i>	×
<i>supporto multilingua</i>	×
<i>autenticazione esterna</i>	×
<i>feed</i>	✓
<i>barra di ricerca</i>	✓
<i>ricerca avanzata</i>	✓
<i>pull requests</i>	✓
<i>2FA</i>	×

7.2 Installazione tramite Ubuntu

Si può installare Kallithea sia su diversi server Unix based che Windows based.

Nei test effettuati si è considerata l'installazione Linux based attraverso le istruzioni fornite di seguito, che prendono spunto dalle indicazioni fornite dalla documentazione di Kallithea[16]. Nel nostro caso abbiamo eseguito i test su una macchina Ubuntu 21.04 e Lubuntu.

Innanzitutto si devono installare tutte le dipendenze necessarie:

```
1 sudo apt-get install build-essential git libffi-dev python3-dev -y
2 sudo apt-get install mercurial -y
3 sudo apt-get install python3-venv -y
4 sudo apt-get install npm -y
5 sudo apt-get update
```

Per scaricare e installare Kallithea:

```
1 hg clone https://kallithea-scm.org/repos/kallithea -u stable
2 cd kallithea
3 sudo apt-get install python3-venv -y
4 python3 -m venv venv
5 . venv/bin/activate
6 sudo apt install python3-pip -y
7 pip install --upgrade pip setuptools
8 pip install --upgrade -e .
9 python3 setup.py compile_catalog
```

Per preparare il file per il front-end:

```
1 kallithea-cli front-end-build
```

Spostarsi nella cartella desiderata. Nel nostro caso abbiamo creato la cartella "srv" e settato i permessi con:

```
1 cd ..
2 sudo mkdir srv
3 sudo chmod 777 srv
4 cd srv
```

Per creare il file my.ini:

```
1 kallithea-cli config-create my.ini
```

Creazione e settaggio dei permessi della cartella "repos" per i repositories più la creazione del database, tabelle e contenuto iniziale:

```
1 cd ..
2 sudo mkdir repos
3 sudo chmod 777 repos
4 cd srv
```

```
5 kallithea-cli db-create -c my.ini --force=yes --user=root --password=root --email=
  root@root.com --repos=/home/$USER/repos
```

Per eseguire Kallithea in un gearbox web server:

```
1 gearbox serve -c my.ini
```

Accedere alla pagina web Kallithea tramite `localhost:5000`.

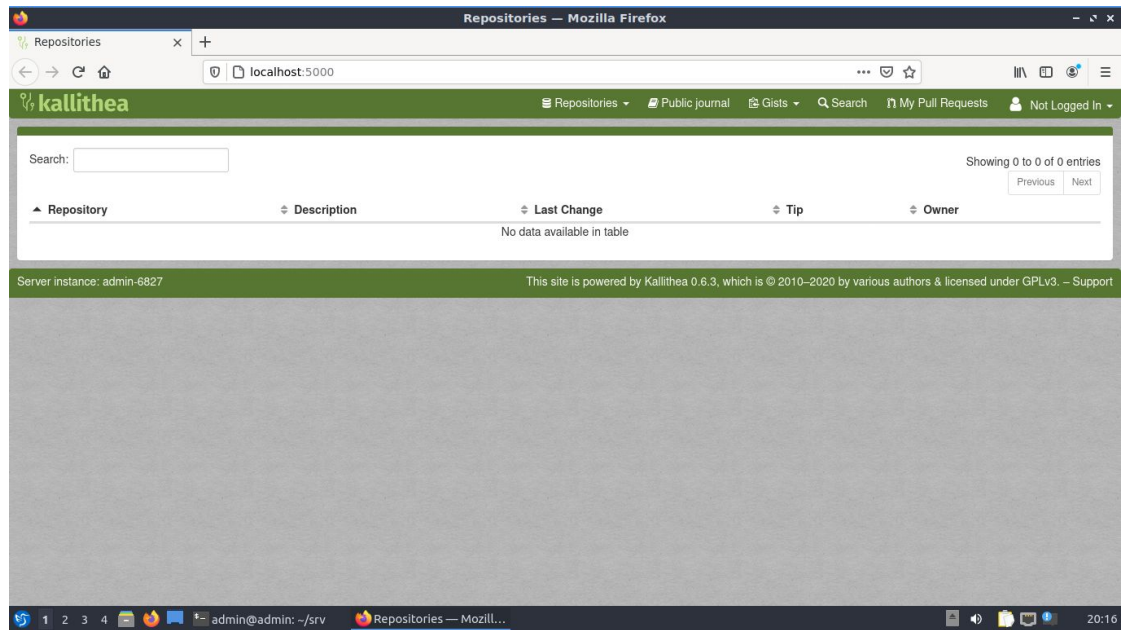


Figura 43: Kallithea - Schermata iniziale

Accedere all'utente amministratore tramite user "root" e password "root".

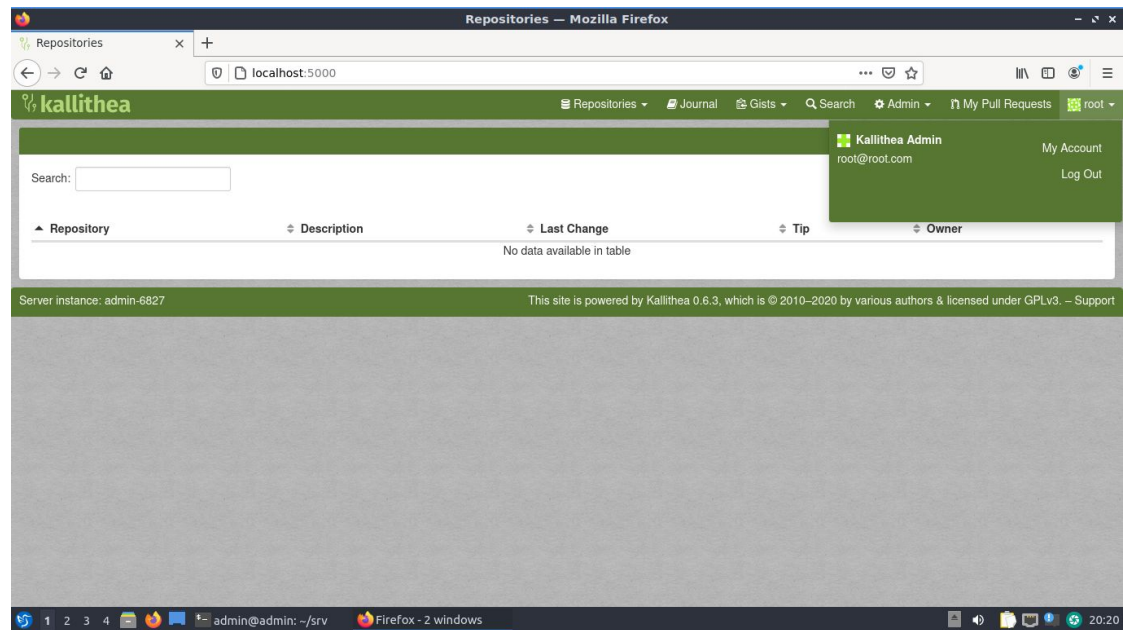


Figura 44: Kallithea - Schermata iniziale dell'utente amministratore

Di seguito alcune schermate delle funzionalità di Kallithea.

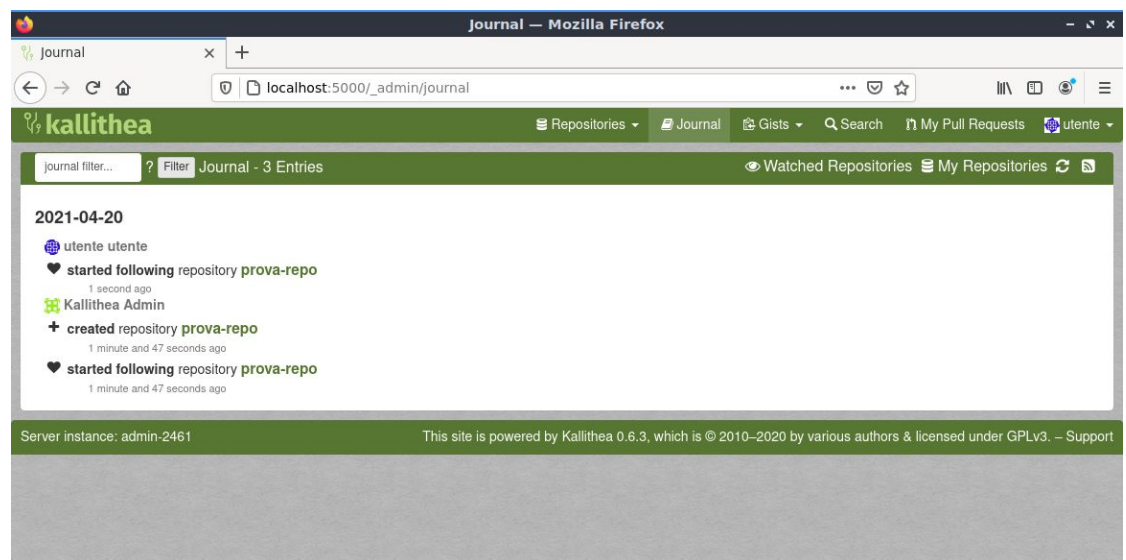


Figura 45: Kallithea - journal

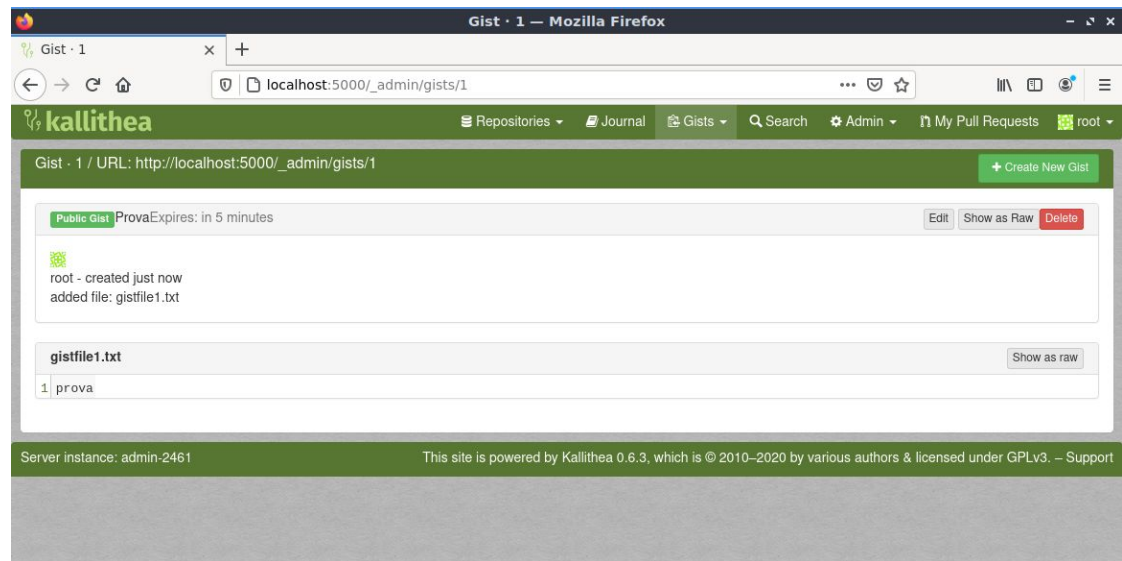


Figura 46: Kallithea - gists

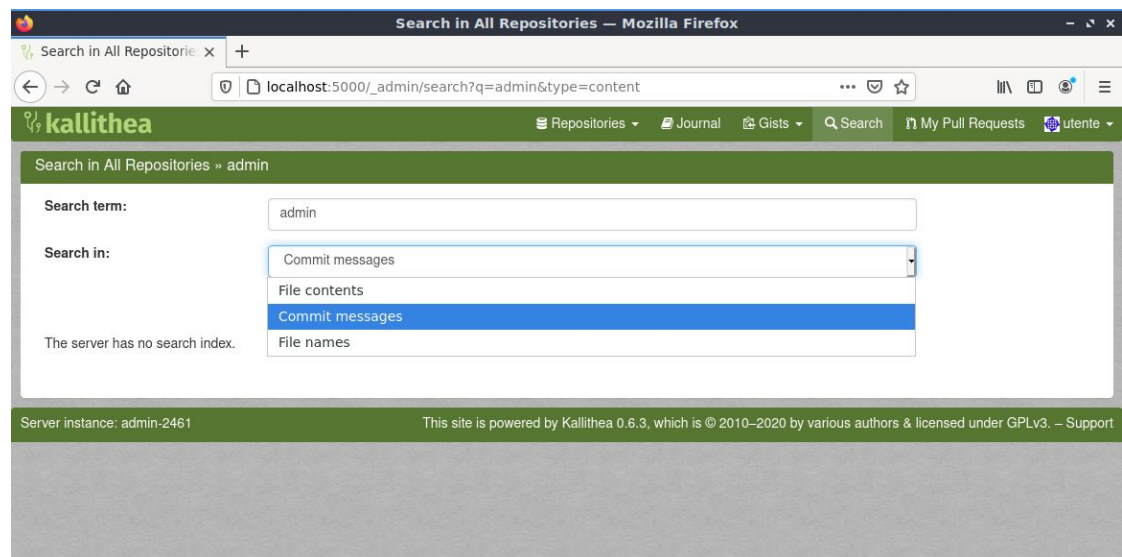


Figura 47: Kallithea - ricerca avanzata

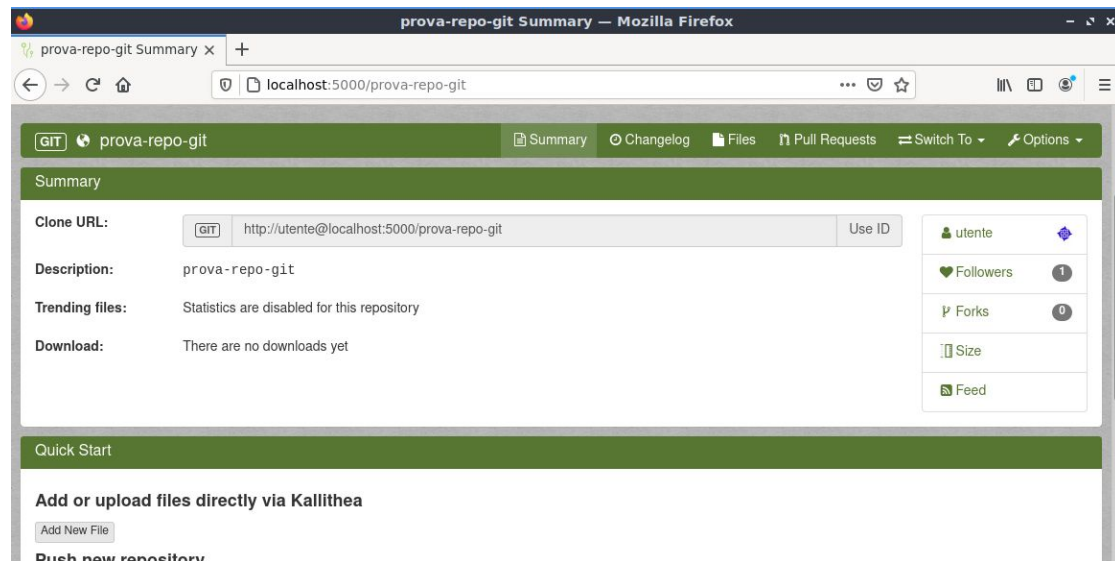


Figura 48: Kallithea - Schermata di un repository

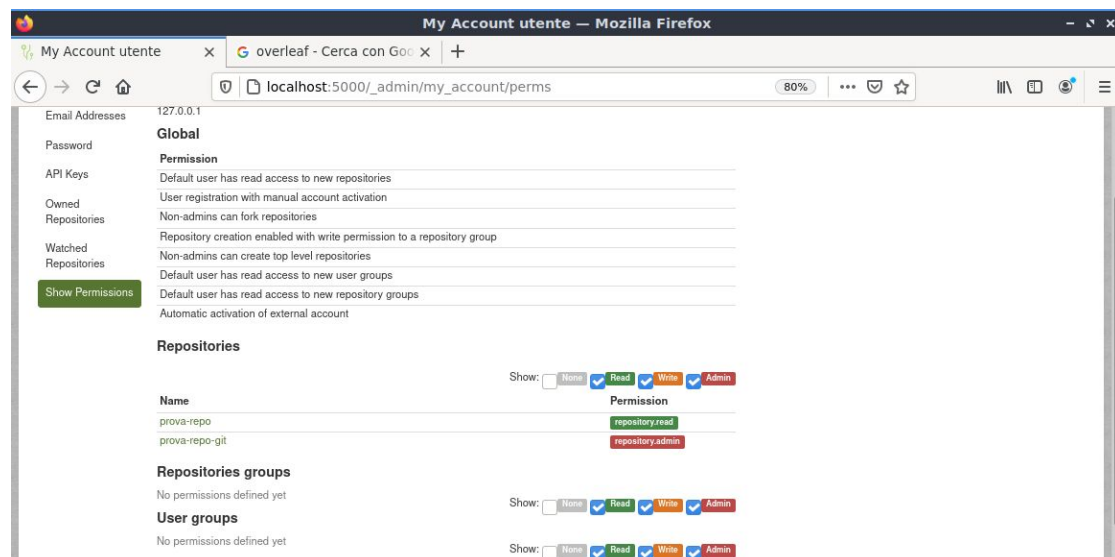


Figura 49: Kallithea - Schermata dei permessi di un utente

8 Tuleap

Tuleap[22] è una piattaforma per hostare e gestire i progetti con metodologia Agile mantenuta da Enalean¹⁵. E' utile per pianificare, tracciare, sviluppare, testare e condividere il software.

Esistono principalmente due versioni di Tuleap ad uso privato, la Enterprise Edition e la Community Edition.

La Enterprise Edition non esiste gratuitamente (oltre a test/demo) invece la versione Community è open-source e gratuita. Esiste inoltre una versione Cloud Hosted particolarmente adatta per le università, le scuole e gli studenti, basata su Tuleap Campus[23].

Tuleap mette a disposizione diverse web-application per testare le sue funzionalità, tra queste le più interessanti sono:

- *Tuleap.net* (<https://tuleap.net/>), una versione Cloud di Tuleap versione Enterprise, in cui i progetti devono essere open-source e free;
- *Tuleap Enterprise* (<https://www.tuleap.org/try/>).

Considerando che siamo interessati ad una versione self-hosted e gratuita, ci concentriamo sulla versione Community.

8.1 Funzioni

Alcune delle funzionalità disponibili:

- Tracking
- Documents and Deliveries
- Test management
- Scrum
- Kanban
- Continuous Integration
- Analytics
- Automate versioning, build, test and release

Un vantaggio di Tuleap è che ha molte funzionalità aggiuntive oltre a Git anzi, è pensato principalmente per l'uso di metodologia Agile con l'aggiunta di plugin tra cui Git¹⁶.

L'uso di una piattaforma come Tuleap pertanto è preferibile da team molto grandi, che hanno bisogno di molta coordinazione. Il suo utilizzo è molto più ampio rispetto ad una

¹⁵<https://www.tuleap.org/company/about-us/>

¹⁶<https://docs.tuleap.org/installation-guide/install-plugins.html>

semplice piattaforma Git. Nel nostro contesto, più orientato alla gestione di repository, sono perciò preferibili soluzioni meno complesse.

Infine, sia la struttura che l'interfaccia dell'intero sistema sono organizzate in maniera totalmente differente dall'approccio standard che abbiamo visto fino ad adesso.

8.2 Installazione tramite Docker

Tuleap è installabile self-hosted C.E. principalmente attraverso tre metodi:

- CentOS Server interamente dedicato;
- RedHat 7.x. Server interamente dedicato;
- Docker.

Tuleap quindi risulterebbe una soluzione vantaggiosa se si avesse un server interamente a disposizione per questo progetto. Un'altra considerazione da fare è che l'immagine Docker C.E. viene assolutamente sconsigliata per l'uso ma solo per test/demo.

Abbiamo comunque scelto di testare su Docker una versione locale di Tuleap, con l'immagine Docker ottenibile da <https://hub.docker.com/r/enalean/tuleap-aio/>.

I comandi da eseguire sono stati scritti prendendo spunto dalle indicazioni fornite dalla pagina appena citata.

Per la creazione della macchina Docker:

```
1 sudo docker run --name tuleap-aio enalean/tuleap-aio:centos7
```

Stampare un indirizzo IP legato al container appena creato:

```
1 sudo docker inspect -f '{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}' tuleap-aio
```

Andare in `/etc` e modificare il file `hosts` inserendo una riga contenente l'indirizzo IP ottenuto precedentemente e collegarlo all'indirizzo `tuleap.local`. Il file dopo le modifiche si dovrebbe presentare come segue:

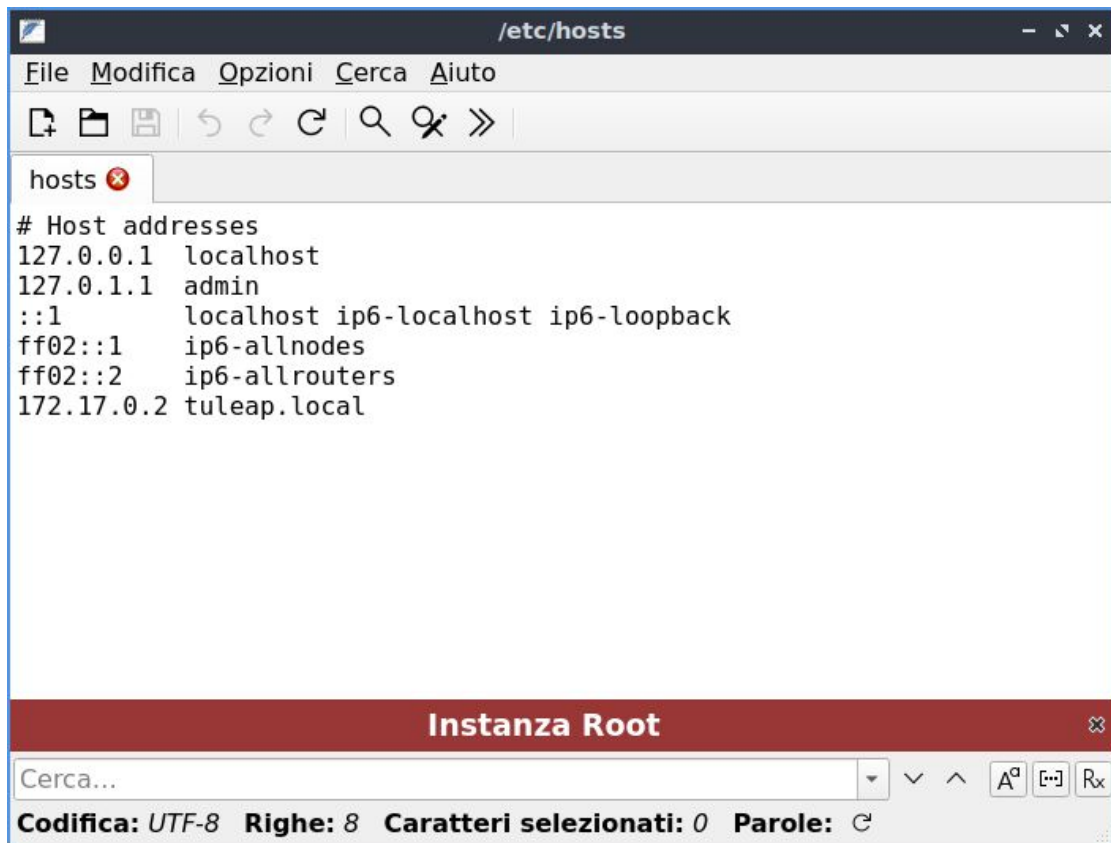


Figura 50: Tuleap - File hosts dopo la configurazione

Per ottenere la password di amministratore con cui fare il login come utente **admin** all'indirizzo <https://tuleap.local/>:

```
1 docker exec tuleap-aio cat /root/.tuleap_passwd
```

Comparirà un messaggio di warning relativo alla certificazione del sito, si proceda accettando il sito.

Dopo aver acceduto la schermata si presenta in questo modo:

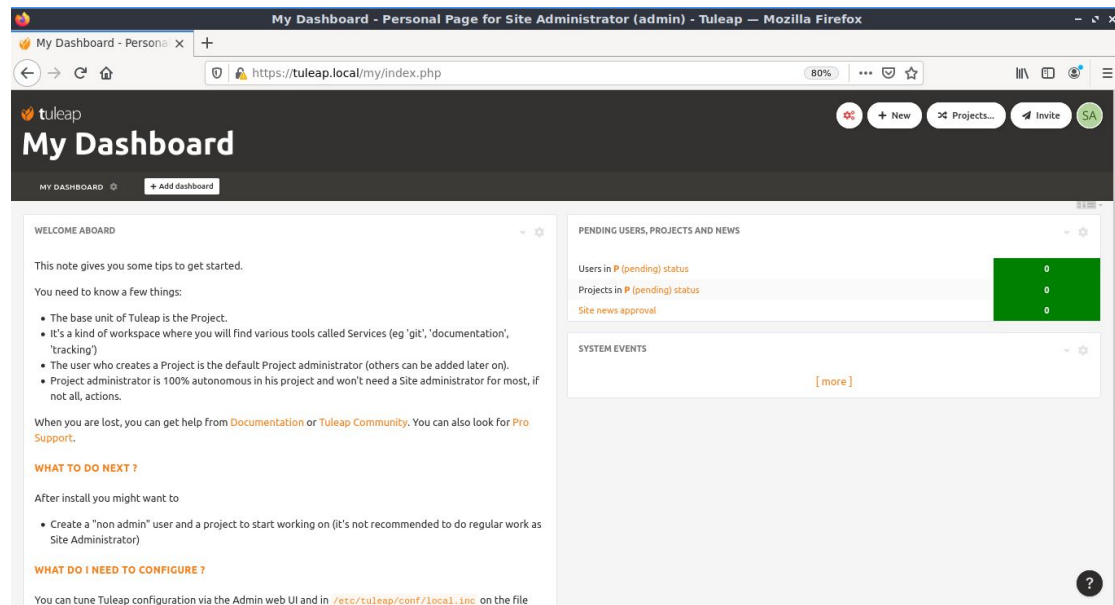


Figura 51: Tuleap - Schermata iniziale

9 Phabricator

Phabricator[19] è una suite LAMP open-source mantenuta da Phacility¹⁷. Supporta servizi CVS come Git, Mercurial e SVN e può essere mantenuta da chiunque attraverso il repository GitHub¹⁸ alla sezione *Contributor*.

E' possibile provare una istanza di test di Phabricator tramite Phacility al seguente link <https://admin.phacility.com/auth/register/> che consente di fare prove sia sulla creazione di repository che la creazione di nuovi progetti.

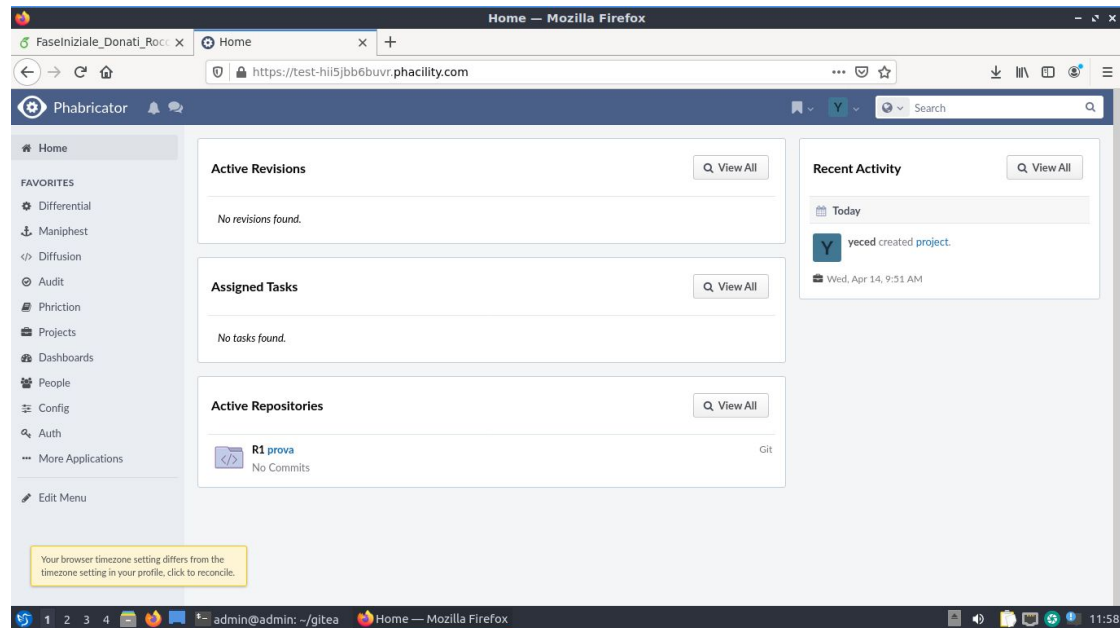


Figura 52: Phabricator - Schermata principale utente

9.1 Funzioni

Phabricator a differenza delle precedenti soluzioni non si basa solamente sulla gestione di repository Git ma offre una suite di applicazioni più ampia, che sono:

- *Differential*: un software utile alla revisione del codice sorgente;
- *Diffusion*: un browser per gli archivi di codice sorgente;
- *Herald*: un software atto al monitoraggio dei cambiamenti nel codice;
- *Maniphest*: un'applicazione per tenere traccia delle segnalazioni di bug all'interno dei software;
- *Phriction*: un software per tenere traccia dei documenti e delle loro modifiche.

¹⁷<https://www.phacility.com>

¹⁸<https://github.com/phacility/phabricator>

La versione completa e gratuita è solamente quella self-hosted ma Phacility consente anche una versione a pagamento, per maggiori informazioni consultare <https://www.phacility.com/pricing/>.

Phabricator ha un layout diverso dalle altre soluzioni Git self-hosted e non rispecchia molto le esigenze richieste da un ambiente accademico.

Infine, attualmente Phacility ha dato notizia sul sito ufficiale¹⁹ che sta dismettendo Phabricator dal 1 giugno 2021.

Per tutte queste ragioni abbiamo deciso di escluderlo dalle soluzioni applicabili al problema proposto da questo studio.

9.2 Installazione (non testata)

Per un'installazione locale i requisiti di base sono descritti nella documentazione²⁰ e necessitano di un ambiente che abbia in esecuzione un flavor Linux o un OS simile.

Non abbiamo testato l'installazione in locale di Phabricator in quanto non avevamo a disposizione molti dei prerequisiti richiesti e avremmo perciò impiegato più tempo a settare correttamente l'ambiente piuttosto che installare Phabricator stesso. Per queste ragioni abbiamo preferito non soffermarci sulla sua installazione locale ma ci siamo limitati a qualche prova online.

¹⁹https://admin.phacility.com/phame/post/view/11/phacility_is_winding_down_operations/

²⁰https://secure.phabricator.com/book/phabricator/article/installation_guide/

10 Confronto

Per concludere lo studio affrontato si è deciso di riportare un confronto generale tra le varie soluzioni proposte riflettendo sulle principali funzionalità, i modi di deploy e alcuni aspetti di sicurezza. Questo con lo scopo di far emergere le differenze dei diversi prodotti self-hosted andando a determinare quello più appropriato per la destinazione di questo progetto: una soluzione privata per il tracciamento e gestione di progetti interni all'ambito universitario.

10.1 Confronto funzionalità

I primi confronti che abbiamo reputato importanti da presentare sono emersi durante la fase di studio e di implementazione di Gitea, GitLab e Gogs.

Gitea e GitLab presentano sostanziali differenze nelle funzionalità: GitLab ne offre molte di più lato DevOps come la presenza di built-in CI/CD e DevOps pipeline a differenza di Gitea che invece ne è priva. Gitea e Gogs invece hanno le stesse features dato che si tratta di due soluzioni derivate, infatti Gitea è un progetto forkato da Gogs. Grafica e utilizzo sono molto simili e l'unica differenza sta nell'aggiunta da parte di Gitea dell'autenticazione a due fattori.

GitBucket poi ha circa le stesse operazioni della triade Gitlab, Gitea e Gogs pur non essendo una loro diretta derivazione in quanto realizzato come open-source in Scala e quindi basato sulla JVM.

Inoltre, secondo lo studio affrontato, Kallithea e GitPrep sono le proposte più scarse in termini di funzionalità (come si può vedere dalla tabella realizzata all'inizio della trattazione¹) dato che non presentano, per esempio il supporto alle issues per un repository, le DevOps pipeline, il supporto ad autenticazioni esterne e il supporto multilingua.

Infine, come opinione puramente soggettiva abbiamo constatato che tutte le soluzioni, a parte Kallithea, forniscono una buona interfaccia grafica e si presentano user-friendly dal punto di vista dell'utilizzo.

10.2 Confronto installazione

Per quanto riguarda l'ambito d'installazione la soluzione più facile e veloce è stata proposta da GitBucket in quanto, essendo un prodotto scala e compilabile con la JVM, si riduceva alla semplice compilazione del jar fornito sulla pagina GitHub ufficiale²¹.

Gogs, GitPrep e Kallithea sono stati considerati tramite l'installazione dei sorgenti forniti dai progetti open-source ed eseguite su macchine virtuali Ubuntu based. Queste installazioni hanno richiesto una maggiore attenzione sia per la loro configurazione che per la risoluzione di eventuali problemi riscontrati. Anche se questi prodotti sfruttano un utilizzo esiguo di risorse tra RAM, processore e archiviazione, abbiamo riscontrato dei netti miglioramenti soprattutto in fase di installazione al crescere della disponibilità di queste risorse. Gogs poi ha anche la possibilità di essere installato ed eseguito tramite immagine Docker, come è avvenuto nel caso di Gitea.

²¹<https://github.com/gitbucket/gitbucket>

Infine, in merito a GitLab, la sua installazione è stata fatta sia tramite immagine Docker che tramite Azure, utilizzando una macchina virtuale che da quanto si è visto si basa su un'immagine Bitnami. Non si tratta però delle uniche modalità per il suo deploy, ci sono infatti molte altre opzioni sia ufficialmente supportate (come linux package, Docker, Kubernetes e cloud providers) che non ufficiali (come arch linux package e freeBSD package)[11].

10.3 Confronto sicurezza

Dal punto di vista della sicurezza gli aspetti considerati sono stati principalmente la possibilità di usare l'autenticazione esterna, l'uso della 2FA e l'aggiunta di chiavi crittografiche.

Il primo aspetto consente infatti di accedere ai servizi utilizzando dei token di autenticità forniti da entità terze, questo per evitare di inserire i dati di accesso direttamente sulla nostra piattaforma. Il secondo aggiunge un ulteriore livello di sicurezza nell'accesso ai servizi: oltre a fornire il proprio login di accesso l'utente dovrà fornire anche un token (come un codice o un'autorizzazione) utilizzando un dispositivo esterno. Il terzo invece, permette di inserire chiavi crittografiche per l'utilizzo di SSH e la firma con GPG.

Gitea e GitLab sono le uniche che supportano l'autenticazione a due fattori mentre tutte le altre proposte esclusa Kallithea hanno il supporto alle chiavi crittografiche. Infine, solo GitPrep e Kallithea non supportano il meccanismo delle autenticazioni esterne.

Tutte queste ragioni hanno portato così a considerare GitLab e Gitea come le due implementazioni più affidabili e sicure tra quelle presentate.

10.4 Soluzione scelta

Per concludere quindi, la soluzione eletta come la migliore è stata quella proposta da GitLab che presenta la rosa di funzionalità più completa ed essenziale per una gestione avanzata dei repository di progetti. Questo nonostante sia la più onerosa in termini di risorse. E' l'unica inoltre ad avere le funzionalità di continuous integration incluse le DevOps pipeline, aspetto molto utile per un controllo tecnico e automatizzato dei progetti tracciati. Altro aspetto considerato è che si tratta di un prodotto supportato e mantenuto sia dall'omonima azienda che da migliaia di programmatori volontari che contribuiscono. Questo fa sì che eventuali bug e problemi emergano subito e che vengano forniti aggiornamenti, patch e supporto per tutti coloro che utilizzano questa soluzione. Infine, il design è accattivante, la web interface è molto chiara e user friendly ed è la soluzione più simile a GitHub. Tutte queste motivazioni hanno perciò propenso a prediligere GitLab come possibile implementazione definitiva.

11 Prova di deploy

Dato che è stata reputata GitLab la soluzione migliore si è deciso di realizzare un esempio reale di GitLab self-hosted, scegliendo di installarlo come immagine Docker su una macchina server.

11.1 Configurazione

Per farlo, abbiamo creato una macchina virtuale su Azure con Ubuntu Server 20.04 LTS, chiamando il gruppo *miogruppo1* e la V.M. *miagitlabvm1*. Le porte aperte in ingresso settate sono *22* (SSH), *80* (HTTP) e *443* (HTTPS) e l'accesso alla V.M. è stato configurato tramite credenziali.

Crea macchina virtuale ...

Informazioni di base Dischi Rete Gestione Avanzate Tag Rivedi e crea

Consente di creare una macchina virtuale che esegue Linux o Windows. Selezionare un'immagine da Azure Marketplace oppure usare un'immagine personalizzata. Compilare la scheda Informazioni di base, quindi la scheda Rivedi e crea per effettuare il provisioning di una macchina virtuale con i parametri predefiniti oppure esaminare ogni scheda per una personalizzazione completa. [Altre informazioni](#)

Dettagli del progetto

Selezionare la sottoscrizione per gestire le risorse distribuite e i costi. Usare i gruppi di risorse come le cartelle per organizzare e gestire tutte le risorse.

Sottoscrizione *	Azure per studenti
Gruppo di risorse *	(Nuovo) miogruppo1

[Crea nuovo](#)

Dettagli istanza

Nome della macchina virtuale *	miagitlabvm1
Area *	(Europe) Francia centrale
Opzioni di disponibilità	La ridondanza dell'infrastruttura non è richiesta
Immagine *	Ubuntu Server 20.04 LTS- Gen1
Istanza Spot di Azure	☐
Dimensioni *	Standard_B2ms - 2 cpu virtuali, 8 GiB di memoria (58,11 €/mese)

[Visualizza tutte le immagini](#)
[Visualizza tutte le dimensioni](#)

Figura 53: Configurazione iniziale Azure V.M.

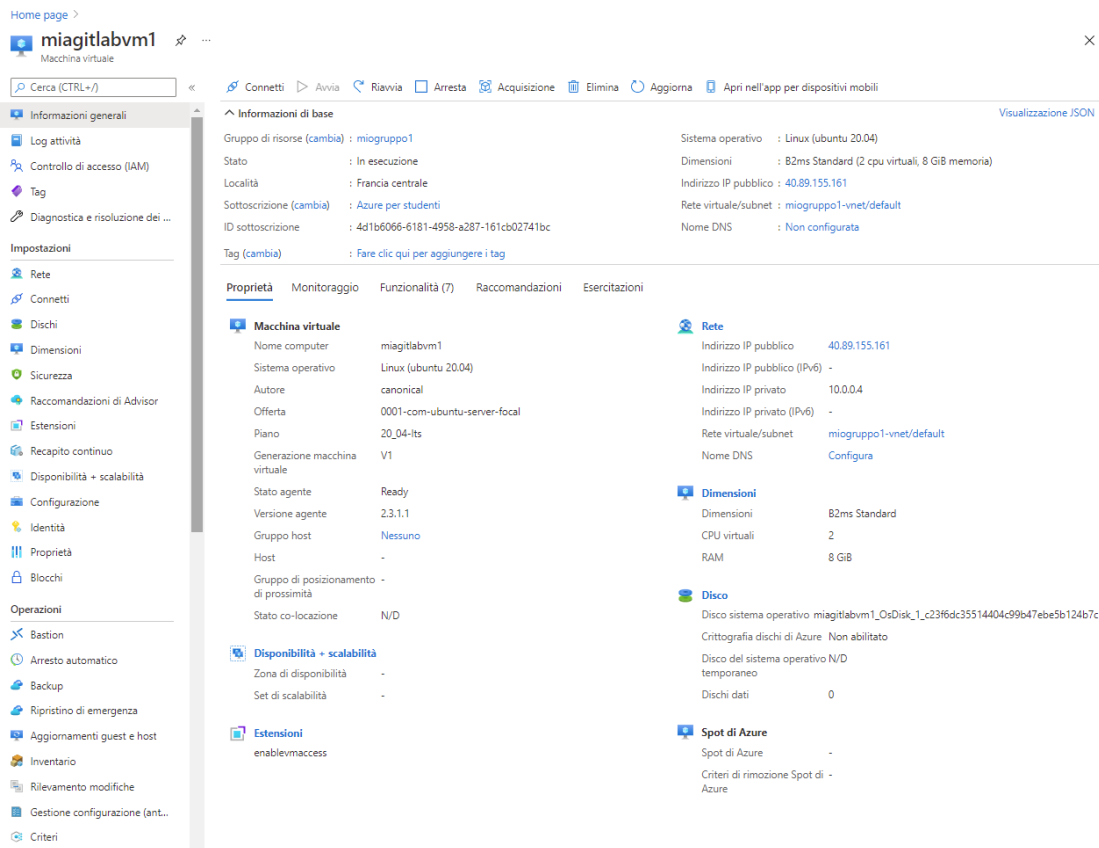


Figura 54: Configurazione finale Azure V.M.

Una volta creata la macchina e fatto l'accesso tramite SSH, seguendo la guida di Azure [24], abbiamo installato e configurato l'ambiente grafico e xrdp attraverso i seguenti comandi:

```
1 sudo apt-get update
2 sudo apt-get -y install xfce4
3 sudo apt-get -y install xrdp
4 sudo systemctl enable xrdp
5 echo xfce4-session > ~/.xsession
6 sudo service xrdp restart
```

Infine, per aprire la porta 3389, utile al servizio RDP, nella macchina virtuale usiamo Azure CLI (utilizzabile anche da web):

```
1 sudo apt install azure-cli
2 az vm open-port --resource-group miogruppo1 --name miagitlabvm1 --port 3389
```

La macchina ora è pronta a ospitare una versione hosted di GitLab, similmente a quanto spiegato nel capitolo 2.2.1:

```
1 sudo apt-get install docker.io
2 sudo docker pull gitlab/gitlab-ee
3 export GITLAB_HOME=/srv/gitlab
```

Per avviare il container, inserendo come *hostname* il dominio scelto e legato alla macchina virtuale Azure creata:

```
1 sudo docker run --detach \
2   --hostname gitlabunibo.tk \
3   --publish 443:443 --publish 80:80 --publish 1022:22 \
4   --name gitlab \
5   --restart always \
6   --volume $GITLAB_HOME/config:/etc/gitlab \
7   --volume $GITLAB_HOME/logs:/var/log/gitlab \
8   --volume $GITLAB_HOME/data:/var/opt/gitlab \
9   gitlab/gitlab-ee:latest
```

Da precisare la scelta della pubblicazione delle porte del container, in quanto attraverso l'utilizzo di una Azure V.M. la porta 22 dell'host sarà già occupata dai servizi legati a SSH (per l'accesso esterno alla V.M.). E' quindi necessario reindirizzare alla porta 22 della "network" interna al container GitLab una porta diversa dalla 22 dell'host. Nel nostro caso abbiamo scelto la porta 1022, ricordandosi di indicare la porta scelta ogni volta che si decide di fare il clone di un repository attraverso SSH, in quanto non si trova più su quella di default (22).

In alternativa, da utilizzare solo nel caso in cui non si voglia accedere alla V.M. attraverso SSH, si possono disabilitare tutti i processi legati a SSH all'avvio. Questo permette di utilizzare GitLab anche su questa porta²². Per controllare i processi attivi su una determinata porta (22) utilizzare il comando:

```
1 sudo lsof -i:22
```

Per disabilitare permanentemente invece i processi SSH all'avvio:

```
1 sudo systemctl disable sshd
```

Dato che nel nostro esempio si è fatto il collegamento tra la porta 1022 e la 22 per usare SSH con GitLab, abbiamo dovuto aggiungere la regola di rete per completare la configurazione. Per farlo, andare nella pagina della nostra V.M. su Azure e tramite "Impostazioni" → "Rete" → "Aggiungi regola porta in ingresso" inserire una nuova regola relativa alla porta 1022.

La password dell'utente root iniziale è presente nel file

/srv/gitlab/config/initial_root_password

e ha durata di 24 ore dopo la prima configurazione del sistema.

²²<https://webhostinggeeks.com/howto/how-to-disable-autostart-for-a-service-at-boot-on-linux-centos-7-rhel-7/>

Nel caso non venisse richiesto il reset della password di root, che dovrebbe avvenire in maniera automatica al primo accesso, si può entrare all'interno della nostra macchina Docker con:

```
1 sudo docker exec -it gitlab /bin/bash
```

per poi resettare la password dell'utente root a mano tramite questo comando²³:

```
1 gitlab-rake "gitlab:password:reset[root]"
```

Infine, viene impostato da dinamico a statico l'indirizzo che abbiamo, andando da "Informazioni generali" → "Proprietà" → "Rete" e "Configura" alla voce "Nome DNS".

11.2 Configurazione HTTPS

Successivamente si è registrato il dominio per il server GitLab usando come provider <https://my.freenom.com/> e associando al nome scelto l'indirizzo IP pubblico della macchina virtuale.

Abbiamo inoltre deciso, per ragioni di sicurezza, di mettere il dominio sotto HTTPS generando gratuitamente un certificato con <https://app.zeross1.com>. Seguendo poi le istruzioni fornite da <https://www.digicert.com/kb/ssl-support/pem-ssl-creation.htm> si è creato il `cert.pem`, come richiesto da GitLab²⁴. Questo file che contiene come prima parte il file `private.key` e come seconda il file `certificate` dovrà essere aggiunto a GitLab come segue:

```
1 sudo docker cp cert.pem gitlab:/etc/gitlab/trusted-certs
2 sudo docker exec -it gitlab /bin/bash
3 gitlab-ctl reconfigure
```

Ora che abbiamo un dominio HTTPS legato ad un indirizzo IP pubblico dobbiamo inserirlo all'interno del file `etc/gitlab/gitlab.rb` per avere il direccionamento automatico alla propria istanza GitLab self-hosted. Scommentare quindi la riga `# external_url 'GENERATED_EXTERNAL_URL'` sostituendo a `'GENERATED_EXTERNAL_URL'` il dominio desiderato, nel nostro caso diventa: `external_url 'https://gitlabunibo.tk'`.

Infine, digitare dentro la nostra immagine Docker

```
1 gitlab-ctl reconfigure
```

per riconfigurare[18].

²³https://docs.gitlab.com/ee/security/reset_user_password.html#reset-your-root-password

²⁴<https://docs.gitlab.com/omnibus/settings/ssl.html>

12 Conclusioni

Concludendo, sono state studiate in modo approfondito alcune delle soluzioni possibili a diventare un gestore di repository Git self-hosted. E' stata, in particolare, implementata la soluzione GitLab in quanto eletta la più adatta. Il sistema è stato realizzato sia in locale con una macchina virtuale, che attraverso un'implementazione cloud tramite il provider Azure.

12.1 Lavori futuri

Attualmente i test effettuati per la realizzazione del sistema proposto non prevedono l'implementazione su un server fisico. Quello che potrebbe essere fatto in futuro è l'installazione su un server centralizzato o meglio ancora su un sistema di server distribuiti. Il caso distribuito, ad esempio attraverso l'utilizzo dei server dei vari campus dell'ateneo, consentirebbe di gestire meglio il carico di lavoro, sfruttando la funzione *Geo*²⁵ messa a disposizione da GitLab. Inoltre un sistema distribuito, al contrario di uno centralizzato, consentirebbe di avere più punti di backup e una fault-tolerance più alta.

12.2 Cosa abbiamo imparato

La redazione di questo studio ha permesso di prendere confidenza con le varie strategie di implementazione di soluzioni Git self-hosted e con le principali tecnologie applicate a contesti distribuiti.

Sono infatti migliorate le competenze riguardanti ambienti Linux based e strumenti come Docker, per l'installazione delle proposte valutate durante questo studio. Inoltre, sono state acquisite molte abilità durante la realizzazione di un prototipo distribuito attraverso un servizio cloud quale Azure e maggiore confidenza è stata raggiunta nei protocolli di accesso remoto, come SSH.

Tutta l'esperienza infine, ha portato all'acquisizione di una competenza critica nei confronti di nuove tecnologie legate al mondo open-source.

²⁵<https://docs.gitlab.com/ee/administration/geo/>

Glossario

2FA Autenticazione a due fattori. 5, 7, 27, 32, 36, 43, 49, 62

activity Funzionalità che salva lo storico delle azioni di un utente o di una rete di utenti. 5, 7, 27, 32, 36, 43, 49

aggiunta di chiavi crittografiche Metodo di autenticazione al server remoto basato su chiave pubblica-privata di un utente. Questo permette di evitare l'accesso inserendo username e password (SSH) o di verificare i commit firmati (GPG). 5, 7, 27, 32, 36, 43, 49, 62

autenticazione esterna Possibilità di aggiungere una nuova risorsa di autenticazione al server attraverso servizi come LDAP, PAM, SMTP, Azure, SAML, etc. 5, 7, 27, 32, 36, 43, 49, 62

barra di ricerca Parte di un programma o sito web che consente di effettuare una ricerca all'interno dell'ambiente in esecuzione. 5, 7, 27, 32, 36, 43, 49

branches I branches sono delle catene di sviluppo separate da quella principale (master). I branches possono essere creati per gestire progetti complessi. Vengono usate ad esempio per la gestione di patch, feature e release. 5, 7, 27, 32, 36, 43, 49

built-in container register Alcuni servizi permettono di avere un gestore di container interno a servizi come DockerHub, ogni progetto ha la possibilità di avere un registro in cui tenere le proprie immagini Docker. 5, 7, 27, 32, 36, 43, 49

controllo avanzato degli accessi Il controllo avanzato degli accessi permette di gestire in maniera avanzata aspetti come chi e quali utenti possono accedere o registrarsi al sistema. 5, 7, 27, 32, 36, 43, 49

dashboard Vedi activity. 5, 27, 32, *Vai a* activity

deploy keys Permettono di dare accesso ai progetti attraverso coppie di chiavi. 5, 7, 27, 32, 36, 43, 49

Dev-Ops PipeLine Alcuni servizi permettono di creare un ambiente virtuale in cui svolgere determinati compiti. Le pipeline sono delle catene di operazioni svolte nell'ambiente virtualizzato. Ad esempio, sono molto utili in fase di test del software. 5, 7, 27, 32, 36, 43, 49

explore Vedi barra di ricerca. 5, 27, 32, *Vai a* barra di ricerca

feed File di testo che contiene le informazioni per reperire i nuovi contenuti pubblicati su un sito web. 5, 7, 27, 32, 36, 43, 49

fork Copia di repository remoto che viene inserita tra le proprie repository. Se non espressamente voluto i comandi git vengono eseguiti sul proprio repository e non sul repository da cui si è fatto il fork. 5, 7, 27, 32, 36, 43, 49

gestione gruppi Insiemi di utenti, utili per gestire gli accessi a determinate risorse. 5, 7, 27, 32, 36, 43, 49

gestione repository Gestione dei repository e dei permessi ad essi collegati. 5, 7, 27, 32, 36, 43, 49

gestione utenti da admin L'amministratore del sistema può gestire gli utenti. Ha quindi il potere di cancellarli, attivarli, decidere i loro permessi, ecc. 5, 7, 27, 32, 36, 43, 49

gists Vedi snippet. 5, 49, *Vai a* snippet

issues Letteralmente issues sono dei problemi; in ambito DevOps sono delle funzionalità che consentono di tracciare idee, bug, cose da fare in un determinato repository. Le issues possono essere aperte dagli utenti e vengono successivamente chiuse alla loro risoluzione. 5, 7, 27, 32, 36, 43, 49

journal Vedi activity. 5, 49, *Vai a* activity

merge requests Vedi pull requests. 5, 7, *Vai a* pull requests

migrazione La migrazione permette di importare dei repository da un servizio Git ad un altro. 5, 7, 27, 32, 36, 43, 49

milestones Le milestones sono un modo per tracciare e ottenere dei goal in un certo periodo di tempo opzionale. Si creano legate ad un repository e si possono aprire, chiudere e assegnare a un utente. 5, 7, 27, 32, 36, 43, 49

news feed Vedi activity. 5, 36, *Vai a* activity

organization Vedi gestione gruppi. 5, 27, 32, *Vai a* gestione gruppi

pull requests Richiesta di pull di cambiamenti da un branch ad un altro. Utili per la gestione di cambiamenti e revisione del codice. 5, 7, 27, 32, 36, 43, 49

ricerca avanzata Funzione di ricerca che consente di esplorare sulla base di una serie di parametri da definire. 5, 7, 27, 32, 36, 43, 49

snippet Funzionalità che permette di condividere snippet di codice (privati o pubblici). 5, 7, 27, 32, 36, 43, 49

supporto multilingua Il sistema supporta la gestione di più lingue. 5, 7, 27, 32, 36, 43, 49

ToDo list Collezione di tutte le attività da svolgere, solitamente vengono riportate qui le menzioni nei commenti, le issue e/o le merge request assegnate all'utente corrente. 5, 7, 27, 32, 36, 43, 49

web Integrated Development Environment Certi servizi permettono di gestire il codice direttamente da un editor avanzato hostato e visualizzato dal servizio. Esso permette una gestione facile e veloce del codice. 5, 7, 27, 32, 36, 43, 49

webhook Callback HTTP definite dall'utente che si scatenano ad eventi di repository, come pull-request, push di codice o merge request. 5, 7, 27, 32, 36, 43, 49

wiki Documentazione relativa ad un progetto o repository. 5, 7, 27, 32, 36, 43, 49

Elenco delle figure

1	GitLab - Schermata iniziale di reimpostazione password root	10
2	GitLab - Non aggiornato	10
3	GitLab - Aggiornato	10
4	GitLab - Schermata di dashboard utente root	11
5	GitLab - Schermata deploy keys	12
6	GitLab - Schermata Gitaly Servers	12
7	GitLab - Schermata gestione utenti da root	13
8	GitLab - Schermata creazione nuovo gruppo	13
9	GitLab - Schermata creazione nuovo progetto	14
10	GitLab - Schermata creazione nuovo snippet	14
11	GitLab - Schermata settings admin	15
12	GitLab - Schermata modifica appearance	15
13	GitLab - Schermata appearance modificata	16
14	GitLab - commit	16
15	GitLab - fork	17
16	GitLab - merge request	17
17	GitLab - merge esecuzione	18
18	GitLab - merge request eseguita	18
19	GitLab - branch	19
20	GitLab - Errore presentato con Runner in Linux	20
21	GitLab - Errore presentato con Runner in Docker	20
22	GitLab - Schermata installazione Runner con Linux	22
23	GitLab - Schermata informazioni per Runner utente root	22
24	GitLab - Schermata di creazione avvenuta con successo di un Runner	23
25	GitLab - Errore fatale registrazione nuovo Runner	24
26	Gogs - Schermata di prova fornita da Gogs.io	26
27	Gogs - Schermata installazione funzionante	28
28	Gogs - Schermata configurazione	29
29	Gogs - Schermata gestione utenti da amministratore	30
30	Gitea - Schermata principale admin area	34
31	GitBucket - Schermata iniziale	37
32	GitBucket - Schermata creazione issue	38
33	GitBucket - Schermata issue chiuse	38
34	GitBucket - Schermata creazione milestone	39
35	GitBucket - Schermata chiusura milestone	39
36	GitBucket - Schermata impostazioni di sistema	40
37	GitBucket - Schermata gestione plugins	40
38	GitBucket - Schermata gestione database	41
39	GitPrep - Schermata test con successo	45
40	GitPrep - Schermata iniziale come utente admin	46
41	GitPrep - Schermata iniziale come utente standard	46
42	GitPrep - Schermata di management di un repository	47

43	Kallithea - Schermata iniziale	51
44	Kallithea - Schermata iniziale dell'utente amministratore	52
45	Kallithea - journal	52
46	Kallithea - gists	53
47	Kallithea - ricerca avanzata	53
48	Kallithea - Schermata di un repository	54
49	Kallithea - Schermata dei permessi di un utente	54
50	Tuleap - File hosts dopo la configurazione	57
51	Tuleap - Schermata iniziale	58
52	Phabricator - Schermata principale utente	59
53	Configurazione iniziale Azure V.M.	63
54	Configurazione finale Azure V.M.	64

Riferimenti bibliografici

- [1] GitBucket. Gitbucket, 2020. URL <https://gitbucket.github.io/>.
- [2] V. Gite. 6 github alternatives that are open source and self-hosted, 2020. URL <https://www.cyberciti.biz/open-source/github-alternatives-open-source-self-hosted/>.
- [3] Gitea. Gitea, 2020. URL <https://docs.gitea.io>.
- [4] Gitlab. Gitlab, 2020. URL <https://about.gitlab.com/solutions/open-source>.
- [5] Gitlab. Gitlab configuration, 2020. URL <https://gitlab.com/gitlab-org/omnibus-gitlab/-/blob/master/doc/settings/gitlab.yml.md>.
- [6] Gitlab. Gitlab docker, 2020. URL <https://docs.gitlab.com/omnibus/docker/>.
- [7] Gitlab. Gitlab features, 2020. URL <https://about.gitlab.com/features/>.
- [8] Gitlab. Gitlab install guide, 2020. URL <https://docs.gitlab.com/ee/install/>.
- [9] Gitlab. Gitlab runner installations, 2020. URL <https://docs.gitlab.com/Runner/install/index.html>.
- [10] Gitlab. Gitlab runner, 2020. URL <https://docs.gitlab.com/ee/ci/yaml/README.html>.
- [11] Gitlab. Gitlab installations, 2021. URL <https://about.gitlab.com/install/>.
- [12] GitPrep. Gitprep, 2020. URL <http://gitprep.yukikimoto.com>.
- [13] Gogs. Gogs, 2020. URL <https://gogs.io/>.
- [14] Gogs. Gogs installation, 2020. URL https://gogs.io/docs/installation/install_from_source.
- [15] Kallithea. Kallithea, 2020. URL <https://kallithea-scm.org/>.
- [16] Kallithea. Kallithea installation, 2020. URL <https://kallithea.readthedocs.io/en/default/installation.html>, <https://kallithea.readthedocs.io/en/default/setup.html#setup>.
- [17] D. Morelo. 5 best self-hosted github alternatives, 2019. URL https://linuxhint.com/self_hosted_github_alternatives/.
- [18] J. Mutai. How to secure gitlab server with ssl certificate, 2019. URL <https://computingforgeeks.com/how-to-secure-gitlab-server-with-ssl-certificate/>.
- [19] Phacility. Phabricator, 2020. URL <https://www.phacility.com/phabricator/>.

- [20] Sconosciuto. useradd, 2020. URL <https://unix.stackexchange.com/questions/182180/why-is-the-home-directory-not-created-when-i-create-a-new-user>.
- [21] C. Sotiriou. Oramind, 2020. URL <https://oramind.com/private-cicd-using-gitlab-docker/>.
- [22] Tuleap. Tuleap, 2020. URL <https://tuleap.org/>.
- [23] Tuleap. Tuleap campus, 2020. URL <https://tuleap-campus.org/>.
- [24] Vari. Install and configure xrdp to use remote desktop with ubuntu, 2021. URL <https://docs.microsoft.com/en-us/azure/virtual-machines/linux/use-remote-desktop>.
- [25] Vari. Issue 1123 - docker gitlab, n.d.. URL <https://github.com/sameersbn/docker-gitlab/issues/1123>.
- [26] Vari. Issue - docker gitlab runner, n.d.. URL <https://forum.gitlab.com/t/cant-register-gitlab-runner-via-docker-compose/49049>.