Forgejo + Runner: il Git self-hosted a prova di bomba (e leak)

Forgejo: la leggerezza fatta a Git

Se vogliamo un server Git personale, abbiamo fondamentalmente tre strade. La prima consiste nell'usare GitHub o GitLab regalando i nostri fatti a terzi. La seconda è tirare su GitLab self-hosted, a patto di avere un server dedicato con 8GB di RAM pronti a essere fagocitati al primo avvio. La terza strada, l'unica sensata, si chiama **Forgejo** (fork comunitario di Gitea).

È scritto in Go, consuma pochissimo ed è incredibilmente completo. Ma un server Git senza una pipeline di CI/CD (integrazione e deploy continuo) è solo a metà dell'opera. Ecco perché ci serve affiancargli un runner ufficiale: **forgejo-runner**.

In questo articolo vedremo come dispiegare Forgejo tramite Docker e come configurare un runner nativo sull'host con un po' di sano hardening di Systemd, ottimo se dobbiamo compilare siti statici (ad esempio con Zola o Hugo) e piazzarli direttamente nella `/var/www/` del server senza fare accrocchi orribili con i permessi di Docker.

Passo 1: Dispiegare Forgejo (Docker Compose + Postgres Blindato)

Per un ambiente di produzione decente, lasciamo stare SQLite ed optiamo per un database relazionale serio come **PostgreSQL**, isolato in una rete privata non accessibile dall'esterno.

Come al solito, per mantenere l'ambiente pulito e facilmente backuppabile, usiamo Docker Compose con volumi relativi. A me i *named volumes* hanno sempre suscitato un grande ribrezzo: con i volumi relativi ci basta fare un bel tarball della cartella a servizi fermi e il backup è bello che servito.

Creiamo la directory per la nostra istanza di Forgejo:

```
mkdir -p /opt/forgejo
cd /opt/forgejo
```

Per evitare di lasciare password cablate nel file YAML, creiamo un file `.env` contenente il nome del progetto compose e una password robusta generata casualmente per il database:

```
COMPOSE_PROJECT_NAME=dreadful_forgejo
FORGEJO_DATABASE_PASSWORD="un_segreto_estremamente_lungo_e_complesso"
```

A questo punto, scriviamo il nostro `docker-compose.yml` definendo sia la rete isolata di backend per Postgres che la rete esterna per esporre Forgejo verso il reverse proxy (o il gateway Tor):

```
services:
  server:
    image: codeberg.org/forgejo/forgejo:15.0.3
    container_name: forgejo-server
    restart: always
    environment:
      - TZ=Europe/Rome
      - FORGEJO__database__DB_TYPE=postgres
      - FORGEJO__database__HOST=database:5432
      - FORGEJO__database__NAME=forgejo
      - FORGEJO__database__USER=forgejo
      - FORGEJO__database__PASSWD=${FORGEJO_DATABASE_PASSWORD}
    networks:
      - forgejo-backend # Rete isolata per parlare con il DB
      - forgejo_net     # Rete esterna per ricevere traffico da Tor/Proxy
    volumes:
      - ./data:/data
    ports:
      - 3205:3000
    depends_on:
      - database

  database:
    image: postgres:18
    container_name: forgejo-database
    restart: always
    environment:
      - POSTGRES_DB=forgejo
      - POSTGRES_USER=forgejo
      - POSTGRES_PASSWORD=${FORGEJO_DATABASE_PASSWORD}
```

```
    networks:
      - forgejo-backend # Il database vive SOLO qui dentro, zero clearnet
    volumes:
      - ./postgres:/var/lib/postgresql/data # Persistenza dei dati fisici di
Postgres

networks:
  # Rete di backend creata ad hoc e blindata
  forgejo-backend:
    internal: true

  # La rete esterna pre-esistente per il reverse proxy o il gateway Tor
  forgejo_net:
    external: true
```

L'occhio attento noterà due cose fondamentali:

1. Il database Postgres non ha porte esposte (`ports:`) ed è associato solo alla rete `forgejo-backend` marcata come `internal: true`. Questo garantisce isolamento totale da internet e prevenzione di leak.
2. Il volume di Postgres monta `./postgres:/var/lib/postgresql/data` (notare il `/data` finale) per garantire la persistenza corretta secondo le specifiche dell'immagine ufficiale.

Tiriamo su l'infrastruttura con il solito comando:

```
docker compose up -d
```

A questo punto, accedendo da browser su `http://[IP-DEL-SERVER]:3205`, possiamo completare la prima installazione (molto guidata ed intuitiva) e registrare il nostro utente amministratore.

Passo 2: Registrare il Runner nell'istanza

Prima di poter avviare il runner, dobbiamo dirgli come parlare con Forgejo. Accediamo all'interfaccia web come amministratore:

1. Andiamo in **Amministrazione del sito** -> **Azioni** -> **Runner**.
2. Clicchiamo su **Crea nuovo runner**.

3. La schermata ci mostrerà due informazioni vitali che non vedremo mai più: un **UUID** e un **Token** di registrazione, oltre a un blocco YAML pre-configurato. Copiamoci questi valori.

Passo 3: Configurare il Runner sull'host

Per far girare i comandi di deploy direttamente sulla macchina (ad esempio per spostare i file in `/var/www/` o lanciare comandi locali), installeremo `forgejo-runner` direttamente sull'host, facendolo girare sotto un utente dedicato privo di privilegi amministrativi.

1. Creazione dell'utente

```
sudo useradd -m -s /bin/bash forgejo-runner
sudo mkdir -p /home/forgejo-runner/.forgejo-runner
```

2. Generazione della configurazione

Scarichiamo l'eseguibile di `forgejo-runner` per la nostra architettura e copiamolo in `/usr/local/bin/forgejo-runner` rendendolo eseguibile. Dopodiché generiamo il file di configurazione di base:

```
sudo -u forgejo-runner forgejo-runner generate-config > /home/forgejo-runner/.forgejo-runner/config.yaml
```

3. Modifica del file `config.yaml`

Apriamo il file `/home/forgejo-runner/.forgejo-runner/config.yaml` e modifichiamo due blocchi fondamentali.

Il primo è il blocco `server`, dove incolleremo i dettagli presi dal pannello di Forgejo:

```
server:
  connections:
    mio-server-git:
      url: http://127.0.0.1:3205  # Indirizzo di Forgejo (locale o onion)
      uuid: 3a85d7d2-e97d-4069-a71c-e573adc5e31d  # Il nostro UUID
      token: dcbc6dc71365d4a74b2fa0e1d8f6c91652f15ad3 # Il nostro Token
```

```
      labels:
        - "ubuntu-latest:host"  # <--- CRITICO: dice al runner di eseguire i job direttamente sull'host
```

L'occhio attento noterà che abbiamo inserito `ubuntu-latest:host` sotto le labels della connessione. Di default, il runner proverebbe a lanciare i job dentro container Docker. Mappandolo come `:host`, quando scriveremo `runs-on: ubuntu-latest` nei nostri workflow, il runner eseguirà le istruzioni come semplici comandi bash sul server reale.

Sistemiamo la proprietà dei file:

```
sudo chown -R forgejo-runner:forgejo-runner /home/forgejo-runner/
```

Passo 4: Il servizio Systemd (e un po' di sano hardening)

Far girare comandi direttamente sull'host comporta dei rischi di sicurezza. Per mitigare la cosa, creiamo un servizio Systemd blindato tramite le funzionalità di hardening del kernel di Linux e controlli sull'allocazione delle risorse per evitare attacchi DoS o scansioni di rete non autorizzate.

Creiamo il file `/etc/systemd/system/forgejo-runner.service`:

```
[Unit]
Description=Forgejo Runner Service
After=network.target

[Service]
Type=simple
User=forgejo-runner
Group=forgejo-runner
WorkingDirectory=/home/forgejo-runner/.forgejo-runner
ExecStart=/usr/local/bin/forgejo-runner daemon --config /home/forgejo-runner/.forgejo-runner/config.yaml
Restart=always
RestartSec=10

# --- LIMITAZIONE RISORSE (Anti-DoS) ---
# Limita l'uso della CPU al 50% di un core (adatta in base alle nostre esigenze)
```

```
CPUQuota=50%
# Limita la RAM massima allocabile dal runner
MemoryMax=1G
# Limita il numero massimo di thread/processi (previene le fork-bomb nel
workflow)
TasksMax=100

# --- LIMITAZIONE RETE (Anti-PortScan/Leak) ---
# Consente solo connessioni a localhost (necessarie per parlare con Forgejo
locale)
IPAddressAllow=localhost
# Vieta qualsiasi altro tipo di traffico di rete (LAN, VPN, WAN)
IPAddressDeny=any

# --- HARDENING FILE SYSTEM & KERNEL ---
NoNewPrivileges=true
ProtectSystem=strict

# Diamo i permessi di scrittura SOLO alla cartella del runner e alla webroot
del sito
ReadWritePaths=/home/forgejo-runner/.forgejo-runner /var/www/mio-sito-
statico.it/

# Isoliamo la home degli altri utenti reali sul server
ProtectHome=tmp
InaccessiblePaths=/home/mio-utente-admin

# Limitiamo l'accesso all'hardware, ai cgroups e alle funzionalità del kernel
PrivateTmp=true
PrivateDevices=true
ProtectKernelTunables=true
ProtectControlGroups=true
ProtectKernelModules=true
RestrictRealtime=true
RestrictSUIDSGID=true

[Install]
WantedBy=multi-user.target
```

Abilitiamo e avviamo il servizio:

```
sudo systemctl daemon-reload
sudo systemctl enable --now forgejo-runner.service
```

Passo 5: Test del workflow (e una dritta salvavita)

Ora possiamo creare una pipeline di deploy creando il file `.forgejo/workflows/deploy.yml` dentro al nostro repository.

Se usiamo i runner in modalità `host` con percorsi assoluti, dobbiamo tenere a mente una cosa fondamentale: se un build fallisce prima del termine, la directory temporanea rimarrà sporca e bloccherà i successivi tentativi di clone. **Assicuratevi sempre di fare pulizia prima di clonare.**

Ecco un esempio di workflow robusto ed auto-riparante:

```
name: Deploy Website
on:
  push:
    branches:
      - main
  workflow_dispatch:

jobs:
  build-and-deploy:
    runs-on: ubuntu-latest
    steps:
      - name: Pulizia e Clone del repository
        run: |
          rm -rf /tmp/sito-sorgenti
          git clone http://127.0.0.1:3205/mio-utente/mio-sito.git /tmp/sito-sorgenti
          cd /tmp/sito-sorgenti

      - name: Compilazione con Zola
        run: |
          cd /tmp/sito-sorgenti
          zola build

      - name: Deploy in produzione
        run: |
          # Svuota la webroot e copia i nuovi file generati da Zola
          rm -rf /var/www/mio-sito-statico.it/*
          cp -r /tmp/sito-sorgenti/public/* /var/www/mio-sito-statico.it/
```

```
          # Pulizia finale
          rm -rf /tmp/sito-sorgenti
```

Ora ci basta fare un push su `main` o avviare il workflow a mano dall'interfaccia web per vedere il nostro sito compilarsi e pubblicarsi in produzione a velocità luce e in totale sicurezza.

Fluxer Icon

Canale di discussione e supporto

Canale di discussione e supporto. Per confrontarti su configurazioni o altro entra su fluxer.dreadful.work (https://fluxer.dreadful.work/invite/dreadful).