Doc history

Rev	Date	Author	Description
00	11/02/2025	Silvio Mario Pastori	Configurazione Minima per una Cloud Privata con Kubernetes e GitLab
01	12/02/2025	Silvio Mario Pastori	Integrazione tra i PC
02	13/02/2025	Silvio Mario Pastori	Quale distribuzione Linux per lo sviluppo usare?
03	14/02/2025	Silvio Mario Pastori	Schema Architetturale
04	15/02/2025	Silvio Mario Pastori	Ecosistema di Strumenti per Cloud, Container e DevOps
05	26/02/2025	Marco Selva	Review and link

Linked doc

Cod	Link	Name	Description
A1	link	[A1] - Hosting and Cloud project	Project main doc
A3	link	[A3] - Energy and cost simulation	Excel with energy and cost estimation and PC selection

Configurazione Minima per una Cloud Privata con Kubernetes e GitLab

Per creare un'infrastruttura efficiente e affidabile, il numero minimo di PC dipende dal livello di tolleranza ai guasti e dalle funzionalità richieste. Di seguito una configurazione minima e una gerarchia dettagliata.

1. Numero Minimo di PC

Ruolo	Numero	Specifiche consigliate
Master Node	1	CPU 4 core, RAM 8 GB, 100 GB SSD
Worker Nodes	2	CPU 2 core, RAM 4-8 GB, 100 GB SSD
Backup Node (opzionale)	1	CPU 2 core, RAM 4 GB, 100 GB HDD
GitLab Node	1	CPU 4 core, RAM 8 GB, 200 GB SSD

Totale: 4 PC (Minimo raccomandato)
Si può partire con 3 PC (1 master + 2 worker), ma per servizi complessi come GitLab e backup offsite ne serviranno almeno 5.

2. Gerarchia dei Ruoli

Master Node (Control Plane)

├── Scheduler

├── Controller Manager

└── API Server

│

├── **Worker Node** 1 (Pods per app)

├── **Worker Node** 2 (GitLab Runner Pods)

└── Persistent Storage (Ceph/NFS/Local PV)

Backup Node (Velero o MinIO)

Ruoli Dettagliati

- **Master Node:**
  Gestisce l'intero cluster Kubernetes, pianifica i pod e risponde alle richieste degli utenti.
- **Worker Nodes:**
  Eseguono i pod delle applicazioni e servizi come GitLab Runners o microservizi personalizzati.
- **GitLab Node (separato o condiviso su un worker):**
  Ospita GitLab per gestione del codice e integrazione continua.
- **Backup Node:**
  Memorizza copie di backup dei dati su cloud locale (es. MinIO) o remoto.

PC 1 - Master Node (Control Plane)

Ruolo: Gestisce il cluster Kubernetes, pianifica i pod e controlla lo stato dei worker.

Specifiche consigliate:

- CPU: 4 Core
- RAM: 8 GB
- Storage: 100 GB SSD

Software da installare:

1. **Kubernetes Control Plane:**

   ```
   sudo apt update

   sudo apt install kubeadm kubectl kubelet

   sudo kubeadm init --pod-network-cidr=192.168.0.0/16
   ```

2. **Network Plugin:**

   Installazione di Calico come rete overlay:

   ```
   kubectl apply -f https://docs.projectcalico.org/v3.14/manifests/calico.yaml
   ```

3. **Ingress Controller:**

   ```
   kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/main/deploy/static/provider/cloud/deploy.yaml
   ```

PC 2 - Worker Node 1 (App & Microservizi)

Ruolo: Esegue i pod delle applicazioni e microservizi personalizzati.

Specifiche consigliate:

- CPU: 2 Core
- RAM: 4 GB

- Storage: 100 GB SSD

Software da installare:

1. **Kubernetes Worker Components:**

   ```
   sudo apt update

   sudo apt install kubeadm kubelet

   sudo kubeadm join <master-ip>:6443 --token <token> --discovery-token-ca-cert-hash sha256:<hash>
   ```

2. **Docker Engine:**

   Kubernetes richiede un container runtime:

   ```
   sudo apt install docker.io

   sudo systemctl enable docker
   ```

PC 3 - Worker Node 2 (GitLab Runner + CI/CD)

Ruolo: Esegue i pod per GitLab Runner, destinati alla Continuous Integration (CI/CD).

Specifiche consigliate:

- CPU: 4 Core
- RAM: 8 GB
- Storage: 200 GB SSD

Software da installare:

1. **Kubernetes Worker Components:**

   ```
   sudo apt update

   sudo apt install kubeadm kubelet

   sudo kubeadm join <master-ip>:6443 --token <token> --discovery-token-ca-cert-hash sha256:<hash>
   ```

2. **GitLab Runner:**

   Installa il runner per GitLab:

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

   sudo apt install gitlab-runner
   ```

3. **Configura GitLab Runner:**

Collega il runner al tuo GitLab:

```
sudo gitlab-runner register
```

PC 4 - Storage & Backup Node

Ruolo: Gestione dei backup persistenti dei dati del cluster e dello storage.

Specifiche consigliate:

- CPU: 2 Core
- RAM: 4 GB
- Storage: 200 GB HDD o SSD

Software da installare:

1. **NFS Server:**

   Configura lo storage condiviso tra i nodi:

   ```
   sudo apt install nfs-kernel-server
   sudo mkdir -p /data/nfs
   sudo chown nobody:nogroup /data/nfs
   echo "/data/nfs *(rw,sync,no_subtree_check)" | sudo tee -a /etc/exports
   sudo exportfs -a
   ```

2. **MinIO (Backup Object Storage)**

   Esegui MinIO come servizio di storage:

   ```
   wget https://dl.min.io/server/minio/release/linux-amd64/minio
   chmod +x minio
   ./minio server /data/nfs
   ```

3. **Velero (Backup Kubernetes)**
   Installa Velero per i backup:

   ```
   velero install --provider aws --bucket cluster-backups --use-restic
   ```

Specifiche del comando kubeadm join

sintassi completa

sudo kubeadm join <master-ip>:6443 --token <token> --discovery-token-ca-cert-hash sha256:<hash>

Spiegazione del comando e dei parametri

Parametro	Descrizione
sudo	**Esegue il comando con privilegi di amministratore.**
kubeadm join	**Comando di Kubeadm per unire il nodo al cluster Kubernetes esistente.**
<master-ip>:6443	**L'indirizzo IP del control plane (master node) e la porta 6443, che è la porta predefinita per l'API Server di Kubernetes.**
--token <token>	**Il token di autenticazione generato dal comando kubeadm init sul master. Serve per consentire al worker di unirsi al cluster.**
--discovery-token-ca-cert-hash sha256:<hash>	**Il valore hash SHA-256 del certificato CA del master, utilizzato per verificare l'autenticità del cluster ed evitare attacchi man-in-the-middle.**

Dove trovo i valori per <token> e <hash>?

Dopo aver inizializzato il cluster Kubernetes sul master node con:

```
sudo kubeadm init
```

si otterrà un messaggio simile a questo:

```
kubeadm join 192.168.1.100:6443 --token abc123.abcdef1234567890 \
    --discovery-token-ca-cert-hash sha256:6f8c930d1...xyz
```

dove si trova il token e l'hash da copiare sui worker node!

Come rigenerare il token se è scaduto?

I token in Kubernetes hanno una scadenza (di default 24 ore). Se il token non è più valido, si può generarne uno nuovo con:

```
sudo kubeadm token create --print-join-command
```

Si otterrà il comando completo da eseguire sui worker.

Esempio pratico

Se il **master node** ha IP 192.168.1.100 e il token generato è abc123.abcdef1234567890, il comando sul worker sarà:

```
sudo kubeadm join 192.168.1.100:6443 --token abc123.abcdef1234567890 \
    --discovery-token-ca-cert-hash sha256:6f8c930d1abcdef1234567890xyz
```

Dopo aver eseguito questo comando su tutti i worker, si può verificare che siano connessi al cluster eseguendo sul master:

```
kubectl get nodes
```

Integrazione tra i PC

- **Networking:** Tutti i PC devono comunicare tra loro sulla stessa rete.
- **Persistent Volume (NFS):** Condividere lo storage per backup e persistenza dei dati dei pod.
- **Certificati SSL:** Usare cert-manager per proteggere le comunicazioni.
- **GitLab Backup:** Configurare MinIO come backend per i backup GitLab.

Configurazione rete subnet Linux

Script di configurazione automatica:

Bisogna creare uno script bash per configurare automaticamente gli indirizzi IP e il gateway. Questo è un esempio di script che configura un indirizzo IP statico:

Esempio di script bash

Crea un file di script, ad esempio configura_rete.sh:

```
#!/bin/bash

# Configurazione della rete
INTERFACCIA="eth0"
IPADDR="192.168.1.2"
NETMASK="255.255.255.0"
GATEWAY="192.168.1.1"

# Configura l'indirizzo IP
sudo ip addr add $IPADDR/$NETMASK dev $INTERFACCIA
sudo ip link set dev $INTERFACCIA up

# Configura il gateway
sudo ip route add default via $GATEWAY

# Verifica la configurazione
ip a
```

```
ping -c 4 192.168.1.3
```

Per rendere lo script eseguibile:

```
chmod +x configura_rete.sh
```

Per esegui lo script:

```
./configura_rete.sh
```

Automatizzare la configurazione per più PC

Per configurare più PC, copiare lo script su ogni macchina e modificare l'indirizzo IP e il gateway, adattandoli per ciascun PC, o usare uno script centralizzato tramite SSH per configurare tutte le macchine.

Esempio di utilizzo di SSH per eseguire lo script su più PC

Se si ha accesso SSH a ciascun PC, si può eseguire lo script da una macchina centrale:

```
ssh user@192.168.1.2 'bash -s' < configura_rete.sh
```

Script: configura_rete.sh

Lo scopo di questo script è configurare la rete di un PC assegnandogli un indirizzo IP statico, una subnet mask e un gateway, nonché testare la connettività di rete tra i PC.

Questa la spiegazione di ogni parte dello script.

```
#!/bin/bash
```

- Questa riga è chiamata **shebang**. Indica al sistema che questo file deve essere eseguito utilizzando bash (il Bourne Again Shell), che è uno degli interpreti di shell più comuni su Linux.

```
# Configurazione della rete
INTERFACCIA="eth0"
IPADDR="192.168.1.2"
NETMASK="255.255.255.0"
GATEWAY="192.168.1.1"
```

- Qui vengono dichiarate delle **variabili** che contengono i parametri necessari per configurare la rete:

- INTERFACCIA="eth0": Il nome dell'interfaccia di rete a cui si vuole assegnare l'indirizzo IP. Questo potrebbe essere diverso su ogni pc (ad esempio enp3s0 o wlp2s0 per Wi-Fi).
- IPADDR="192.168.1.2": L'indirizzo IP che si vuole assegnare al PC.
- NETMASK="255.255.255.0": La subnet mask, che in questo caso è una classica subnet /24. Significa che i primi 24 bit dell'indirizzo IP sono utilizzati per identificare la rete e il resto per i dispositivi nella rete.
- GATEWAY="192.168.1.1": L'indirizzo IP del **gateway**, solitamente il router che permette la comunicazione con altre reti (ad esempio Internet).

Questi valori sono solo esempi, e vanno modificati in base alla configurazione di rete specifica.

Configura l'indirizzo IP

sudo ip addr add $IPADDR/$NETMASK dev $INTERFACCIA

- **Comando ip addr add**: Questo comando configura l'indirizzo IP sull' interfaccia di rete del PC.
  - $IPADDR è il valore dell'indirizzo IP che è stato dichiarato precedentemente (ad esempio 192.168.1.2).
  - $NETMASK è la subnet mask (ad esempio 255.255.255.0).
  - $INTERFACCIA è il nome dell'interfaccia di rete (ad esempio eth0).
  - Il comando ip addr add aggiunge l'indirizzo IP specificato all'interfaccia di rete indicata.

Ad esempio, se la variabile IPADDR è 192.168.1.2 e la variabile NETMASK è 255.255.255.0, il comando risultante sarà

sudo ip addr add 192.168.1.2/24 dev eth0

Attivazione interfaccia di rete

sudo ip link set dev $INTERFACCIA up

Comando ip link set: Questo comando attiva (o "accende") l'interfaccia di rete. Se l'interfaccia è già attiva, non succederà nulla.

- $INTERFACCIA è il nome dell'interfaccia di rete.
- Il comando ip link set dev eth0 up imposta l'interfaccia eth0 come attiva. Questo è necessario per permettere il traffico di rete.

Configura il gateway

```
sudo ip route add default via $GATEWAY
```

Comando ip route add: Imposta il **gateway** predefinito, che è il dispositivo di rete (tipicamente un router) a cui il PC si rivolge per raggiungere altre reti (come Internet).

- $GATEWAY è l'indirizzo IP del gateway (ad esempio 192.168.1.1).
- sudo ip route add default via 192.168.1.1 aggiunge una rotta predefinita che invia il traffico non destinato alla rete locale al gateway.

```
# Verifica la configurazione
ip a
```

Comando ip a: Questo comando mostra tutte le interfacce di rete e le configurazioni IP associate. Puoi usarlo per verificare se l'indirizzo IP è stato configurato correttamente sull'interfaccia di rete.

```
# Verifica connettività

ping -c 4 192.168.1.3
```

Comando ping: Questo comando verifica la connettività di rete tra il PC corrente e un altro dispositivo della rete (in questo caso, l'indirizzo IP 192.168.1.3).

- -c 4 indica al comando di inviare solo 4 pacchetti ICMP.
- Puoi cambiare l'indirizzo IP per testare la comunicazione con un altro PC della rete.
- Il comando ping aiuta a verificare se il PC può raggiungere altri dispositivi nella rete.

Funzionamento complessivo dello script

Lo script esegue i seguenti passaggi:

1. Imposta variabili per l'indirizzo IP, la subnet mask e il gateway.
2. Configura l'indirizzo IP e la subnet mask sulla tua interfaccia di rete.
3. Attiva l'interfaccia di rete.
4. Imposta il gateway di rete.
5. Verifica che l'indirizzo IP sia stato correttamente configurato utilizzando il comando ip a.
6. Verifica la connettività con un altro PC della rete utilizzando il comando ping.

Questo script permette di configurare rapidamente la rete su un PC con indirizzo IP statico, ed è facilmente adattabile per più PC, cambiando solo i valori di indirizzo IP e gateway.

Quale distribuzione Linux per lo sviluppo usare?

La scelta della distribuzione Linux per lo sviluppo dipende dal tipo di progetti su cui lavori, dalle risorse hardware disponibili e dalle preferenze personali. Ecco un confronto delle migliori distribuzioni per lo sviluppo:

1. Ubuntu (GNOME) – Scelta Generale e Affidabile

Perché sceglierlo?

- Ampia comunità di supporto e documentazione.
- Compatibile con la maggior parte degli strumenti di sviluppo.
- Ottima integrazione con Docker, Kubernetes e strumenti cloud.
- Facile da installare e configurare.

Quando usarlo?

- Sviluppo web, backend, frontend.
- Applicazioni cloud e containerizzate (Docker, Kubernetes).
- Sviluppo software generico (Python, C++, Java, Go, etc.).

Requisiti minimi: 4 GB di RAM, 25 GB di spazio su disco.

2. Ubuntu MATE – Leggero e Stabile

Perché sceglierlo?

- Ambiente desktop MATE leggero, ideale per PC meno potenti.
- Stessa compatibilità software di Ubuntu standard.
- Performance migliori su hardware datato.

Quando usarlo?

- Se preferisci un sistema leggero con meno consumo di RAM e CPU.
- Per lo sviluppo su macchine virtuali o hardware più vecchio.

Requisiti minimi: 2 GB di RAM, 10 GB di spazio su disco.

3. Fedora – Per gli Sviluppatori più Aggiornati

Perché sceglierlo?

- Sempre aggiornato con le ultime versioni di software e librerie.
- Ottimo per sviluppatori che usano container, grazie all'integrazione con **Podman e Kubernetes**.
- Stabile e usato da Red Hat per testing delle tecnologie aziendali.

Quando usarlo?

- Se vuoi sempre le ultime versioni di compiler e tool di sviluppo.
- Per lo sviluppo **enterprise** (vicino a Red Hat Enterprise Linux).

Requisiti minimi: 4 GB di RAM, 20 GB di spazio su disco.

4. Arch Linux – Per Sviluppatori Esperti

Perché sceglierlo?

- **Massima personalizzazione**, installi solo ciò che ti serve.
- Pacchetti sempre aggiornati (Rolling Release).
- AUR (Arch User Repository) con moltissime librerie e strumenti.

Quando usarlo?

- Se vuoi il pieno controllo del sistema.
- Se non ti spaventa la configurazione manuale.

Requisiti minimi: 2 GB di RAM, 20 GB di spazio su disco.

5. Debian – Stabilità e Affidabilità

Perché sceglierlo?

- Estremamente stabile, adatto a server e ambienti di produzione.
- Ottimo per chi sviluppa software a lungo termine.
- Versioni più conservative rispetto a Ubuntu/Fedora.

Quando usarlo?

- Se vuoi un sistema solido e testato.
- Per lo sviluppo su sistemi embedded o mission-critical.

Requisiti minimi: 2 GB di RAM, 10 GB di spazio su disco.

6. openSUSE – Per DevOps e Sviluppo Enterprise

Perché sceglierlo?

- Strumento **YaST** per gestione avanzata del sistema.
- Supporta Kubernetes, Docker e strumenti enterprise.
- Versione **Leap** (stabile) o **Tumbleweed** (rolling release).

Quando usarlo?

- Se vuoi un sistema gestibile e robusto per ambienti professionali.
- Se lavori in DevOps o sviluppo enterprise.

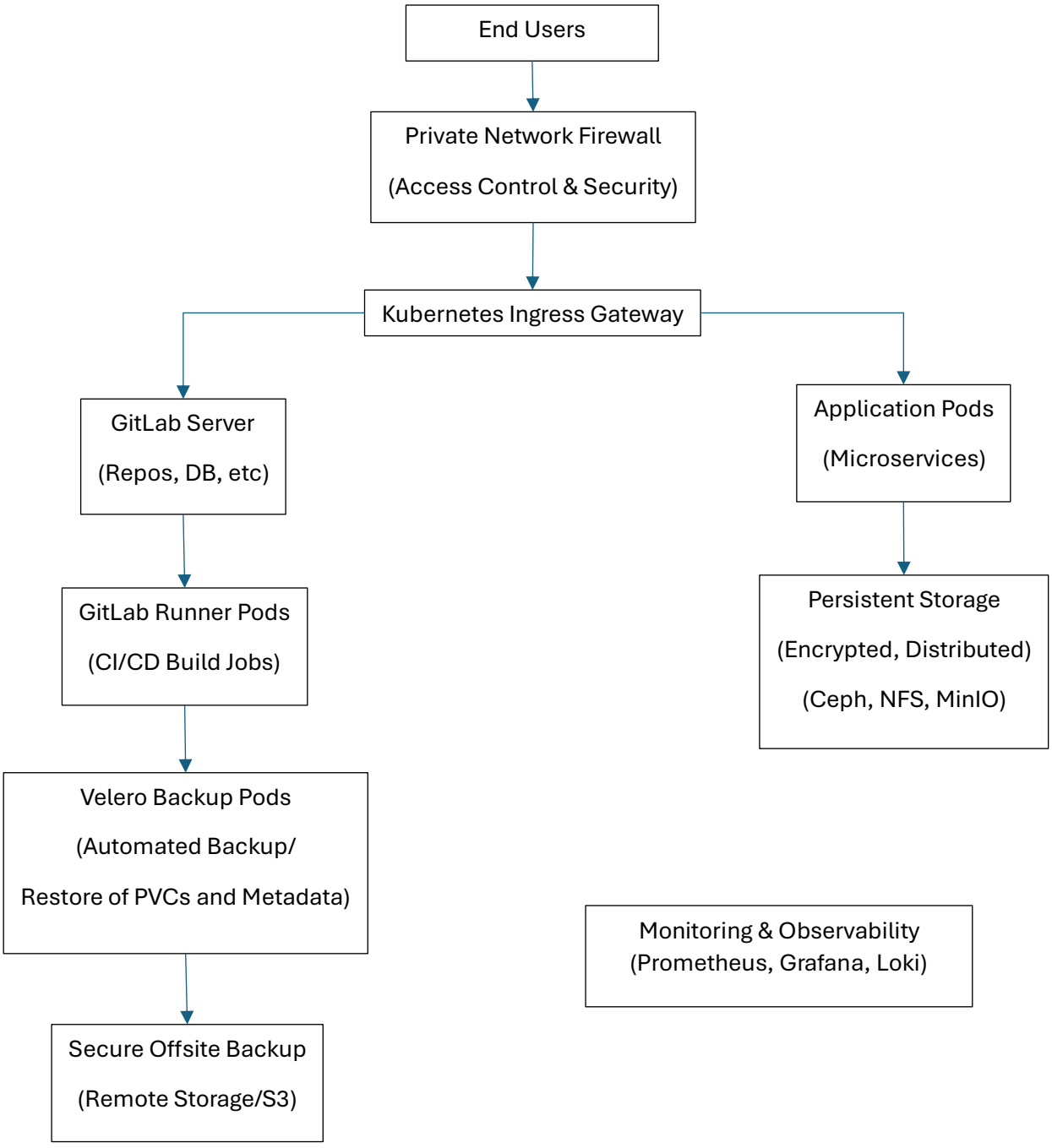
Requisiti minimi: 4 GB di RAM, 25 GB di spazio su disco.

Conclusione: Quale Distribuzione Scegliere?

Tipo di Sviluppo	Distribuzione Consigliata
Generale e facile da usare	Ubuntu (GNOME)
PC vecchi o leggeri	Ubuntu MATE, Debian
Ultime tecnologie e sviluppo cloud	Fedora, openSUSE
Esperti e personalizzazione totale	Arch Linux
Ambienti enterprise e server	Debian, openSUSE Leap

Se si è principianti o si vuole un sistema pronto all'uso, **Ubuntu è la scelta migliore**. Se si vuole aggiornamenti frequenti, Fedora è ottima. Per il massimo controllo, provare Arch Linux.

Schema Architetturale



Dettagli Chiave

1. End Users (Utenti Finali)

- Gli utenti finali accedono alle applicazioni o al sistema GitLab tramite un'interfaccia web.
- Gli accessi sono filtrati e gestiti tramite un firewall per garantire la sicurezza delle connessioni.
- Tipicamente i protocolli usati sono **HTTPS** per comunicazioni crittografate.

2. Private Network Firewall (Firewall di Rete Privato)

- Limita l'accesso solo agli indirizzi IP autorizzati, bloccando attacchi esterni.
- Configurabile con strumenti come **iptables**, **UFW**, o firewall hardware.
- Si può integrare una **VPN** per connettere utenti e sviluppatori autorizzati alla cloud privata.

3. Kubernetes Ingress Gateway

- Un punto centrale per la gestione del traffico verso i servizi Kubernetes, come GitLab o microservizi applicativi.
- Utilizza controller come **NGINX Ingress Controller** o **Traefik** per bilanciare il carico e applicare regole di routing.
- Configurato per supportare certificati **TLS** per comunicazioni sicure.

4. GitLab Server (Repos, Configs, Database)

- Gestisce repository Git, configurazioni di progetto e pipeline CI/CD.
- Archivia i dati su **Persistent Volume Claims (PVC)** per garantire la persistenza dei dati in caso di riavvii o migrazioni dei Pod.
- Richiede la configurazione di backup sicuri dei seguenti dati:
    - Database PostgreSQL
    - Configurazioni GitLab
    - Repository utente

5. Application Pods (Microservizi)

- Contengono le applicazioni o servizi backend, orchestrati da Kubernetes.
- Ogni pod ha il proprio volume persistente per memorizzare dati necessari (ad esempio, log o configurazioni temporanee).
- Possono essere replicati per garantire alta disponibilità.

6. Persistent Storage (Encrypted, Distributed)

- I dati dei Pod (incluso GitLab) sono salvati su un backend di storage distribuito e crittografato come:
    - **Ceph:** Storage distribuito ad alte prestazioni.
    - **NFS:** File system condiviso semplice da configurare.
    - **MinIO:** Compatibile con S3 per oggetti storage.
- La crittografia garantisce la protezione dei dati sia a riposo che in transito.

7. GitLab Runner Pods (CI/CD Build Jobs)

- Pods dedicati all'esecuzione delle pipeline CI/CD definite su GitLab.
- Possono scalare automaticamente in base al carico.
- Eseguono compilazioni, test e deployment dei progetti Git.

8. Velero Backup Pods (Backup/Restore PVC)

- Pods specializzati per effettuare backup e restore dei dati Kubernetes (PVC e configurazioni).
- Automatizzano la creazione di snapshot dei dati persistenti, essenziali in caso di guasti o incidenti.
- Velero può inviare i backup a storage remoti (S3, Google Cloud Storage, etc.).

9. Secure Offsite Backup (Archiviazione Remota)

- Backup dei dati in una posizione remota sicura.
- Può utilizzare:
    - **S3 compatibile** (come MinIO su server remoti)
    - **Soluzioni cloud private**
- La crittografia garantisce che i dati siano sicuri anche in caso di compromissione della rete.

10. Monitoring & Observability (Prometheus, Grafana, Loki)

- **Prometheus:** Raccoglie metriche di sistema e applicazioni.
- **Grafana:** Visualizzazione delle metriche tramite dashboard personalizzabili.
- **Loki:** Gestione centralizzata dei log per il debugging e il monitoraggio delle applicazioni.

Conclusione

Questa architettura offre una soluzione robusta e sicura per una cloud privata, garantendo resilienza, backup dei dati e strumenti di osservabilità.

Ecosistema di Strumenti per Cloud, Container e DevOps

Container & Orchestration

- **Docker** (Open Source in parte) → Piattaforma per la creazione, gestione ed esecuzione di container.
  https://www.docker.com
- **Kubernetes** (Open Source) → Sistema di orchestrazione di container per la gestione scalabile di applicazioni.
  https://kubernetes.io

Web Server & Load Balancer

- **NGINX** (Open Source) → Server web e reverse proxy, usato anche come bilanciatore di carico.
  https://nginx.org

Storage & File System

- **PVC (Persistent Volume Claim)** → Oggetto di Kubernetes che richiede spazio di archiviazione da un Persistent Volume.
  https://kubernetes.io/docs/concepts/storage/persistent-volumes/
- **Ceph** (Open Source) → Sistema di storage distribuito per object storage, block storage e file storage.
  https://ceph.io
- **NFS (Network File System)** (Open Source) → Protocollo per la condivisione di file su rete.
  https://en.wikipedia.org/wiki/Network_File_System
- **MinIO** (Open Source) → Object storage compatibile con S3, ottimizzato per Kubernetes.
  https://min.io
- **S3 (Amazon Simple Storage Service)** (Non Open Source) → Servizio di object storage di AWS.
  https://aws.amazon.com/s3/
- **Velero** (Open Source) → Strumento di backup e disaster recovery per cluster Kubernetes.
  https://velero.io

DevOps & CI/CD

- **GitLab** (Open Source in parte) → Piattaforma DevOps con gestione del codice, CI/CD e deploy automation.
  https://gitlab.com

Monitoring & Logging

- **Prometheus** (Open Source) → Sistema di monitoraggio e alerting per metriche time-series.
  https://prometheus.io
- **Grafana** (Open Source) → Strumento di visualizzazione e analisi dati, spesso usato con Prometheus.
  https://grafana.com
- **Loki** (Open Source) → Sistema di logging scalabile, simile a Prometheus ma per log.
  https://grafana.com/oss/loki/