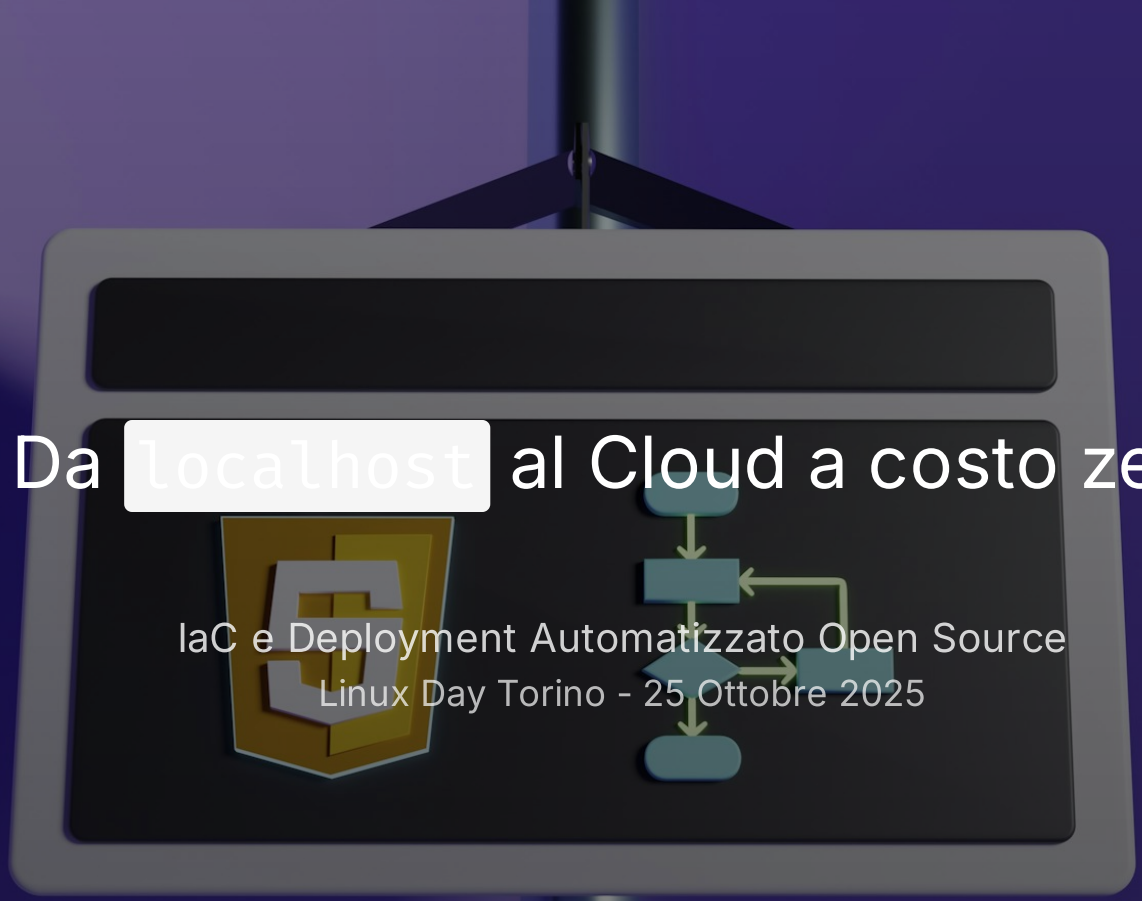
Da `localhost` al Cloud a costo zero



IaC e Deployment Automatizzato Open Source

Linux Day Torino - 25 Ottobre 2025

Who am I?

Leonardo Poggiani

GitLab: @leopoggiani

GitHub: @leonardopoggiani

Da poco sono diventato **contributor** della repository del Linux Day

Il problema che ho incontrato

- Ho iniziato a contribuire al progetto Linux Day
- Ogni volta che faccio una modifica... come fanno i reviewer a verificare?
- "Fidati, funziona sul mio laptop" non è una bella frase da dire
- Serve un modo per far vedere **dal vivo** cosa ho fatto
- Senza chiedere ai reviewer di fare setup locali complicati

> 💡 **L'idea**: ogni branch = un ambiente live accessibile via URL

L’obiettivo

- Costruire una **pipeline CI/CD completa** seguendo i principi DevOps
- Usare **solo strumenti open source** (zero vendor lock-in)
- Mantenere i **costi contenuti** (~€15/anno, non €500/anno)
- Ogni componente deve essere **sostituibile**
- Evitare servizi che ti "intrappolano" con prezzi esagerati

> 🎯 **Il vero obiettivo**: indipendenza tecnologica senza spendere una fortuna

Stack tecnologico

OpenTofu

La tua infrastruttura in codice

k3s

Kubernetes leggero per ARM64

Traefik

Ingress controller con TLS automatico

Podman

Containerizzazione rootless

GitLab CI/CD

Deploy automatico ad ogni push

Oracle Cloud

Free Tier (sempre gratis)

I costi reali

💰 Cosa costa

- Dominio: €10-15/anno
  - *Alternative gratis*: afraid.org, duckdns.org
  - **Setup DNS**: `demo.poggiolab.net` + `*.demo.poggiolab.net`
- Oracle Cloud: carta richiesta
- Tempo: setup iniziale

✅ Cosa è gratis

- Software open source
- Compute e storage
- Certificati TLS
- CI/CD illimitato
- 10TB bandwidth/mese

Architettura generale

Dal push del codice all’ambiente live in 2-3 minuti

Oracle Cloud Free Tier

✅ Vantaggi

- Always Free (non scade mai)
- 4 OCPU ARM + 24GB RAM
- 200GB storage
- 10TB bandwidth/mese

⚠️ Rischi

- Carta di credito obbligatoria
- Chiusura account arbitraria
- Region sovraccariche
- VM idle >7gg terminate

💡 **Mitigazione**: Stack portabile → migrazione in 30 minuti

OpenTofu vs Terraform

Terraform

- ❌ Licenza BSL (proprietaria)
- ⚠️ Controllo HashiCorp
- 💰 Lock-in commerciale
- ✅ Matura e stabile

OpenTofu

- ✅ 100% Open Source (MPL)
- 🏛️ Linux Foundation
- 🔄 100% compatibile
- 🚀 Community-driven
- ✅ Garanzia futura

Drop-in replacement: rinomini il binario e sei già migrato

Workflow OpenTofu

```
# Inizializzazione
tofu init

# Anteprima modifiche
tofu plan

# Applicazione modifiche
tofu apply
```

OpenTofu nel progetto: struttura files

```
main.tf              # Provider OCI, data sources
network.tf           # VCN, subnet, security lists
compute.tf           # VM instance con cloud-init
variables.tf         # Input con validazioni
outputs.tf           # IP pubblico, info cluster
cloud-init.yml       # Script inizializzazione
terraform.tfvars     # I tuoi valori (non in git!)
```

📂 Directory: `examples/infrastructure/`

OpenTofu: esempio di risorsa OCI

```hcl
# Crea una VM Oracle Cloud
resource "oci_core_instance" "k3s_node" {
  compartment_id = var.compartment_id

  # Shape free tier ARM64
  shape = "VM.Standard.A1.Flex"
  shape_config {
    ocpus         = 4
    memory_in_gbs = 24
  }

  # Cloud-init per setup automatico
  metadata = {
    user_data = base64encode(
      templatefile("${path.module}/cloud-init.yml", {
        hostname             = "k3s-master"
        letsencrypt_email    = var.letsencrypt_email
        gitlab_runner_token  = var.gitlab_runner_token
      })
    )
  }
}
```

💡 `templatefile()` sostituisce variabili nel cloud-init

OpenTofu: variabili con validazione

```
variable "instance_ocpus" {
  description = "Numero di OCPUs (max 4 per free tier)"
  type        = number
  default     = 4

  validation {
    condition     = var.instance_ocpus >= 1 && var.instance_ocpus <= 4
    error_message = "Free tier ARM: max 4 OCPUs"
  }
}

variable "gitlab_runner_token" {
  description = "Token registrazione GitLab Runner"
  type        = string
  sensitive   = true  # Non viene mostrato nei log
}
```

✅ Le validazioni prevengono errori costosi (fuori dal free tier)

Podman vs Docker

- Architettura daemonless
- Esecuzione rootless
- Maggiore sicurezza
- Compatibile con Dockerfile/Containerfile
- Drop-in replacement di Docker

Podman: comandi base (identici a Docker!)

```
# Build immagine
podman build -f docker/Containerfile -t website:latest .

# Run container
podman run -p 8080:80 website:latest

# Lista container
podman ps

# Logs
podman logs <container-id>
```

💡 Sostituisci `docker` con `podman` e funziona tutto uguale!

Containerfile del progetto

```
# Base image leggera
FROM caddy:2.7-alpine

# Copia configurazione Caddy
COPY docker/Caddyfile /etc/caddy/Caddyfile

# Copia sito statico
COPY website/ /usr/share/caddy/
RUN chmod -R 755 /usr/share/caddy
EXPOSE 80

# Health check
HEALTHCHECK --interval=30s --timeout=3s \
  CMD wget --no-verbose --tries=1 --spider \
      http://localhost || exit 1
```

📁 File: `docker/Containerfile` (compatibile Dockerfile/Podman)

Caddy: web server del progetto

```
# Configurazione Caddy (docker/Caddyfile)
:80 {
    root * /usr/share/caddy

    # Compressione automatica
    encode gzip

    # Headers di sicurezza
    header / {
        X-Content-Type-Options "nosniff"
        X-Frame-Options "DENY"
        X-XSS-Protection "1; mode=block"
        Referrer-Policy "strict-origin-when-cross-origin"
    }

    file_server
    handle_errors {
        @404 { expression {http.error.status_code} == 404 }
        rewrite @404 /404.html
        file_server
    }
}
```

k3s: Kubernetes leggero

- Kubernetes production-ready in <100MB
- Ottimizzato per ARM64
- Installazione con un comando
- Ideale per single-node
- API compatibile 100% con Kubernetes

k3s: installazione (nel cloud-init)

```
# Installazione k3s con un comando
curl -sfL https://get.k3s.io | sh -s - \
  --tls-san ${public_ip} \
  --disable traefik --disable servicelb

# k3s è già attivo!
export KUBECONFIG=/etc/rancher/k3s/k3s.yaml

# Verifica
kubectl get nodes
```

💡 `--tls-san` aggiunge l'IP pubblico al certificato TLS del cluster

k3s: comandi utili

```
# Lista nodi
kubectl get nodes

# Lista pod in tutti i namespace
kubectl get pods -A

# Stato deployment in production
kubectl get deployments -n production

# Logs applicazione
kubectl logs -f deployment/website -n production
```

🔍 Stesso comando `kubectl` di Kubernetes full, 100% compatibile

Kubernetes: deployment manifest del progetto

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: website
  namespace: production
spec:
  replicas: 2
  template:
    spec:
      containers:
      - name: website
        image: ${CI_REGISTRY_IMAGE}:${CI_COMMIT_SHA}
        resources:
          limits: { memory: "128Mi", cpu: "200m" }
          requests: { memory: "64Mi", cpu: "100m" }
---
apiVersion: v1
kind: Service
metadata:
  name: website
  namespace: production
spec:
  selector: { app: website }
  ports: [{ port: 80 }]
```

Traefik: routing e HTTPS automatico

- Ingress controller nativo per Kubernetes
- Integrazione Let’s Encrypt automatica
- Routing basato su hostname
- Zero configurazione certificati
- Rinnovo automatico TLS

Traefik: installazione via Helm (nel cloud-init)

```
# Installa Traefik Ingress Controller
helm repo add traefik https://traefik.github.io/charts
helm repo update

# Deploy con Let's Encrypt configurato
helm install traefik traefik/traefik \
  --namespace kube-system \
  --set ports.web.port=80 \
  --set ports.websecure.port=443 \
  --set ports.websecure.tls.enabled=true \
  --set certificatesResolvers.letsencrypt.acme.email=${letsencrypt_email}
```

🔐 Let’s Encrypt configurato automaticamente al provisioning

Ingress: come esporre un servizio con HTTPS

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: website
  namespace: production
  annotations:
    # Abilita HTTPS e Let's Encrypt
    traefik.ingress.kubernetes.io/router.entrypoints: websecure
    traefik.ingress.kubernetes.io/router.tls: "true"
    traefik.ingress.kubernetes.io/router.tls.certresolver: letsencrypt
spec:
  rules:
  - host: ${DOMAIN}  # es: feature-xyz.demo.poggiolab.net
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service: { name: website, port: { number: 80 } }
```

Traefik: workflow automatico

1. **Rileva Ingress** → monitora Kubernetes
2. **Configura routing** → regole per hostname
3. **Ottiene certificato** → Let’s Encrypt
4. **HTTPS live** → rinnovo automatico

⚡ Da Ingress a HTTPS in ~60 secondi

GitLab Runner

- Self-hosted sulla nostra infrastruttura
- Pieno controllo delle risorse
- Nessun limite di minuti
- Esecuzione pipeline personalizzate

GitLab Runner: registrazione (nel cloud-init)

```
# Registra runner con Kubernetes executor
gitlab-runner register \
  --executor "kubernetes" \
  --kubernetes-host "https://127.0.0.1:6443" \
  --kubernetes-namespace "gitlab-runner" \
  --tag-list "k3s,arm64,docker"

# I job CI/CD diventano pod temporanei!
# Build e deploy avvengono come Kubernetes Jobs
# nella stessa infrastruttura
```

🚀 Runner self-hosted = zero costi, nessun limite di minuti

Pipeline GitLab CI/CD: struttura base

```yaml
# .gitlab-ci.yml
stages: [build, deploy]

# Stage 1: Build container
build:
  stage: build
  tags: [k3s, arm64]
  script:
    - docker build -t $CI_REGISTRY_IMAGE:$CI_COMMIT_SHA .
    - docker push $CI_REGISTRY_IMAGE:$CI_COMMIT_SHA
    - docker push $CI_REGISTRY_IMAGE:latest

# Stage 2: Deploy to Kubernetes
deploy:
  stage: deploy
  tags: [k3s, arm64]
  script:
    - envsubst < deployment/deployment.yaml | kubectl apply -f -
  only: [main]
```

GitLab CI/CD: variabili built-in

```
$CI_COMMIT_SHA          # Hash commit (tag immagine)
$CI_COMMIT_BRANCH       # Nome branch
$CI_REGISTRY_IMAGE      # Path registry completo
$CI_REGISTRY_USER       # Autenticazione registry
$CI_REGISTRY_PASSWORD   # Password registry
```

💡 `${CI_COMMIT_SHA}` tagga l'immagine, `${DOMAIN}` per l'Ingress

Kaniko: build senza Docker daemon

❌ Docker-in-Docker

- Richiede privilegi root
- Problemi con ARM64
- Overhead risorse

✅ Kaniko

- Daemonless e unprivileged
- ARM64 ottimizzato
- Kubernetes-native
- OCI-compliant

🚀 Build container images in Kubernetes senza Docker daemon

Build stage: Kaniko in azione

```yaml
build:
  stage: build
  tags: [k3s, arm64]
  image:
    name: gcr.io/kaniko-project/executor:v1.23.2-debug
    entrypoint: [""]

  script:
    - mkdir -p /kaniko/.docker
    - echo "{\"auths\":{\"${CI_REGISTRY}\":{\"auth\":\"$(printf "%s:%s" "${CI_REGISTRY_USER}" "${CI_REGISTRY_PASSWORD
    - /kaniko/executor
      --context "${CI_PROJECT_DIR}"
      --dockerfile "${CI_PROJECT_DIR}/docker/Containerfile"
      --destination "${CI_REGISTRY_IMAGE}:${CI_COMMIT_SHA}"
      --destination "${CI_REGISTRY_IMAGE}:latest"
      --snapshot-mode=redo
      --use-new-run
      --compressed-caching=false
```

💡 **snapshot-mode=redo** = ottimizzato per ARM64, risolve problemi di file system layer

Deploy stage: kubectl apply

```yaml
deploy:
  stage: deploy
  tags: [k3s, arm64]
  image: bitnami/kubectl:latest

  variables:
    KUBECONFIG: /etc/rancher/k3s/k3s.yaml
    DOMAIN: "demo.poggiolab.net"

  before_script:
    - kubectl config set-cluster k3s --server=https://127.0.0.1:6443

  script:
    - envsubst < deployment/deployment.yaml | kubectl apply -f -
    - kubectl rollout status deployment/website -n production --timeout=300s

  only:
    - main
```

⚙️ `envsubst` sostituisce `${CI_REGISTRY_IMAGE}`, `${CI_COMMIT_SHA}`, `${DOMAIN}`

Ambienti effimeri

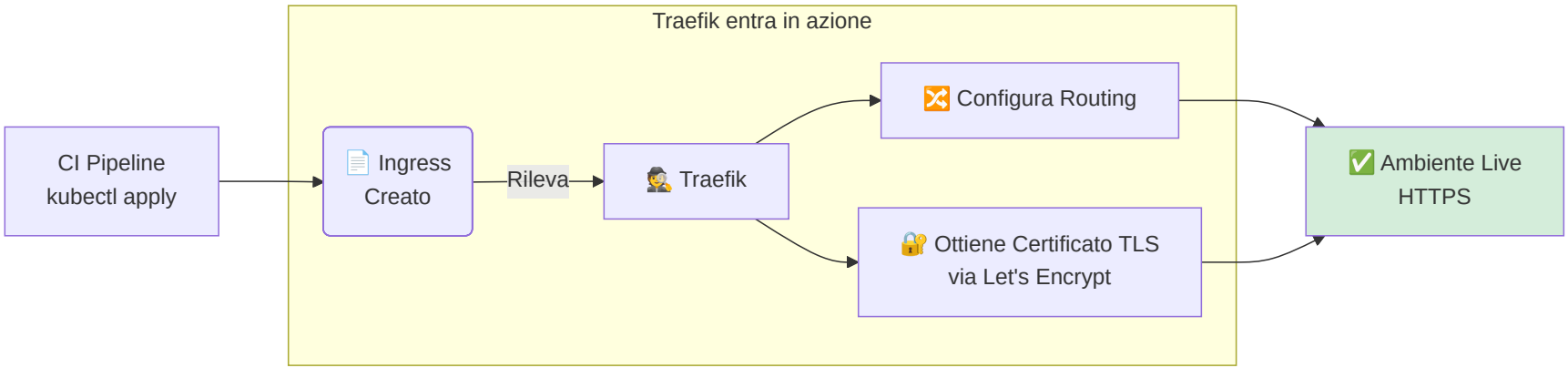
Il problema tradizionale

- 😫 Ambiente di staging condiviso
- 🔀 Conflitti tra feature
- 🐛 Bug difficili da isolare
- ⏰ Code review senza preview
- 🎲 "Funziona sul mio PC"

La soluzione

- ✅ 1 branch = 1 ambiente
- 🚀 Deploy automatico al push
- 🔗 `feature-xyz.demo.poggiolab.net`
- 🔐 HTTPS automatico
- 🗑️ Cleanup alla chiusura PR

Workflow completo: Deploy dietro le quinte



🎬 DEMO LIVE

Ambiente live accessibile

🌐 `https://feature-xyz.demo.poggiolab.net`

✅ HTTPS valido • ✅ Isolato • ✅ Automatico

Cosa abbiamo costruito

🏗️ **Infrastruttura in codice**

🤖 **Deploy automatico**

🌿 **Ambienti effimeri**

🔐 **HTTPS automatico**

🆓 **Zero vendor lock-in**

💰 **~€15/anno**

🎯 **Portabilità • Riproducibilità • Sovranità tecnologica • Pattern enterprise accessibili**

Repository e risorse

📦 gitlab.com/leopoggiani/opensource-deploy-demo

📁 File OpenTofu
Configurazione infrastruttura completa

🐳 Containerfile
Build immagini Podman

⚙️ Pipeline CI/CD
.gitlab-ci.yml completo

📖 Documentazione
Guide step-by-step dettagliate

Next steps

1. 📦 **Fork del repository** → clona e personalizza
2. ☁️ **Setup Oracle Cloud** → free tier (carta richiesta)
3. ⚙️ **Adatta al tuo progetto** → modifica configurazioni
4. 🚀 **Sperimenta e contribuisci** → migliora la community

💡 **gitlab.com/leopoggiani/opensource-deploy-demo**

Feedback

Il tuo feedback è prezioso per migliorare!

📝 app.formbricks.com/s/cmgwanv8w0wfhad01ii9j26×9

Grazie! 🙏

Domande? Curiosità? Discussioni?

@leopoggiani @leonardopoggiani

📦 Repository: gitlab.com/leopoggiani/opensource-deploy-demo

Made with ❤️ and Slidev