TP OPENTOFU AVANCÉ

TP 1 : METTRE EN PLACE UNE PLATEFORME DEVSECOPS LOCALE

CONTEXTE METIER

Une équipe de développement interne veut **héberger sa chaîne de développement en local** pour des raisons :

- De **confidentialité** (code source interne),
- De **coût** (pas de SaaS facturé par utilisateur),
- De **résilience** (pouvoir continuer à bosser même si l'accès Internet est coupé).

L'équipe veut donc une **petite plateforme DevSecOps auto-hébergée** comprenant :

1. **Gitea** : forge Git légère (équivalent auto-hébergé de GitHub/GitLab)
2. **Jenkins** : moteur d'intégration continue pour lancer les builds/tests
3. **SonarQube** : analyse de qualité de code
4. **PostgreSQL** : base de données partagée pour SonarQube (et potentiellement d'autres services)
5. **Un réseau Docker dédié** pour isoler la stack
6. **Des volumes persistants** pour ne pas tout perdre à chaque tofu apply

Le tout doit être **décrit en IaC** avec **OpenTofu** pour pouvoir être :

- Versionné dans Git,
- Rejoué facilement sur les postes des devs,
- Détruit/recréé pour les tests.

OBJECTIFS

1. Décrire une **infrastructure applicative complète** en OpenTofu, pas juste un conteneur.
2. Gérer un **réseau Docker dédié** (pas le bridge par défaut).
3. Déclarer des **volumes persistants** pour Gitea, Jenkins et SonarQube.
4. Faire dialoguer plusieurs services dans **le même réseau**.
5. Comprendre les **contraintes de ressources** (SonarQube est plus lourd).
6. Lancer et arrêter la plateforme en une commande (tofu apply / tofu destroy).

ARCHITECTURE CIBLE

- **docker_network.devops_net**
  - **postgres** (port 5432 exposé en local)
  - **gitea** (port 3000 exposé en local)
  - **jenkins** (port 8080 exposé en local)
  - **sonarqube** (port 9000 exposé en local)
- **Volumes :**
  - pg_data ⟶ Postgres
  - gitea_data ⟶ Gitea
  - jenkins_home ⟶ Jenkins
  - sonar_data ⟶ SonarQube

Tous ces conteneurs tournent **en local** via Docker, et sont **déclarés par OpenTofu**.

CONTRAINTES DU TP

- **Aucun cloud** (pas de provider AWS/Azure/GCP)
- **Uniquement Docker local**
- **OpenTofu** et pas Terraform
- Le TP doit pouvoir tourner dans **la VM Ubuntu** qu'on a déjà utilisée pour les TP précédents (Docker + Minikube)
- On accepte d'exposer beaucoup de ports en local, c'est un environnement de dev
- On met des **mots de passe simples** (pour le TP)

PRE-REQUIS TECHNIQUES (VERSION LOCALE)

Élément	Objectif	Vérification
Docker Desktop	Fournit le moteur Docker local	`docker ps`
OpenTofu	Moteur IaC local	`tofu version`
RAM disponible	≥ 6 Go (SonarQube + Jenkins consomment un peu)	–
OS supporté	macOS / Windows / Linux	–

DECOUPAGE DU TP

Étape 1 – Squelette OpenTofu

- créer main.tf
- déclarer le provider Docker
- créer le réseau devops-net
- tofu init

Étape 2 – Base de données (Postgres)

- ajouter volume Postgres
- conteneur Postgres sur le réseau
- exposer 5432
- tofu plan / tofu apply

Étape 3 – Gitea

- ajouter volume Gitea
- conteneur Gitea (image officielle gitea/gitea:latest)
- brancher sur devops-net
- exposer 3000
- configurer variables d'env de base

Étape 4 – Jenkins

- ajouter volume Jenkins (/var/jenkins_home)
- conteneur jenkins/jenkins:lts-jdk21 (par ex.)
- exposer 8080
- exposer aussi 50000 (agent) mais optionnel en local
- depends_on pour s'assurer que le réseau existe

Étape 5 – SonarQube

- ajouter volume Sonar
- conteneur sonarqube:community
- configurer pour utiliser Postgres du TP
- exposer 9000

Étape 6 – Validation fonctionnelle

- ouvrir http://localhost:3000 → Gitea
- ouvrir http://localhost:8080 → Jenkins
- ouvrir http://localhost:9000 → SonarQube
- vérifier les conteneurs : docker ps

Étape 7 – Nettoyage

- tofu destroy
- vérifier que les conteneurs se sont bien arrêtés

RESSOURCES/DOCUMENTATION UTILES

- Documentation OpenTofu : "OpenTofu – CLI and configuration language"
- Provider Docker (kreuzwerker/docker) : "Docker Provider – Resources: docker_container, docker_image, docker_network, docker_volume"
- Documentation Gitea : "Installation with Docker"
- Documentation Jenkins : "Docker – Jenkins official image"
- Documentation SonarQube : "Install SonarQube with Docker"

LIVRABLES ATTENDUS

- un dossier ateliers/opentofu-devsecops/
- un main.tf fonctionnel
- un README.md qui explique :
    - comment lancer : tofu init && tofu apply
    - les URLs d'accès aux outils
    - comment détruire : tofu destroy
- (optionnel) un variables.tf si tu veux rendre les mots de passe paramétrables

TP 2 : SUPERVISION COMPLETE D'UN ENVIRONNEMENT DE PRODUCTION LOCAL

CONTEXTE

En 2025, les entreprises veulent **surveiller en continu leurs applications** sans dépendre de solutions SaaS payantes (Datadog, New Relic…).
Une équipe DevOps te demande de mettre en place une **stack d'observabilité locale**, utilisée comme environnement de **préproduction**.

Le but est de **superviser une application Node.js** (simulée) via :

- **Prometheus** : collecte de métriques,
- **Grafana** : visualisation de tableaux de bord,
- **Loki** : gestion centralisée des logs.

OBJECTIFS PEDAGOGIQUES

1. Déployer une stack d'observabilité complète via OpenTofu.
2. Configurer un réseau Docker dédié (monitoring-net).
3. Créer des volumes persistants (Prometheus, Grafana, Loki).
4. Lier un conteneur applicatif Node.js avec un exporter Prometheus.
5. Visualiser métriques + logs dans Grafana.
6. Tout détruire proprement (tofu destroy).

SERVICES A DEPLOYER

Service	Image Docker	Port	Volume	Rôle
Prometheus	`prom/prometheus:latest`	9090	`/etc/prometheus`	Scraping des métriques
Grafana	`grafana/grafana:latest`	3000	`/var/lib/grafana`	Visualisation des données
Loki	`grafana/loki:latest`	3100	`/data`	Collecte et indexation des logs

Service	Image Docker	Port	Volume	Rôle
Node App	`stefanprodan/podinfo:latest` *(app simulée)*	8080	–	Application à monitorer

PRE-REQUIS

Élément	Vérification
Docker Desktop fonctionne	`docker ps`
OpenTofu installé	`tofu version`
Ports libres	3000, 8080, 9090, 3100

DEROULEMENT DU TP (ETAPE PAR ETAPE)

ÉTAPE 1 – CREER LE SQUELETTE OPENTOFU

- Déclarer le provider Docker
- Créer un réseau monitoring-net
- tofu init
- Vérifier avec docker network ls

ÉTAPE 2 – DEPLOYER PROMETHEUS

- Créer un volume prometheus_data
- Créer un fichier prometheus.yml avec une configuration simple
- Créer le conteneur Prometheus
- Monter prometheus.yml dans /etc/prometheus/prometheus.yml
- Exposer le port 9090

ÉTAPE 3 – AJOUTER L'APPLICATION NODE.JS

- Utiliser l'image stefanprodan/podinfo:latest (léger, exporte déjà des métriques Prometheus)
- Attacher au réseau monitoring-net
- Exposer le port 8080

ÉTAPE 4 – AJOUTER GRAFANA

- Volume grafana_data
- Conteneur grafana/grafana:latest
- Ports : 3000
- Attacher à monitoring-net

ÉTAPE 5 – AJOUTER LOKI (LOGS)

- Volume loki_data
- Conteneur grafana/loki:latest
- Port : 3100
- Fichier de config minimal monté sur /etc/loki/local-config.yaml

ÉTAPE 6 – RELIER GRAFANA AUX SOURCES

Depuis Grafana (http://localhost:3000) :

- ➕ Add Data Source → Prometheus → http://prometheus:9090
- ➕ Add Data Source → Loki → http://loki:3100

ÉTAPE 7 – VERIFICATION FINALE

Service	Test	Résultat attendu
Node App	curl http://localhost:8080	page HTML podinfo
Prometheus	http://localhost:9090/targets	status = UP
Grafana	http://localhost:3000	tableau de bord visible
Loki	http://localhost:3100/ready	ready

DOCUMENTATION UTILE

Sujet	Lien
OpenTofu Docker Provider	registry.opentofu.org/providers/kreuzwerker/docker
Prometheus	prometheus.io/docs/
Grafana	grafana.com/docs/
Loki	grafana.com/oss/loki/
Podinfo (app demo)	github.com/stefanprodan/podinfo