**Introduction au DevSecOps — Cours express (1h)**

Culture • SDLC sécurisé • CI/CD avec GitHub Actions + Trivy

**Plan (60 min)**

|  |  |
| --- | --- |
| Temps | Sujet |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
| 0–5 | Contexte et objectifs |
| 5–15 | Culture DevSecOps: shift‑left, collaboration |
| 15–25 | SDLC sécurisé (contrôles par étape) |
| 25–45 | Démo CI/CD: tests, build, scans Trivy (FS+Image) |
| 45–55 | Lecture des rapports, remédiations rapides |
| 55–60 | Q/R, extensions (SBOM, SARIF, IaC) |

# Culture DevSecOps (raccourci utile)

* Shift‑left: déplacer les contrôles sécurité au plus tôt (PR).
* Collaboration Dev/Sec/Ops: politiques codées (policy‑as‑code).
* Automatisation: la sécurité comme test reproductible en CI.
* Mesure: CRITICAL/HIGH bloquent, backlog pour le reste.

Anti‑patterns: sécurité en fin de chaîne, scans rares, PDF statiques.

# SDLC sécurisé (vue d’ensemble)

Diagramme (Mermaid lisible) :

flowchart LR  
 A[Idée / User stories] --> B[Conception & Menaces (STRIDE allégé)]  
 B --> C[Dev: PR + tests]  
 C --> D[Build CI]  
 D --> E[Scans sécurité: Trivy FS/Repo]  
 D --> F[Image Docker]  
 F --> G[Scan Image: Trivy]  
 G --> H[Déploiement]  
 H --> I[Runtime: logs, patch mgmt]  
 I -->|feedback| C

1. Contrôles par étape :
2. Conception: revue menaces, pas de secrets en clair.
3. Dev/PR: tests unitaires + Trivy FS (vuln/secrets/misconfig).
4. Build: Trivy Image, seuil CRITICAL/HIGH → fail.
5. Prod: images signées, registry de confiance, patching, monitoring.

# Démo: GitHub Actions + Trivy

Arborescence du dépôt:

.  
├── app.py  
├── requirements.txt  
├── tests/  
│ └── test\_app.py  
├── Dockerfile  
├── .dockerignore  
├── .github/workflows/ci.yml  
├── trivy.yaml  
├── scripts/  
│ ├── scan\_local.sh  
│ └── scan\_local\_dockerized.sh  
└── README.md

Extraits clés (fichiers complets dans le repo):

## app.py (Flask minimal)

from flask import Flask  
app = Flask(\_\_name\_\_)  
  
@app.get("/")  
def hello():  
 return {"msg": "Hello, DevSecOps!"}  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 app.run(host="0.0.0.0", port=8000)

## Dockerfile (bonnes pratiques de base)

FROM python:3.12-slim  
ENV PYTHONDONTWRITEBYTECODE=1 PYTHONUNBUFFERED=1  
RUN adduser --disabled-password --gecos "" appuser  
WORKDIR /app  
COPY requirements.txt .  
RUN pip install --no-cache-dir -r requirements.txt  
COPY . .  
USER appuser  
EXPOSE 8000  
CMD ["gunicorn", "-b", "0.0.0.0:8000", "app:app"]

## trivy.yaml (seuils + scanners)

format: table  
exit-code: 1  
ignore-unfixed: true  
severity: CRITICAL,HIGH  
scanners: [vuln, secret, misconfig]  
list-all-pkgs: true

## .github/workflows/ci.yml (scan FS + Image)

- name: Trivy FS scan  
 uses: aquasecurity/trivy-action@0.28.0  
 with:  
 scan-type: fs  
 scan-ref: .  
 trivy-config: trivy.yaml  
 format: table  
 exit-code: '0'  
- name: Trivy Image scan  
 uses: aquasecurity/trivy-action@0.28.0  
 with:  
 scan-type: image  
 image-ref: devsecops-demo:${{ github.sha }}  
 trivy-config: trivy.yaml  
 format: table  
 exit-code: '1'  
 vuln-type: 'os,library'

## Schéma pipeline (sequence)

sequenceDiagram  
 participant Dev as Dev (push/PR)  
 participant GH as GitHub Actions  
 participant Build as Buildx  
 participant Trivy as Trivy  
 Dev->>GH: push/PR  
 GH->>GH: checkout + tests  
 GH->>Build: docker build (load)  
 Build-->>GH: image:devsecops-demo:SHA  
 GH->>Trivy: scan FS (repo)  
 GH->>Trivy: scan image (CRIT/HIGH => fail)  
 Trivy-->>GH: rapports (logs / tables)

# Extensions possibles

* SBOM (CycloneDX/SPDX) et upload comme artefact CI.
* Export SARIF vers l’onglet Security (Code Scanning).
* Policies (OPA/Conftest) pour Dockerfiles et IaC.
* Scan IaC (Trivy config) pour K8s/Terraform.