Liberté • Égalité • Fraternité
RÉPUBLIQUE FRANÇAISE

MINISTÈRE CHARGÉ DE L'EMPLOI

DOSSIER PROFESSIONNEL (DP)

Nom de naissance - Fialon
Nom d'usage -
Prénom - Sandrine
Adresse - 13008 Marseille

Titre professionnel visé

ADMINISTRATEUR SYSTÈME DEVOPS
Devops & Cloud Engineer

MODALITÉ D'ACCÈS :

- ☑ Parcours de formation
- ☐ Validation des Acquis de l'Expérience (VAE)

Présentation du dossier

Le dossier professionnel (DP) constitue un élément du système de validation du titre professionnel. **Ce titre est délivré par le Ministère chargé de l'emploi.**

Le DP appartient au candidat. Il le conserve, l'actualise durant son parcours et le présente **obligatoirement à chaque session d'examen**.

Pour rédiger le DP, le candidat peut être aidé par un formateur ou par un accompagnateur VAE.

Il est consulté par le jury au moment de la session d'examen.

Pour prendre sa décision, le jury dispose :

1. des résultats de la mise en situation professionnelle complétés, éventuellement, du questionnaire professionnel ou de l'entretien professionnel ou de l'entretien technique ou du questionnement à partir de productions.
2. du **Dossier Professionnel** (DP) dans lequel le candidat a consigné les preuves de sa pratique professionnelle
3. des résultats des évaluations passées en cours de formation lorsque le candidat évalué est issu d'un parcours de formation
4. de l'entretien final (dans le cadre de la session titre).

[Arrêté du 22 décembre 2015, relatif aux conditions de délivrance des titres professionnels du ministère chargé de l'Emploi]

Ce dossier comporte :

- pour chaque activité-type du titre visé, un à trois exemples de pratique professionnelle ;
- un tableau à renseigner si le candidat souhaite porter à la connaissance du jury la détention d'un titre, d'un diplôme, d'un certificat de qualification professionnelle (CQP) ou des attestations de formation ;
- une déclaration sur l'honneur à compléter et à signer ;
- des documents illustrant la pratique professionnelle du candidat (facultatif)
- des annexes, si nécessaire.

DOSSIER PROFESSIONNEL (DP)

Pour compléter ce dossier, le candidat dispose d'un site web en accès libre sur le site.

http://travail-emploi.gouv.fr/titres-professionnels

Sommaire

Exemples de pratique professionnelle

EXEMPLES DE PRATIQUE PROFESSIONNELLE

Activité-type 1 Création de conteneurs avec Docker

Exemple n°1 – Conteneurisation d'une application web multi-étages

1. Décrivez les tâches ou opérations que vous avez effectuées, et dans quelles conditions :

J'ai définit dans un fichier nommé docker-compose.yml une application multi-conteneurs composée de trois services : une base de données MariaDB (db), un backend, et un proxy Nginx.

A. Détail des services :

- **Db (MariaDB)**

Utilise l'image officielle mariadb:latest.
Les variables d'environnement (mot de passe root, nom de la base, utilisateur, mot de passe) sont injectées via un fichier .env ou des variables d'environnement système.
Les données sont persistées dans un volume Docker nommé mysql_data pour éviter la perte de données lors du redémarrage du conteneur.

```
db:
  image: mariadb:latest
  container_name: db
  restart: always
  environment:
    - MARIADB_ROOT_PASSWORD=${MARIADB_ROOT_PASSWORD}
    - MARIADB_DATABASE=${MARIADB_DATABASE}
    - MARIADB_USER=${MARIADB_USER}
    - MARIADB_PASSWORD=${MARIADB_PASSWORD}
  volumes:
    - mysql_data:/var/lib/mysql
```

- **Backend**

Le BackEnd est construit à partir du dossier local ./backend avec le Dockerfile spécifié.
Il reçoit les paramètres de connexion à la base de données via des variables d'environnement.
Il démarre uniquement après le conteneur db grâce à depends_on.

```
backend:
  build:
    context: ./backend
    dockerfile: Dockerfile
  container_name: backend
  environment:
    - DB_HOST=${DB_HOST}
    - DB_USER=${DB_USER}
    - DB_PASSWORD=${DB_PASSWORD}
```

```
  - DB_NAME=${DB_NAME}
 depends_on:
  - db
```

- **Proxy (Nginx)**

Le proxy Nginx utilise l'image officielle nginx:latest.
Il monte un fichier de configuration Nginx local (./proxy/conf) dans le conteneur.
Puis, expose le port 80 du conteneur sur le port 80 de la machine hôte.
Et démarre après le backend.

```
proxy:
  image: nginx:latest
  container_name: proxy
  volumes:
   - ./proxy/conf:/etc/nginx/nginx.conf:ro
  ports:
   - "80:80"
  depends_on:
   - backend
```

- **Volumes**

"mysql_data" est le volume Docker pour stocker les données de la base MariaDB de façon persistante.

```
volumes:
 mysql_data:
```

B. Résultats Obtenus

Démarrage Orchestré : La commande **`docker-compose up -d`** permet de lancer simultanément l'intégralité des trois services dans le bon ordre (**db** → **backend** → **proxy**), grâce à la directive **`depends_on`**.

Accessibilité et fonctionnalité : L'application est devenue accessible via le port **80** de la machine hôte. Le **proxy Nginx** a correctement acheminé les requêtes vers le **backend**, qui a réussi à établir la connexion avec la base de données **MariaDB** en utilisant le nom de service (`db`) comme hôte.

Portabilité Validée : L'ensemble de la configuration étant dans le fichier `docker-compose.yml`, l'application peut être démarrée de manière **identique et reproductible** sur n'importe quelle machine disposant de Docker, indépendamment du système d'exploitation hôte.

2. Précisez les moyens utilisés :

- Docker Engine : Moteur de Conteneurisation
- Docker Compose : Outil d'Orchestration
- Images Officielles Docker : Utilisation de mariadb:latest et nginx:latest pour la rapidité et la fiabilité.
- Dockerfile : Fichier de Configuration qui définit l'image personnalisée du service backend (étapes de build spécifiques : dépendances, code applicatif).
- Volumes Docker pour la persistance des Données
- Fichier .env : Utilisation pour injecter les variables d'environnement sensibles dans la configuration.

3. Avec qui avez-vous travaillé ?

J'ai mené ce projet de manière **autonome**, en me basant sur la documentation officielle de Docker et les cours de ma formation pour concevoir cet exercice.

4. Contexte

Nom de l'entreprise, organisme ou association	**La Capsule**
Chantier, atelier, service	Centre de formation
Période d'exercice	**Du** 22/09/2025 **au** 22/09/2025

5. Informations complémentaires *(facultatif)*

Cliquez ici pour taper du texte.

Activité-type 1 Création de conteneurs avec Docker

Exemple n°2 – Création d'une image Docker personnalisée : Nginx

1. Décrivez les tâches ou opérations que vous avez effectuées, et dans quelles conditions :

L'objectif était de créer une image Docker légère contenant un serveur web **Nginx** configuré pour servir une page d'accueil personnalisée, garantissant ainsi la **reproductibilité** de l'environnement de service.

A. Création du Dockerfile et des Fichiers de Configuration

Sur une VM (Machine Virtuelle) Vagrant, j'ai créé un Dockerfile et défini les étapes de construction de l'image à partir de fichiers locaux :

1. **Image de base :** utilisation de l'image **`nginx:stable-perl`** pour bénéficier d'un environnement Nginx stable et minimal.
2. **Configuration personnalisée :** J'ai copié les fichiers locaux (`default.conf`, `nginx.conf`, `index-lacapsule.html`) dans les répertoires appropriés du conteneur.
   - Le fichier **`default.conf`** a été configuré pour écouter sur le port **`81`** du conteneur.
3. **Permissions et exposition :** J'ai assuré les permissions de lecture nécessaires sur le fichier HTML et exposé le port **`80`** (dans le Dockerfile) pour indiquer le port de l'application.

```
FROM nginx:stable-perl
COPY default.conf /etc/nginx/conf.d/default.conf
COPY nginx.conf /etc/nginx/nginx.conf
COPY index-lacapsule.html /usr/share/nginx/html/index.html
RUN chmod +r /usr/share/nginx/html/index.html
EXPOSE 80
```

Extrait Dockerfile

B. Cycle de Vie de l'Image et du Conteneur

1. **Construction de l'Image :** L'image a été construite et taguée localement :

```
docker build -t nginx-lacapsule:alpha .
```

2. **Lancement du Conteneur :** Un conteneur nommé `webserver` a été lancé, exposant le port interne **`81`** sur le port **`8080`** de la machine hôte.

```
docker run -d -p 8080:81 --name webserver nginx-lacapsule:alpha
```

3. **Vérification :** J'ai validé la bonne exécution et la personnalisation de l'image :

a. **Vérification d'accès :** L'accès au serveur web sur le port `8080` de l'hôte a permis de visualiser la page `index-lacapsule.html`.
b. **Inspection des Fichiers :** J'ai utilisé **`docker exec`** pour vérifier la présence des fichiers de configuration et de la page HTML à l'intérieur du conteneur.

C. Résultats Obtenus

- **Image optimale :** Une image Docker personnalisée (`nginx-lacapsule:alpha`) a été créée avec succès, ne contenant que les éléments essentiels pour servir l'application web.
- **Accessibilité validée :** L'application web a été rendue accessible à l'extérieur, validant le port mapping entre le port d'écoute interne défini dans le `default.conf` (`81`) et le port exposé sur l'hôte (`8080`).
- **Isolation :** Le conteneur fonctionne de manière isolée (`docker run -d`), assurant que les dépendances et la configuration du serveur web sont indépendantes de l'environnement hôte.

2. Précisez les moyens utilisés :

- Docker Engine : moteur de Conteneurisation pour l'exécution des commandes Docker (build, run, exec, images).
- Dockerfile : fichier de Configuration qui décrit les étapes de construction de l'image
- nginx:stable-perl : image parent utilisée pour la construction.
- Configuration Nginx : fichier .conf : personnalisation du comportement du serveur web

3. Avec qui avez-vous travaillé ?

J'ai mené ce projet de manière **autonome**, en me basant sur la documentation officielle de Docker et les cours de ma formation pour concevoir cet exercice.

4. Contexte

Nom de l'entreprise, organisme ou association ▸ **La Capsule**

Chantier, atelier, service ▸ Atelier en Centre de formation

Période d'exercice ▸ **Du** 23/09/2025 **au** 23/09/2025

5. Informations complémentaires *(facultatif)*

Cliquez ici pour taper du texte.

Activité-type 2 Teste d'intégration continue (CI)

Exemple n°2 - *Sonar Qube Cloud : Analyse de la qualité et de la sécurité du code*

1. Décrivez les tâches ou opérations que vous avez effectuées, et dans quelles conditions :

SonarQube Cloud est un outil cloud en ligne qui permet de tester un répertoire git en analysant la qualité du code et sa sécurité.

Tâches et opérations Effectuées

L'objectif principal de cette tâche était d'activer l'analyse automatique de la qualité et de la sécurité de mon code source en utilisant l'intégration native de SonarQube Cloud avec GitHub.

Les étapes clés ont été les suivantes :

1. **Préparation et Connexion :**

J'ai créé un compte sur SonarQube Cloud, puis autorisé SonarQube à accéder à mes dépôts GitHub via l'interface, établissant une liaison bidirectionnelle.

2. **Configuration du Projet :**

J'ai **importé le projet** spécifique depuis GitHub dans SonarQube Cloud.

SonarQube a automatiquement détecté le langage du code (ici JavaScript) et a configuré les règles d'analyse par défaut.

3. **Déclenchement de l'Analyse Automatique :**

Contrairement à une intégration CI/CD manuelle, l'analyse a été déclenchée de manière automatique.

J'ai effectué des modifications de code sur mon projet et j'ai lancé un *commit* suivi d'un *push* vers le dépôt GitHub. Chaque *push* sert de déclencheur pour que SonarQube Cloud récupère le code et exécute son analyse.

4. **Vérification et Interprétation des Résultats :**

J'ai analysé les résultats sur le tableau de bord de SonarQube, en examinant le Quality Gate, Reliability (Fiabilité, bugs), les Vulnérabilités (sécurité) et la maintenabilité du code.

L'objectif est d'atteindre le **'A'** pour valider la bonne santé de l'application.

De plus, l'analyse a mis en lumière les points suivants :

- **Hotspots Review** : L'outil a identifié les zones du code nécessitant une attention particulière pour la sécurité (les **Hotspots**). Ces zones ont été révisées manuellement, permettant de valider qu'elles ne présentent pas de risque de vulnérabilité.
- **Duplications** : La mesure des lignes de code répétées a fourni un pourcentage précis. La correction des duplications a été initiée pour rendre le code plus modulaire, il en reste encore pour atteindre 0 %.
- **Validation du Quality Gate** : L'ensemble de ces métriques (notamment les notes 'A') a permis d'obtenir le statut **"Passed"** (Réussi), validant que le code respecte les standards définis avant de passer à l'étape suivante du pipeline (déploiement).

2. Précisez les moyens utilisés :

- SonarQube Cloud : Plateforme en ligne, d'analyse statique du code.
- GitHub : Hébergement du code et déclencheur des analyses.
- Commandes Git : *git commit* et *git push* ont servi à déclencher l'analyse.

3. Avec qui avez-vous travaillé ?

J'ai mené ce projet de manière **autonome** sur des projets personnels essentiellement.

4. Contexte

Nom de l'entreprise, organisme ou association	**La Capsule**
Chantier, atelier, service	Centre de formation
Période d'exercice	**Du** 04/11/2025 **au** 21/11/2025

5. Informations complémentaires *(facultatif)*

Activité-type 2 Teste d'intégration continue (CI)

Exemple n°2 - *Validation de code terraform et Ansible*

1. Décrivez les tâches ou opérations que vous avez effectuées, et dans quelles conditions :

L'objectif de cet exercice était d'assurer la **qualité**, la **conformité** et le **respect des bonnes pratiques** de l'Infrastructure-as-Code (IaC) en intégrant des outils de **validation statique** directement dans le flux de développement.

Tâches et Opérations Effectuées

J'ai mis en place une routine de validation locale pour les deux composants principaux de l'IaC :

1. **Validation Syntaxique et Structurelle (Terraform) :**

J'ai systématiquement exécuté la commande `terraform init` en début de projet pour initialiser le répertoire de travail, télécharger les *providers* nécessaires et valider la structure de base.

La commande `terraform validate` était exécutée après chaque modification significative des fichiers `.tf`. Cette opération vérifie la cohérence syntaxique du langage HCL (*HashiCorp Configuration Language*) et s'assure que les arguments, les blocs et les références entre les ressources sont corrects, sans nécessiter de connexion à l'environnement Cloud.

Résultat : Cette étape me renvoyait soit un message de `Success`, soit une liste d'erreurs précises avec le nom du fichier et le numéro de ligne.

2. **Audit de Qualité et de Conformité (Ansible) :**

J'ai utilisé l'outil `ansible-lint` pour analyser la qualité de mes *playbooks* et de mes *rôles* Ansible (fichiers `.yml`).

Cette commande ne vérifie pas seulement la syntaxe YAML, mais applique un ensemble de **règles de bonnes pratiques** (exemple : ne pas utiliser telle commande en production, utiliser des modules Ansible spécifiques).

Exemple d'exécution : J'ai ciblé un rôle spécifique pour l'installation de Prometheus *: ansible-lint roles/prometheus/tasks/main.yml.*

Résultat : Les messages d'erreur de `ansible-lint` me permettaient d'identifier et de corriger les failles de style ou les potentielles erreurs de configuration avant le déploiement.

2. Précisez les moyens utilisés :

- Terraform Code : tous les fichiers .tf
- Terraform CLI : exécution de init et validate pour la vérification des fichiers HCL.
- Ansible Playbooks & Roles : tous les fichiers .yml
- Ansible-lint : linter : outil Python pour l'analyse statique des fichiers YAML d'Ansible et la détection des mauvaises pratiques.

3. Avec qui avez-vous travaillé ?

J'ai mené ce projet de manière **autonome** sur un projet de fin de formation.

4. Contexte

Nom de l'entreprise, organisme ou association	**La Capsule**
Chantier, atelier, service	Centre de formation
Période d'exercice	**Du** 27/10/2025 **au** 08/11/2025

5. Informations complémentaires *(facultatif)*

Cliquez ici pour taper du texte.

Activité-type 3 — Automatisation de déploiement - IaC

Exemple n°1 – ***Provisionning avec Terraform : Déploiement d'une infrastructure cloud complète***

1. Décrivez les tâches ou opérations que vous avez effectuées, et dans quelles conditions :

J'ai piloté l'intégralité de la phase de provisionnement de l'infrastructure sur AWS en utilisant Terraform, que j'ai entièrement rédigé dans un `main.tf`.

A. Tâches de Conception et de Sécurité

- **Rédaction du Code IaC** : j'ai rédigé le fichier main.tf, définissant l'intégralité de l'architecture AWS comme code.
- **Gestion du *State* via Terraform Cloud :** J'ai configuré un **backend distant** vers **Terraform Cloud**. Cela garantit nativement le **verrouillage de l'état** (*State Locking*), la **centralisation sécurisée** des données d'état, et la gestion des **espaces de travail** (*Workspaces*).
- **Gestion des Clés SSH :** j'ai utilisé le *provider* **`tls`** pour **générer une paire de clés SSH (RSA 4096 bits)** dynamiquement, et le *provider* `local` pour sauvegarder la clé privée (`.pem`) avec des permissions restrictives (`0600`) sur le poste de contrôle.

```
resource "aws_key_pair" "cle_ssh" {
  key_name   = "cle_ssh_${terraform.workspace}"
  public_key = file("${path.root}/keys/ci/cle_ssh.pem.pub")
  tags = {
    Name = "SSH Key - ${terraform.workspace}"
  }
}
```

Extrait du fichier main.tf

B. Provisionnement du Réseau et de l'Instance EC2

- **Création d'une architecture réseau dédiée :** j'ai provisionné une infrastructure réseau complète incluant un **VPC**, un **Subnet public**, une **Internet Gateway (IGW)**, et une **Route Table**.
- **Sélection Dynamique de l'AMI :** j'ai utilisé la source de données **`aws_ami`** pour récupérer l'ID de l'**AMI Linux**.
- **Durcissement (Hardening) du Groupe de Sécurité :** j'ai créé un groupe de sécurité strict, n'autorisant le trafic **SSH (port 22)** entrant qu'à partir de mon **adresse IP publique** spécifique (via `var.my_ip`), minimisant la surface d'attaque.
- **Provisionnement de l'Instance :** j'ai créé l'instance EC2 (`aws_instance`) en lui assignant tous les composants réseau et de sécurité.

- **Ouverture des droits AIM pour l'agent Cloudwatch** afin de récupérer les metrics, logs et tâches de cette instance.

```
resource "aws_instance" "monitoring" {
  subnet_id            = var.subnet_id
  vpc_security_group_ids = [var.security_group_id]
  ami                  = "ami-02d7ced41dff52ebc" # Amazon Linux 2023
  instance_type        = "t3.micro"
  iam_instance_profile = aws_iam_instance_profile.monitoring_profile.name

  key_name = var.ssh_key_name

  tags = {
    Name        = "Prometheus-Grafana-${terraform.workspace}"
    Service     = "Monitoring"
    Workspace   = terraform.workspace
    AnsibleGroup = "monitoring" # Tag pour Ansible
  }
}
```

Extrait du fichier main.tf

C. Validation

- **Récupération de l'IP :** j'ai configuré un `output` pour exporter l'**adresse IP publique** de l'instance.
- **Test sur la console AWS :** j'ai vérifié que l'instance et les groupes de sécurité et vpc étaient bien provisionnés.
- **Test de Connexion :** j'ai testé et validé la connexion **SSH via le terminal** en utilisant le fichier `.pem` généré localement pour confirmer l'accès initial au serveur.

D. Résultats Obtenus

Le processus de provisionnement avec Terraform a été un succès, permettant de valider les objectifs suivants :

- **Infrastructure déterminée :** après exécution des commandes `terraform plan` puis `terraform apply`, l'intégralité de l'architecture (VPC, sous-réseau, Instance EC2, Groupe de Sécurité) a été créée sur **AWS** de manière **répétable** et **déterministe**, comme défini dans le `main.tf`.
- **State Management Centralisé :** Le fichier d'état (*state file*) a été stocké avec succès sur **Terraform Cloud**, assurant que le suivi de l'infrastructure est **centralisé, verrouillé** et géré via le service web.
- **Gestion du *State* via Terraform Cloud :** j'ai configuré un **backend distant** vers **Terraform Cloud**. Cela garantit nativement le **verrouillage de l'état** (*State Locking*), la **centralisation sécurisée** des données d'état, et la gestion des **espaces de travail** (*Workspaces*).

- **Connectivité Validée :** La connexion SSH à l'instance EC2, utilisant la paire de clés générée dynamiquement, a été établie avec succès, validant à la fois l'accès réseau (VPC, IGW, SG) et la liaison correcte de la clé SSH à l'instance.
- **Intégration Future Facilitée :** L'instance a été taguée avec `AnsibleGroup = "monitoring"`, ce qui la rend directement exploitable par Ansible pour l'étape de configuration (Exemple n°2 : Configuration avec Ansible).

2. Précisez les moyens utilisés :

- Terraform Code : Fichiers .tf
- Terraform CLI : Exécution de `init`, `plan`, `apply` pour le cycle de vie de l'infrastructure.
- AWS : Cloud Provider : Plateforme cible pour le provisionnement des ressources (VPC, EC2, S3, IAM).

3. Avec qui avez-vous travaillé ?

J'ai mené ce projet de manière **autonome** sur un projet de fin de formation.

4. Contexte

Nom de l'entreprise, organisme ou association	**La Capsule**
Chantier, atelier, service	Centre de formation
Période d'exercice	**Du** 27/10/2025 **au** 08/11/2025

5. Informations complémentaires *(facultatif)*

Cliquez ici pour taper du texte.

Activité-type 3 Automatisation de déploiement - IaC

Exemple n°2 - Gestion de l'infrastructure Cloud avec les CLI : AWS, Azure CLI et GCP

1. Décrivez les tâches ou opérations que vous avez effectuées, et dans quelles conditions :

L'objectif était de maîtriser l'utilisation des outils CLI (Command Line Interface) natifs de chaque fournisseur Cloud pour l'authentification, le déploiement et la gestion des ressources.

A. Tâches de Configuration et d'Authentification

J'ai effectué l'installation et la configuration des outils CLI sur mon poste de travail :

- **Google Cloud (GCP) :** Installation du **Cloud SDK** (`./google-cloud-sdk/install.sh`) et configuration du compte par défaut avec **`gcloud init`**.
- **Azure :** Installation de l'**Azure CLI** (`az cli`) via `brew` (sur Mac) et authentification interactive via **`az login`** pour établir une session sécurisée.
- **AWS :** J'ai vérifié la configuration de l'**AWS CLI** qui permet d'utiliser le même profil de sécurité que Terraform pour les opérations manuelles.

B. Opérations Cloud (Exemple d'Hébergement Statique sur GCP)

Pour illustrer l'utilisation du CLI pour un déploiement, j'ai réalisé un exemple d'hébergement statique de site web sur **Google Cloud Storage (GCS)** :

1. **Création du Bucket :** J'ai créé un *bucket* GCS via l'interface web, puis interagi avec lui localement.

Transfert de Fichiers : J'ai utilisé l'utilitaire **`gsutil`** (intégré au Cloud SDK) pour transférer le contenu local vers le *bucket* :

```
gsutil cp -r <fichier_en_local> gs://<nom_du_bucket>
```

Configuration de l'Hébergement : J'ai configuré la page d'index par défaut pour le *bucket* via `gcloud storage` :

```
gcloud storage buckets update gs://VOTRE-BUCKET-NAME --web-main-page-suffix index.html
```

2. **Ouverture Publique :** J'ai utilisé **`gcloud storage buckets add-iam-policy-binding`** pour rendre le contenu du *bucket* accessible publiquement, en assignant le rôle `roles/storage.objectViewer` à `allUsers`.

C. Résultats Obtenus

- **Validation de l'Authentification :** L'authentification a été réussie sur les trois environnements (AWS, Azure, GCP), confirmant la capacité à gérer l'accès aux ressources à partir du poste de contrôle local.
- **Déploiement Fonctionnel :** L'hébergement statique sur **Google Cloud Storage (GCS)** a été un succès. Le site web a été rendu accessible publiquement via l'URL du *bucket*.
- **Maîtrise du Scripting :** La preuve de concept a montré que des tâches complexes (comme la configuration des droits IAM et des propriétés d'hébergement) peuvent être gérées intégralement via des **scripts CLI**, ouvrant la voie à une automatisation plus fine des *workflows* CI/CD.

2. Précisez les moyens utilisés :

- AWS CLI
- Azure CLI
- Google Cloud SDK
- Zsh / Terminal : pour lancer les lignes de commandes

3. Avec qui avez-vous travaillé ?

J'ai mené ce projet de manière **autonome** sur ces exercices de fin de formation, en m'aidant de la documentation en ligne des 3 providers.

4. Contexte

Nom de l'entreprise, organisme ou association	**La Capsule**
Chantier, atelier, service	Centre de formation
Période d'exercice	**Du** 20/10/2025 **au** 27/10/2025

5. Informations complémentaires *(facultatif)*

Cliquez ici pour taper du texte.

Activité-type 3 Automatisation de déploiement - IaC

Exemple n°3 – Configuration avec Ansible : Déploiement du monitoring

1. Décrivez les tâches ou opérations que vous avez effectuées, et dans quelles conditions :

J'ai automatisé l'installation et la configuration d'une infrastructure de **monitoring** grâce à **Ansible**.

J'ai dédié une **machine virtuelle** AWS à **Prometheus** et **Grafana**, permettant la **collecte, la centralisation et la visualisation des métriques** de l'ensemble de l'infrastructure.

A. Prometheus - installation

L'installation de **Prometheus** est automatisée via un rôle Ansible spécifique.
Ansible se connecte à l'hôte par **SSH**, télécharge le binaire officiel depuis GitHub à l'aide du module ansible.builtin.get_url, puis **installe et démarre le service**.

```
ansible.builtin.get_url:
  url: "https://github.com/prometheus/prometheus/releases/download/v{{
prometheus_version }}/prometheus-{{ prometheus_version }}.linux-amd64.tar.gz"
  dest: /tmp/prometheus-{{ prometheus_version }}.tar.gz
  mode: '0644'
 when: not prometheus_installed.stat.exists
```

Extrait du fichier /ansible/roles/prometheus/tasks/install_prometheus.yml

B. Grafana - installation

L'installation de **Grafana OSS** est gérée par un second rôle Ansible.
Le rôle installe les **dépendances système**, ajoute la **clé GPG** (pour garantir l'authenticité du binaire téléchargé)**,** configure le **dépôt officiel Grafana**, puis installe et démarre le service :

Installation Grafana

```
- name: 3. Install dependencies for apt repository management
 ansible.builtin.apt:
   name: ['apt-transport-https', 'software-properties-common', 'wget', 'apg']
   state: present
   update_cache: true

- name: 4. Add Grafana GPG key
 ansible.builtin.apt_key:
   url: https://packages.grafana.com/gpg.key
   state: present
```

```
- name: 5. Add Grafana repository
 ansible.builtin.apt_repository:
  repo: "deb https://packages.grafana.com/oss/deb stable main"
  state: present

- name: 6. Install Grafana (OSS)
 ansible.builtin.apt:
  name: grafana
  state: present
…
```

Extrait du fichier /ansible/roles/grafana/tasks/install_grafana.yml

C. Grafana - configuration des Dashboard

La configuration initiale de Grafana a nécessité plusieurs ajustements.
Par défaut, Ansible ne permet pas de modifier le mot de passe de l'utilisateur *admin*.
J'ai donc contourné cette limitation en créant un **nouvel utilisateur administrateur** via Ansible, avant de **réinitialiser le mot de passe admin existant**.

Concernant les **dashboards**, j'ai d'abord tenté de créer mes propres modèles, mais j'ai rencontré des **incompatibilités avec les sources CloudWatch** et quelques **retards de synchronisation**.
Finalement, j'ai importé et adapté des **dashboards open source** déjà optimisés, en format json, pour la surveillance des EC2 et des métriques AWS.

L'ajout automatique des **datasources CloudWatch** et **Prometheus**, ainsi que l'import des dashboards, est également géré par Ansible via des requêtes HTTP (ansible.builtin.uri).

```
- name: 2f. Créer la datasource CloudWatch (via agent) ansible.builtin.uri:
  url: "http://localhost:3000/api/datasources"
  body_format: json
  body:
    name: "CloudWatch"
    type: "cloudwatch"
    access: "proxy"
    jsonData:
      authType: "ec2_iam_role"
      customMetricsNamespaces: "{{ aws_custom_metrics_namespaces }}"

- name: 3b. Importer le dashboard EC2 personnalisé dans Grafana
 ansible.builtin.uri:
  url: "http://localhost:3000/api/dashboards/import"
```

```
  body_format: json
  body:
    dashboard: "{{ grafana_ec2_dashboard_json }}"
    overwrite: true
    inputs:
      - name: "DS_CLOUDWATCH"
        type: "datasource"
        pluginId: "cloudwatch"
        value: "CloudWatch"
  status_code: [200, 201]
 register: grafana_ec2_dashboard_result
…
```

Extrait du fichier /ansible/roles/grafana/tasks/add_dashboard_inline.yml

Les dashboards Grafana offrent ensuite une **vue en temps réel** sur l'état des instances EC2, des bases RDS, et des métriques collectées via Prometheus.
L'interface est accessible via le port 3000 : http://<IP_VM_PUBLIC>:3000.

Extraits de Dashboards – CloudWatch sur plusieurs instances EC2

Ce dashboard (capture d'écran ci-dessus) permet de **visualiser en temps réel l'activité des instances EC2**, notamment les variations de CPU, mémoire, réseau et stockage.

On distingue clairement des **pics d'activité inhabituels** sur l'instance publique de monitoring, observés autour de 7h du matin. Ils correspondent probablement à des **tentatives d'accès externes automatisées** sur les ports **SSH** ou **HTTP**, un phénomène courant pour les instances **exposées sur Internet**.

D. Axes d'amélioration du Monitoring

Une évolution pertinente serait de **remplacer les identifiants techniques des instances par leurs noms explicites** dans le dashboard *CloudWatch EC2*, afin de rendre la supervision **plus claire et plus intuitive** pour l'équipe.

2. Précisez les moyens utilisés :

- Ansible : outil de configuration de l'intégralité du déploiement
- Prometheus et Grafana : logiciels cibles
- SSH (Clé Privée) : Méthode utilisée par Ansible pour se connecter à l'hôte EC2 (via la clé générée par Terraform).

3. Avec qui avez-vous travaillé ?

J'ai mené ce projet de manière **autonome**, ce qui m'a permis de démontrer ma capacité à gérer l'intégralité du cycle de vie de l'automatisation.

J'ai principalement utilisé la **documentation officielle d'Ansible** comme référence technique principale, garantissant l'application des bonnes pratiques et des modules canoniques.

En complément de mon travail autonome, j'ai eu recours à **Copilot, l'agent IA sur VSCode**. Cet outil d'assistance a été essentiel pour expliquer des concepts complexes et effectuer une vérification syntaxique et logique rapide du code YAML. L'intégration de cet outil moderne a permis d'améliorer la qualité du code et d'accélérer le processus de débogage.

4. Contexte

Nom de l'entreprise, organisme ou association	**La Capsule**
Chantier, atelier, service	Centre de formation

Période d'exercice Du 27/10/2025 au 08/11/2025

5. Informations complémentaires *(facultatif)*

Cliquez ici pour taper du texte.

Activité-type 3 — Automatisation de déploiement - IaC

Exemple n°4 — ***Pipeline : CI/CD de l'IaC : Planification et application automatisées***

1. Décrivez les tâches ou opérations que vous avez effectuées, et dans quelles conditions :

Tous les éléments de l'IaC (Terraform pour le provisionnement et Ansible pour la configuration) sont liés et exécutés séquentiellement au sein du même pipeline CI/CD.

A. Validation : Tests intégrés

J'ai mis en place des tests automatisés lors de l'intégration continue via GitLab CI.
À chaque merge sur la branche staging, la pipeline exécute automatiquement la validation du code Terraform et Ansible, afin de détecter toute erreur de syntaxe ou de configuration avant le déploiement, afin d'empêcher de merger du code cassé.

Terraform	Ansible
terraform-validate: stage: validate only: refs: - merge_requests script: - cd terraform - terraform init -input=false - terraform validate	ansible-validate: stage: validate only: refs: - merge_requests script: … - ansible-playbook -i inventories/staging.yml site.yml --syntax-check --check --vault-password-file .vault_pass - rm .vault_pass
Extrait du fichier .gitlab-ci.yml	

B. Déploiement : Lancement des script d'automatisation et de configuration

Ces jobs ne s'exécutent que sur les branches staging ou main, et seulement manuellement pour plus de sécurité.
Terraform – Déploiement de l'infrastructure
Le terraform va créer les instances, les groupes de sécurité et tout le réseau de l'application.

```
terraform:
 stage: deploy
 rules:
  - if: '$CI_COMMIT_REF_NAME == "staging" || $CI_COMMIT_REF_NAME == "main"'
    when: manual
 script:
  - terraform plan -input=false -out=tfplan
  - terraform apply -input=false -auto-approve tfplan
```

Extrait du fichier .gitlab-ci.yml

Ansible va ensuite configurer les machines, le monitoring, installer et lancer l'application en la connectant aux bases de données.

```
- ansible-playbook -i inventories/$TF_WORKSPACE.yml site.yml --vault-password-file .vault_pass
```

Extrait du fichier .gitlab-ci.yml

2. Précisez les moyens utilisés :

- GitLab CI : Moteur d'automatisation des tâches, orchestrant l'intégralité du pipeline (Validation, Déploiement).
- Fichier .gitlab-ci.yml : Fichier YAML décrivant les étapes (stages), les tâches (jobs) et les règles d'exécution.
- Terraform CLI : Outils IaC exécuté dans le pipeline pour les commandes validate, plan et apply pour lancer les tâches du main.tf
- Ansible : Outil de configuration exécuté dans le pipeline pour la configuration des hôtes via les playbooks.

3. Avec qui avez-vous travaillé ?

J'ai mené ce projet de manière **autonome** sur un projet de fin de formation.

4. Contexte

Nom de l'entreprise, organisme ou association La Capsule

Chantier, atelier, service Centre de formation

Période d'exercice **Du** 27/10/2025 **au** 08/11/2025

5. Informations complémentaires *(facultatif)*

Cliquez ici pour taper du texte.

Titres, diplômes, CQP, attestations de formation

(facultatif)

Intitulé	Autorité ou organisme	Date
Concepteur-Développeur d'Applications Web & Mobile	La Capsule	2023

Déclaration sur l'honneur

Je soussigné(e) [prénom et nom] *Sandrine Fialon* ,

déclare sur l'honneur que les renseignements fournis dans ce dossier sont exacts et que je suis l'auteur(e) des réalisations jointes.

Fait à *Marseille* le *21 novembre 2025*

pour faire valoir ce que de droit.

Signature :

SF

Documents illustrant la pratique professionnelle

(facultatif)

Intitulé
Cliquez ici pour taper du texte.

ANNEXES

(Si le RC le prévoit)