**Activités Types et Compétences Professionnelles**

1. **Automatiser la création de serveurs à l’aide de scripts- Explication**: Cette compétence consiste à utiliser des scripts pour créer et configurer des serveurs automatiquement, sans intervention manuelle. Cela permet de gagner du temps et d'assurer une configuration uniforme et reproductible.

**Automatiser la création de serveurs à l’aide de scripts**

**Description de la compétence – processus de mise en œuvre**

Automatiser la création de serveurs à l'aide de scripts implique la conception et l'écriture de scripts pour provisionner et configurer des serveurs virtuels de manière efficace et reproductible. Cela comprend la création de scripts pour différents systèmes d'exploitation tels que Linux et Windows, ainsi que l'utilisation de langages de programmation génériques comme Python pour automatiser les tâches liées à la configuration et au déploiement des serveurs.

**Contexte(s) professionnel(s) de mise en œuvre**

Cette compétence est largement utilisée dans les équipes DevOps et les environnements informatiques modernes où l'automatisation est essentielle pour accélérer le déploiement des infrastructures et garantir leur cohérence et leur fiabilité. Elle est particulièrement utile lorsqu'un grand nombre de serveurs doivent être déployés et configurés de manière cohérente.

**Exemple d'implémentation**

1. **Créer un script d’automatisation pour Linux (Bash)**:
   * Un script Bash est écrit pour automatiser la création et la configuration d'un serveur Linux. Ce script peut inclure des commandes pour installer des logiciels, configurer des paramètres système et sécuriser le serveur.
2. **Créer un script d’automatisation pour Windows (PowerShell)**:
   * Un script PowerShell est utilisé pour automatiser le déploiement et la configuration d'un serveur Windows. Il peut contenir des instructions pour installer des rôles et des fonctionnalités Windows, configurer des services, etc.
3. **Créer un script d’automatisation avec un langage générique (Python)**:
   * Un script Python est développé pour automatiser des tâches spécifiques liées à la configuration des serveurs, indépendamment du système d'exploitation. Par exemple, il peut être utilisé pour générer des fichiers de configuration, gérer des packages logiciels, etc.

**Savoirs requis**

1. **Créer une machine virtuelle de serveur manuellement ou automatiquement**:
   * Compréhension des outils de virtualisation et capacité à créer des machines virtuelles manuellement ou à l'aide de scripts ou de modèles prédéfinis.
2. **Automatiser la configuration du serveur**:
   * Compétence à écrire des scripts ou des programmes pour automatiser la configuration du serveur, y compris l'installation de logiciels, la configuration des services réseau, la sécurisation du système, etc.
3. **Vérifier le bon fonctionnement du système installé**:
   * Capacité à effectuer des tests pour vérifier que le système installé fonctionne correctement et répond aux exigences spécifiées dans le cahier des charges. Cela peut inclure des tests de fonctionnalité, des tests de performance, des tests de sécurité, etc.

**Exemple**: Utiliser un script Shell (Bash) pour installer Apache et configurer un serveur web.

bash

Copier le code

#!/bin/bash

# Mise à jour des paquets

sudo apt-get update

# Installation d'Apache

sudo apt-get install -y apache2

# Démarrage du service Apache

sudo systemctl start apache2

# Activer Apache pour qu'il démarre au démarrage

sudo systemctl enable apache2

echo "Serveur Apache installé et démarré."

**2. Automatiser le déploiement d'une infrastructureExplication**: Cela implique l'utilisation d'outils d'infrastructure as code (IaC) pour déployer et gérer les infrastructures. Cela permet de définir l'infrastructure en tant que code, ce qui facilite la gestion des versions et le déploiement automatisé.

**Automatiser le déploiement d'une infrastructure**

**Description de la compétence – processus de mise en œuvre-** Automatiser le déploiement d'une infrastructure implique l'utilisation d'outils et de scripts pour provisionner et configurer les ressources nécessaires à l'infrastructure informatique de manière efficace et reproductible. Ce processus peut être réalisé en utilisant des plateformes automatisation telles que Ansible ou Terraform. Le déploiement peut être effectué en deux étapes : d'abord dans un environnement de pré-production pour tester les configurations, puis dans un environnement de production une fois que les tests sont satisfaisants.

**Contexte(s) professionnel(s) de mise en œuvre**

Cette compétence est utilisée dans divers contextes professionnels, notamment lorsque des entreprises déploient des infrastructures informatiques en cloud public, privé ou hybride. Elle est particulièrement importante pour les équipes DevOps qui cherchent à automatiser les processus de déploiement et à accélérer la mise en production de nouvelles ressources.

**Exemples concrets**

1. **Déploiement dans un environnement de préproduction**:
   * Un script Ansible est utilisé pour provisionner des machines virtuelles et configurer les logiciels nécessaires dans un environnement de test. Cela permet aux équipes de vérifier que les configurations sont correctes avant de les déployer en production.
2. **Déploiement dans un environnement de production**:
   * Une fois que les tests dans l'environnement de préproduction sont réussis, le même script Ansible est utilisé pour déployer les mêmes configurations dans l'environnement de production. Cela garantit la cohérence entre les environnements et réduit les erreurs humaines.

**Savoirs requis**

1. **Connaissance des principes de la virtualisation**:
   * Compréhension des concepts de virtualisation et des technologies associées, telles que la création et la gestion de machines virtuelles.
2. **Connaissance des principes du réseau IP**:
   * Compréhension des fondamentaux du protocole Internet (IP) et des concepts liés au routage, à la segmentation réseau et à la résolution d'adresses.
3. **Connaissance des principaux systèmes d'exploitation Linux et Windows**:
   * Familiarité avec les systèmes d'exploitation les plus couramment utilisés dans les environnements d'infrastructure informatique, notamment Linux et Windows, pour configurer et gérer efficacement les serveurs.

**Exemple**: Utiliser Terraform pour déployer une infrastructure sur AWS.

hcl

Copier le code

provider "aws" {

region = "us-west-2"

}

resource "aws\_instance" "example" {

ami = "ami-0c55b159cbfafe1f0"

instance\_type = "t2.micro"

tags = {

Name = "example-instance"

}

}

**3. Sécuriser l’infrastructure**

**Explication**: Cette compétence consiste à mettre en place des mesures de sécurité pour protéger l'infrastructure contre les menaces. Cela inclut la configuration de pare-feu, la gestion des accès, la surveillance de la sécurité, et la mise en œuvre des bonnes pratiques de sécurité.

**Exemple**: Utiliser Ansible pour configurer un pare-feu sur un serveur.

yaml

Copier le code

---

- name: Configurer un pare-feu avec UFW

hosts: all

become: yes

tasks:

- name: Installer UFW

apt:

name: ufw

state: present

- name: Autoriser les connexions SSH

ufw:

rule: allow

port: 22

- name: Autoriser les connexions HTTP

ufw:

rule: allow

port: 80

- name: Activer UFW

ufw:

state: enabled

**Sécuriser l’infrastructure**

**Description de la compétence – processus de mise en œuvre**

Cette compétence implique l'intégration de règles de sécurité dans les serveurs à déployer, en tenant compte des exigences spécifiques définies dans un cahier des charges ainsi que des risques potentiels liés aux systèmes déployés. Les étapes clés du processus incluent l'utilisation d'un environnement de test et de pré-production pour tester les mises à jour avant leur déploiement en production.

**Contexte(s) professionnel(s) de mise en œuvre**

Cette compétence est une pratique recommandée dans tous les environnements informatiques, même en l'absence d'une demande formelle du responsable. Elle vise à assurer la sécurité et la fiabilité des systèmes déployés.

**Critères de performance**

* Les mises à jour sont testées de manière approfondie avant leur déploiement en production pour garantir la stabilité du système.
* Les accès aux serveurs sont sécurisés à l'aide de méthodes d'authentification robustes et de bonnes pratiques de gestion des identités et des accès.

**Savoir-faire techniques, organisationnels, relationnels, savoirs : Savoir-faire techniques**

1. **Appliquer les recommandations de configuration d’un système GNU/Linux de l’ANSSI**:
   * Capacité à mettre en œuvre les recommandations de sécurité émises par l'ANSSI (Agence Nationale de la Sécurité des Systèmes d'Information) pour configurer et sécuriser les systèmes Linux.
2. **Mettre en place une méthode d’authentification sécurisée**:
   * Compétence à déployer des méthodes d'authentification robustes, telles que l'authentification à deux facteurs ou l'authentification basée sur des certificats.
3. **Configurer le pare-feu du système**:
   * Maîtrise de la configuration du pare-feu pour filtrer le trafic réseau et protéger les serveurs contre les attaques malveillantes.
4. **Déployer des certificats**:
   * Capacité à mettre en place des certificats SSL/TLS pour chiffrer les communications et assurer l'authenticité des services web.

En résumé, sécuriser l'infrastructure informatique implique la mise en œuvre de bonnes pratiques de sécurité à travers des méthodes d'authentification robustes, des configurations de pare-feu adéquates et le déploiement de certificats SSL/TLS, tout en respectant les recommandations de sécurité émises par les autorités compétentes.

**4. Mettre l’infrastructure en production dans le cloud**

**Explication**: Cela consiste à déployer et gérer une infrastructure en production dans un environnement cloud, en utilisant les services et outils disponibles sur les plateformes cloud comme AWS, Azure, ou Google Cloud.

**Exemple**: Utiliser AWS CloudFormation pour déployer une pile d'infrastructure en production.

yaml

Copier le code

AWSTemplateFormatVersion: '2010-09-09'

Resources:

MyEC2Instance:

Type: 'AWS::EC2::Instance'

Properties:

InstanceType: 't2.micro'

ImageId: 'ami-0c55b159cbfafe1f0'

KeyName: 'my-key-pair'

SecurityGroups:

- !Ref MySecurityGroup

MySecurityGroup:

Type: 'AWS::EC2::SecurityGroup'

Properties:

GroupDescription: 'Enable SSH and HTTP access'

SecurityGroupIngress:

- IpProtocol: 'tcp'

FromPort: '22'

ToPort: '22'

CidrIp: '0.0.0.0/0'

- IpProtocol: 'tcp'

FromPort: '80'

ToPort: '80'

CidrIp: '0.0.0.0/0'

**Utilisation de ces compétences dans une formation DevOps**

**1. Automatiser la création de serveurs à l’aide de scripts**

**Formation**: Les étudiants apprennent à écrire des scripts Bash pour automatiser l'installation et la configuration de logiciels sur des serveurs. Par exemple, ils créent des scripts pour installer des serveurs web, des bases de données, ou des environnements de développement.

**2. Automatiser le déploiement d'une infrastructure**

**Formation**: Les étudiants utilisent des outils comme Terraform pour déployer des infrastructures complètes, y compris les réseaux, les instances de calcul, et les services de stockage. Ils apprennent à écrire des fichiers de configuration et à exécuter des commandes pour créer et gérer l'infrastructure de manière programmatique.

**3. Sécuriser l’infrastructure**

**Formation**: Les étudiants apprennent les meilleures pratiques de sécurité et utilisent des outils comme Ansible pour automatiser la configuration de la sécurité. Ils configurent des pare-feu, gèrent les accès, et mettent en place des systèmes de surveillance pour garantir la sécurité de l'infrastructure.

**4. Mettre l’infrastructure en production dans le cloud**

**Formation**: Les étudiants déploient des applications en production sur des plateformes cloud comme AWS, Azure, ou Google Cloud. Ils utilisent des outils natifs des fournisseurs de cloud pour gérer le déploiement, la mise à l'échelle, et la surveillance de leurs applications en production.

En intégrant ces compétences dans une formation DevOps, les étudiants acquièrent des compétences pratiques et pertinentes pour le marché du travail, leur permettant de gérer efficacement des environnements de développement et de production modernes.

**Connaissance des offres cloud (IaaS, PaaS et SaaS)**

**Explication**: Les offres cloud se divisent en trois catégories principales : IaaS (Infrastructure as a Service), PaaS (Platform as a Service) et SaaS (Software as a Service). Chacune de ces offres fournit un niveau différent de gestion et de contrôle des ressources informatiques.

1. **IaaS (Infrastructure as a Service)**:
   * **Explication**: IaaS fournit des ressources de base comme les serveurs, le stockage et les réseaux via internet. L'utilisateur a le contrôle sur le système d'exploitation, le stockage et les applications déployées.
   * **Exemple**: Amazon Web Services (AWS) EC2, Google Compute Engine, Microsoft Azure Virtual Machines.

python

Copier le code

# Exemple d'utilisation d'IaaS avec AWS EC2

import boto3

ec2 = boto3.resource('ec2')

instance = ec2.create\_instances(

ImageId='ami-0abcdef1234567890',

MinCount=1,

MaxCount=1,

InstanceType='t2.micro',

KeyName='my-key-pair'

)

print("Instance créée:", instance[0].id)

1. **PaaS (Platform as a Service)**:
   * **Explication**: PaaS offre une plateforme complète incluant le matériel, les logiciels et l'infrastructure sur laquelle les développeurs peuvent construire, tester et déployer des applications. L'utilisateur gère les applications et les données tandis que le fournisseur gère tout le reste.
   * **Exemple**: Google App Engine, AWS Elastic Beanstalk, Microsoft Azure App Services.

yaml

Copier le code

# Exemple de déploiement d'une application sur AWS Elastic Beanstalk avec un fichier de configuration .ebextensions

option\_settings:

aws:elasticbeanstalk:environment:proxy:staticfiles:

/static: static

1. **SaaS (Software as a Service)**:
   * **Explication**: SaaS fournit des applications logicielles complètes via internet. L'utilisateur n'a pas besoin de gérer l'infrastructure sous-jacente ni les plateformes sur lesquelles l'application est exécutée.
   * **Exemple**: Google Workspace, Salesforce, Microsoft Office 365.

html

Copier le code

<!-- Exemple d'intégration de SaaS avec Google Workspace -->

<iframe src="https://docs.google.com/document/d/your-doc-id/edit" width="100%" height="600px"></iframe>

**Connaissance des principes de la réversibilité**

**Explication**: La réversibilité est la capacité de migrer les données et les applications d'un fournisseur de services cloud vers un autre ou de les rapatrier dans son propre centre de données, sans perte de données et avec une interruption minimale des services. Cela permet d'éviter la dépendance à un fournisseur unique et d'assurer une flexibilité opérationnelle.

**Exemple**: Utilisation d'outils et de standards ouverts pour assurer la réversibilité.

* **Exemple 1**: Utilisation de formats de données standards comme JSON, XML ou CSV pour stocker les données, facilitant ainsi la migration vers d'autres systèmes.
* **Exemple 2**: Utilisation de Kubernetes pour orchestrer des conteneurs, permettant de déplacer facilement des applications entre différents fournisseurs cloud.

yaml

Copier le code

# Exemple de fichier de configuration Kubernetes pour déployer une application

apiVersion: apps/v1

kind: Deployment

metadata:

name: myapp-deployment

spec:

replicas: 3

selector:

matchLabels:

app: myapp

template:

metadata:

labels:

app: myapp

spec:

containers:

- name: myapp

image: myapp:latest

ports:

- containerPort: 80

**Connaissance de l’architecture d’un réseau cloud**

**Explication**: Comprendre l'architecture d'un réseau cloud implique de connaître les composants et les configurations nécessaires pour créer et gérer un réseau sécurisé et performant dans le cloud. Cela inclut les réseaux virtuels, les sous-réseaux, les routeurs, les pare-feu, les VPN et les configurations de sécurité.

**Exemple**: Utilisation de Microsoft Azure pour configurer un réseau virtuel (VNet).

hcl

Copier le code

# Exemple de configuration Terraform pour créer un réseau virtuel Azure

provider "azurerm" {

features {}

}

resource "azurerm\_virtual\_network" "example" {

name = "example-vnet"

address\_space = ["10.0.0.0/16"]

location = "East US"

resource\_group\_name = "example-resources"

}

resource "azurerm\_subnet" "internal" {

name = "internal"

resource\_group\_name = "example-resources"

virtual\_network\_name = azurerm\_virtual\_network.example.name

address\_prefixes = ["10.0.1.0/24"]

}

**Exemples pratiques pour un administrateur système DevOps**

1. **Connaissance des offres cloud (IaaS, PaaS et SaaS)**:
   * **Formation**: Les étudiants apprennent à choisir la bonne offre cloud en fonction des besoins spécifiques de leurs projets. Par exemple, utiliser IaaS pour des besoins de contrôle et de personnalisation, PaaS pour un développement rapide sans gestion de l'infrastructure, et SaaS pour des solutions logicielles prêtes à l'emploi.
   * **Exemple**: Déployer une application web sur AWS EC2 (IaaS), développer et déployer une application sur Google App Engine (PaaS), et intégrer Salesforce (SaaS) dans une solution d'entreprise.
2. **Connaissance des principes de la réversibilité**:
   * **Formation**: Les étudiants apprennent à utiliser des outils et des standards ouverts pour assurer la réversibilité. Ils pratiquent la migration des applications et des données entre différents environnements cloud.
   * **Exemple**: Déployer une application conteneurisée sur Kubernetes pour faciliter le déplacement entre AWS, Google Cloud et Azure. Utiliser des scripts d'exportation et d'importation de bases de données pour migrer les données de MySQL sur AWS RDS à MySQL sur Google Cloud SQL.
3. **Connaissance de l’architecture d’un réseau cloud**:
   * **Formation**: Les étudiants apprennent à configurer des réseaux virtuels, des sous-réseaux, des pare-feu et des VPN pour sécuriser et optimiser les réseaux cloud.
   * **Exemple**: Configurer un VNet dans Azure avec des sous-réseaux, des groupes de sécurité réseau (NSG) pour contrôler le trafic, et un VPN pour connecter un réseau local à un réseau cloud.

En maîtrisant ces compétences, un administrateur système DevOps est capable de gérer efficacement les ressources cloud, d'assurer la flexibilité opérationnelle grâce à la réversibilité, et de configurer des réseaux sécurisés et performants dans le cloud.

4o

**5. Préparer un environnement de test**

**Explication**: Préparer un environnement de test consiste à créer et configurer un espace où les applications peuvent être testées avant leur déploiement en production. Cela inclut la configuration de serveurs, de bases de données, et d'autres services nécessaires pour simuler l'environnement de production.

**Exemple**: Utilisation d'Ansible pour préparer un environnement de test.

yaml

Copier le code

---

- name: Préparer l'environnement de test

hosts: test\_servers

become: yes

tasks:

- name: Installer les dépendances

apt:

name: "{{ item }}"

state: present

with\_items:

- git

- docker.io

- docker-compose

- name: Cloner le dépôt de code

git:

repo: 'https://github.com/mon\_projet/test\_app.git'

dest: /var/www/test\_app

- name: Démarrer les conteneurs Docker

docker\_compose:

project\_src: /var/www/test\_app

state: present

Connaissance de la démarche DevOps Comprendre l’organisation des équipes de développement Comprendre la méthode Agile Connaissance de la démarche CI/CD :

**Connaissance de la démarche DevOps**

**Explication**: La démarche DevOps consiste à intégrer les processus de développement et d'opérations pour améliorer la collaboration entre les équipes de développement et de gestion des systèmes. Elle vise à automatiser et à optimiser les flux de travail, à renforcer la qualité des logiciels, et à accélérer les déploiements.

**Exemple**: Mise en place d'une pipeline DevOps pour une application web.

1. **Utilisation de Git pour la gestion du code source**:
   * Le code source est stocké dans un dépôt Git, par exemple sur GitHub.

bash

Copier le code

# Initialiser un dépôt Git

git init

# Ajouter des fichiers

git add .

# Commiter des changements

git commit -m "Initial commit"

# Pousser vers un dépôt distant

git remote add origin https://github.com/user/repo.git

git push -u origin master

1. **Automatisation des tests avec Jenkins**:
   * Configurer Jenkins pour exécuter des tests automatisés à chaque commit.

groovy

Copier le code

pipeline {

agent any

stages {

stage('Build') {

steps {

sh 'npm install'

}

}

stage('Test') {

steps {

sh 'npm test'

}

}

}

}

1. **Déploiement continu avec Kubernetes**:
   * Utiliser Kubernetes pour déployer automatiquement l'application après un build réussi.

yaml

Copier le code

apiVersion: apps/v1

kind: Deployment

metadata:

name: myapp

spec:

replicas: 3

selector:

matchLabels:

app: myapp

template:

metadata:

labels:

app: myapp

spec:

containers:

- name: myapp

image: myapp:latest

ports:

- containerPort: 80

**Comprendre l’organisation des équipes de développement**

**Explication**: Comprendre l'organisation des équipes de développement implique de savoir comment ces équipes sont structurées et comment elles collaborent pour développer des logiciels. Cela inclut la compréhension des rôles et responsabilités, des processus de communication, et des méthodes de gestion de projet.

**Exemple**: Organisation d'une équipe Agile avec des rôles définis.

1. **Product Owner (PO)**: Responsable de la définition des fonctionnalités et de la priorisation du backlog produit.
2. **Scrum Master**: Facilite le processus Scrum, assure que l'équipe suit les pratiques Agile, et aide à résoudre les obstacles.
3. **Développeurs**: Conçoivent, développent, testent et déploient le produit.
4. **Testeurs QA**: Vérifient la qualité du produit en effectuant des tests manuels et automatisés.

**Comprendre la méthode Agile**

**Explication**: La méthode Agile est une approche de gestion de projet qui favorise la flexibilité, la collaboration et l'amélioration continue. Elle est souvent utilisée dans le développement logiciel pour livrer des produits de haute qualité en cycles courts appelés sprints.

**Exemple**: Utilisation de Scrum pour gérer un projet de développement logiciel.

1. **Sprint Planning**:
   * L'équipe planifie le travail à réaliser pendant le sprint.

plaintext

Copier le code

Product Owner: "Pour ce sprint, nous allons nous concentrer sur l'implémentation de la fonctionnalité de recherche."

1. **Daily Standup**:
   * L'équipe se réunit quotidiennement pour discuter des progrès et des obstacles.

plaintext

Copier le code

Développeur 1: "Hier, j'ai terminé l'implémentation de l'algorithme de recherche. Aujourd'hui, je vais commencer les tests."

Développeur 2: "Je suis bloqué sur un bug avec la fonctionnalité de filtrage. J'ai besoin d'aide pour le résoudre."

1. **Sprint Review**:
   * À la fin du sprint, l'équipe présente le travail accompli aux parties prenantes.

plaintext

Copier le code

Product Owner: "Nous avons ajouté la fonctionnalité de recherche. Voici une démonstration de son fonctionnement."

1. **Sprint Retrospective**:
   * L'équipe réfléchit à ce qui a bien fonctionné et à ce qui pourrait être amélioré.

plaintext

Copier le code

Scrum Master: "Qu'est-ce qui s'est bien passé pendant ce sprint ?"

Développeur 1: "La communication entre les membres de l'équipe a été excellente."

Développeur 2: "Nous devrions améliorer notre processus de gestion des bugs."

**Connaissance de la démarche CI/CD**

**Explication**: La démarche CI/CD (Continuous Integration/Continuous Deployment) consiste à automatiser le processus de développement, de test et de déploiement des logiciels. L'intégration continue permet aux développeurs de fusionner leurs modifications de code régulièrement, et le déploiement continu assure que les modifications validées sont automatiquement déployées en production.

**Exemple**: Mise en place d'un pipeline CI/CD avec GitLab CI.

1. **Configuration du pipeline CI**:
   * Utiliser un fichier .gitlab-ci.yml pour définir les étapes du pipeline CI.

yaml

Copier le code

stages:

- build

- test

- deploy

build\_job:

stage: build

script:

- npm install

test\_job:

stage: test

script:

- npm test

deploy\_job:

stage: deploy

script:

- kubectl apply -f k8s/

only:

- master

1. **Automatisation des tests**:
   * Exécuter des tests automatisés pour chaque modification du code.

bash

Copier le code

# Exemple de script de test en Node.js

npm test

1. **Déploiement automatisé**:
   * Déployer automatiquement les modifications validées sur un cluster Kubernetes.

bash

Copier le code

# Appliquer la configuration Kubernetes

kubectl apply -f k8s/

**En résumé**

**Connaissance de la démarche DevOps**:

* **Exemple**: Utilisation de Jenkins pour l'intégration continue et Kubernetes pour le déploiement continu.

**Comprendre l’organisation des équipes de développement**:

* **Exemple**: Structuration d'une équipe Agile avec des rôles comme Product Owner, Scrum Master, et développeurs.

**Comprendre la méthode Agile**:

* **Exemple**: Gestion d'un projet de développement avec des pratiques Scrum telles que le sprint planning, le daily standup, le sprint review et le sprint retrospective.

**Connaissance de la démarche CI/CD**:

* **Exemple**: Configuration d'un pipeline CI/CD avec GitLab CI pour automatiser le processus de build, de test et de déploiement.

En comprenant et en appliquant ces compétences, un administrateur système DevOps peut améliorer la collaboration entre les équipes de développement et d'opérations, accélérer les cycles de déploiement, et assurer la qualité et la fiabilité des logiciels en production.

4o

**6. Gérer le stockage des données**

**Explication**: Gérer le stockage des données implique la mise en place et la maintenance des systèmes de stockage, tels que les bases de données, les systèmes de fichiers distribués, et les solutions de sauvegarde. Cela inclut également la gestion des accès et la sécurité des données.

**Exemple**: Utilisation de Terraform pour configurer une base de données RDS sur AWS.

hcl

Copier le code

provider "aws" {

region = "us-west-2"

}

resource "aws\_db\_instance" "mydb" {

allocated\_storage = 20

engine = "mysql"

engine\_version = "5.7"

instance\_class = "db.t2.micro"

name = "mydatabase"

username = "admin"

password = "mypassword"

parameter\_group\_name = "default.mysql5.7"

}

**Connaissance des différents types de stockage**

**Explication**: En tant qu'administrateur DevOps, il est crucial de comprendre les différents types de stockage de données utilisés dans les infrastructures modernes. Il existe principalement deux types de stockage de données : le stockage persistant et le stockage éphémère.

1. **Stockage Persistant**: Les données sont conservées même après la fin du processus ou du redémarrage du système.
   * **Exemples**: Disques durs (HDD), disques SSD, stockage en réseau (NAS), stockage dans le cloud (Amazon S3, Google Cloud Storage).
2. **Stockage Éphémère**: Les données sont temporaires et peuvent être perdues après l'arrêt du processus ou du redémarrage du système.
   * **Exemples**: RAM, stockage temporaire des conteneurs (volumes Docker non persistants).

**Connaissance des architectures de données SQL et NoSQL**

**Explication**: SQL (Structured Query Language) et NoSQL (Not only SQL) sont deux types de bases de données utilisés pour stocker et gérer les données. Chaque type a ses propres avantages et inconvénients.

1. **Bases de données SQL**:
   * **Caractéristiques**: Structurées, utilisent des tables, relations entre les données, requêtes SQL pour l'interaction.
   * **Exemples**: MySQL, PostgreSQL, Oracle Database, Microsoft SQL Server.
   * **Utilisation**: Idéal pour les applications nécessitant des transactions complexes et des relations entre les données.

sql

Copier le code

-- Exemple de requête SQL pour créer une table et insérer des données

CREATE TABLE utilisateurs (

id INT PRIMARY KEY,

nom VARCHAR(100),

email VARCHAR(100)

);

INSERT INTO utilisateurs (id, nom, email) VALUES (1, 'Jean Dupont', 'jean@example.com');

1. **Bases de données NoSQL**:
   * **Caractéristiques**: Flexibles, n'utilisent pas de schémas fixes, adaptées à des données non structurées.
   * **Exemples**: MongoDB (documents), Cassandra (colonnes), Redis (clé-valeur), Neo4j (graphes).
   * **Utilisation**: Idéal pour les applications nécessitant une scalabilité horizontale et des données hétérogènes.

javascript

Copier le code

// Exemple de document MongoDB

db.utilisateurs.insert({

id: 1,

nom: 'Jean Dupont',

email: 'jean@example.com'

});

**Connaissance des règles de sécurité (droits d’accès)**

**Explication**: Il est essentiel de sécuriser l'accès aux données en définissant des droits d'accès appropriés pour les utilisateurs et les services. Cela inclut la mise en place de rôles, de permissions et de politiques de sécurité pour protéger les données sensibles.

**Exemple**: Mise en place des droits d'accès pour une base de données MySQL.

sql

Copier le code

-- Création d'un utilisateur avec des permissions spécifiques

CREATE USER 'utilisateur'@'localhost' IDENTIFIED BY 'motdepasse';

-- Attribution des droits de lecture et écriture sur une base de données spécifique

GRANT SELECT, INSERT, UPDATE, DELETE ON ma\_base\_de\_donnees.\* TO 'utilisateur'@'localhost';

-- Appliquer les changements

FLUSH PRIVILEGES;

**Programmer la sauvegarde des données et mettre en place les droits d’accès, conformément à un cahier des charges**

**Explication**: Programmer des sauvegardes régulières et définir des droits d'accès en fonction des exigences spécifiées dans un cahier des charges est crucial pour assurer la continuité et la sécurité des données.

**Exemple**: Mise en place d'une tâche cron pour sauvegarder une base de données MySQL et gestion des accès avec des scripts.

bash

Copier le code

# Script de sauvegarde pour MySQL

#!/bin/bash

DATE=$(date +%F)

mysqldump -u root -p motdepasse ma\_base\_de\_donnees > /chemin/vers/les/sauvegardes/backup\_$DATE.sql

# Ajouter une tâche cron pour exécuter le script chaque jour à 2h du matin

0 2 \* \* \* /chemin/vers/le/script/sauvegarde.sh

**Contexte(s) professionnel(s) de mise en œuvre**

En tant qu'administrateur DevOps, les bases de données à mettre en service peuvent être de type SQL ou NoSQL, selon les besoins de l'application et les exigences de l'entreprise. Voici quelques exemples concrets de mise en œuvre dans un contexte professionnel.

**Exemple 1: Application e-commerce**:

* **Base de données SQL**: Utilisation de MySQL pour stocker les informations sur les produits, les utilisateurs et les commandes.
* **Base de données NoSQL**: Utilisation de Redis pour la mise en cache des sessions utilisateurs et pour améliorer les performances des requêtes fréquentes.

**Exemple 2: Application de réseaux sociaux**:

* **Base de données SQL**: Utilisation de PostgreSQL pour stocker les relations entre les utilisateurs et les messages.
* **Base de données NoSQL**: Utilisation de MongoDB pour stocker les profils d'utilisateurs et les publications avec des schémas flexibles.

**Exemple 3: Gestion des logs**:

* **Base de données NoSQL**: Utilisation d'Elasticsearch pour indexer et rechercher des logs en temps réel provenant de différentes sources de l'infrastructure.

En résumé, la connaissance des différents types de stockage, des architectures SQL et NoSQL, et des règles de sécurité est essentielle pour un administrateur DevOps. Programmer les sauvegardes et mettre en place les droits d'accès en fonction des exigences spécifiques permet de garantir la sécurité et la disponibilité des données dans un environnement professionnel.

**7. Gérer des conteneurs**

**Explication**: Gérer des conteneurs implique la création, le déploiement, la mise à jour et la suppression des conteneurs. Cela inclut également la gestion des orchestrateurs de conteneurs comme Kubernetes pour automatiser le déploiement, la mise à l'échelle et les opérations des conteneurs d'applications.

**Exemple**: Utilisation de Kubernetes pour déployer une application conteneurisée.

yaml

Copier le code

apiVersion: v1

kind: Pod

metadata:

name: mypod

labels:

app: myapp

spec:

containers:

- name: myapp-container

image: myapp:latest

ports:

- containerPort: 80

**2. Déployer en continu une application**

**Explication**: Déployer en continu consiste à automatiser le processus de déploiement de code dans un environnement de production ou de test après chaque modification validée dans le dépôt de code. Cela permet des déploiements rapides et fréquents.

**Exemple**: Utilisation de Jenkins pour configurer un pipeline de déploiement continu.

groovy

Copier le code

pipeline {

agent any

stages {

stage('Build') {

steps {

sh 'make build'

}

}

stage('Test') {

steps {

sh 'make test'

}

}

stage('Deploy') {

steps {

sh 'make deploy'

}

}

}

}

**Description de la compétence – processus de mise en œuvre**

La gestion des containers est une compétence essentielle pour un administrateur DevOps. Elle implique la création et la gestion de containers en tenant compte de l’architecture applicative. Cette compétence inclut la préparation du déploiement des applications, la connexion des containers au réseau et au stockage distant, ainsi que l'utilisation d'outils comme Docker pour gérer les mises à jour des containers.

**Processus de mise en œuvre**

1. **Créer des containers**:
   * **Étape**: Utiliser des images Docker pour créer des containers.
   * **Exemple**: Création d'un container à partir d'une image Docker officielle.

bash

Copier le code

docker run -d --name mon\_container -p 8080:80 nginx

Cette commande télécharge l'image nginx depuis Docker Hub et crée un container nommé mon\_container qui expose le port 80 du container sur le port 8080 de l'hôte.

1. **Définir les ressources nécessaires**:
   * **Étape**: Allouer des ressources spécifiques (CPU, mémoire, espace disque) à chaque container.
   * **Exemple**: Limiter l'utilisation de la mémoire et des CPU pour un container.

bash

Copier le code

docker run -d --name mon\_container\_limited --memory="256m" --cpus="1" nginx

Cette commande crée un container avec une limite de 256 Mo de mémoire et une utilisation maximale de 1 CPU.

1. **Connecter les containers au système hôte**:
   * **Étape**: Configurer les réseaux et les volumes pour les containers.
   * **Exemple**: Créer un réseau Docker et connecter un container à ce réseau.

bash

Copier le code

docker network create mon\_reseau

docker run -d --name mon\_container --network mon\_reseau nginx

Cette commande crée un réseau nommé mon\_reseau et connecte le container mon\_container à ce réseau.

1. **Automatiser la création des containers**:
   * **Étape**: Utiliser des outils comme Docker Compose pour définir et gérer des applications multi-containers.
   * **Exemple**: Fichier docker-compose.yml pour une application web avec une base de données.

yaml

Copier le code

version: '3'

services:

web:

image: nginx

ports:

- "8080:80"

networks:

- mon\_reseau

db:

image: mysql

environment:

MYSQL\_ROOT\_PASSWORD: example

networks:

- mon\_reseau

networks:

mon\_reseau:

Pour lancer les services définis dans docker-compose.yml, utilisez:

bash

Copier le code

docker-compose up -d

1. **Gérer les mises à jour applicatives avec des containers**:
   * **Étape**: Utiliser Docker pour déployer de nouvelles versions d'applications en mettant à jour les containers concernés.
   * **Exemple**: Mise à jour d'une application web en remplaçant le container avec une nouvelle image.

bash

Copier le code

docker pull mon\_application:latest

docker stop mon\_container

docker rm mon\_container

docker run -d --name mon\_container -p 8080:80 mon\_application:latest

Cette séquence de commandes met à jour l'application en récupérant la dernière image, arrêtant le container existant, le supprimant et en lançant un nouveau container avec l'image mise à jour.

**Contexte(s) professionnel(s) de mise en œuvre**

Ces compétences sont essentielles lors de la préparation de la mise en production d'une nouvelle application ou lors de la mise à jour d'une application existante. En tant qu'administrateur DevOps, vous serez amené à gérer les ressources, à assurer la continuité du service et à maintenir des environnements de production stables et performants.

**Savoir-faire techniques**

* **Utiliser des images pour créer des containers**: Connaissance des registres d'images Docker (ex: Docker Hub, registre privé).
* **Définir les ressources nécessaires**: Capacité à allouer efficacement les ressources systèmes aux containers.
* **Connecter le container au système hôte**: Maîtrise de la configuration réseau et des volumes Docker.
* **Automatiser la création des containers**: Utilisation d'outils comme Docker Compose, Kubernetes pour l'orchestration des containers.
* **Utiliser les containers pour gérer les mises à jour applicatives**: Techniques de déploiement continu et gestion des versions avec Docker.

**Savoir-faire organisationnels**

* **Planification et coordination**: Préparation des déploiements en collaboration avec les équipes de développement et de QA.
* **Documentation**: Maintien de documents détaillant les processus de déploiement et de mise à jour.
* **Gestion des environnements**: Configuration et gestion des environnements de développement, de test et de production.

**Savoir-faire relationnels**

* **Communication**: Interaction efficace avec les équipes de développement pour comprendre les besoins et résoudre les problèmes.
* **Formation**: Capacité à former les équipes sur l'utilisation des containers et des outils associés.
* **Collaboration**: Travail en équipe pour assurer une intégration continue et un déploiement fluide des applications.

En résumé, la gestion des containers est une compétence clé pour un administrateur DevOps, impliquant la création, la configuration et la gestion des containers pour déployer et mettre à jour les applications de manière efficace et sécurisée.

**8. Automatiser la mise en production d’une application avec une plateforme**

**Explication**: Automatiser la mise en production consiste à utiliser des outils et des scripts pour automatiser le déploiement des applications en production, en minimisant les erreurs humaines et en augmentant l'efficacité. Les plateformes CI/CD comme Jenkins, GitLab CI, ou CircleCI sont souvent utilisées pour cela.

**Exemple**: Utilisation de GitLab CI/CD pour automatiser la mise en production.

yaml

Copier le code

stages:

- build

- test

- deploy

build\_job:

stage: build

script:

- echo "Building the application..."

- ./build\_script.sh

test\_job:

stage: test

script:

- echo "Running tests..."

- ./test\_script.sh

deploy\_job:

stage: deploy

script:

- echo "Deploying to production..."

- ./deploy\_script.sh

only:

- master

**Automatiser la mise en production d’une application avec une plateforme**

**Description de la compétence – processus de mise en œuvre**

Cette compétence concerne l'automatisation du processus de déploiement d'une application sur des environnements de pré-production et de production à l'aide d'une plateforme d'orchestration, notamment Kubernetes. Voici les étapes clés du processus :

1. **Préparation de l'environnement de pré-production**:
   * Lorsque les premiers tests de l’application sont satisfaisants, créer un environnement de pré-production similaire à celui de production pour tester l'application dans des conditions réelles.
2. **Déploiement et test de l'application en pré-production**:
   * Déployer l'application dans l'environnement de pré-production et effectuer des tests pour identifier les éventuels dysfonctionnements.
   * Remonter les erreurs aux développeurs pour les corriger et améliorer l'application.
3. **Déploiement de l'application en production**:
   * Une fois les tests en pré-production satisfaisants, déployer l'application dans l'environnement de production pour la rendre disponible aux utilisateurs finaux.

**Contexte(s) professionnel(s) de mise en œuvre**

Ces opérations s'appliquent à la mise en production d'une nouvelle application ainsi qu'à ses évolutions successives. L'utilisation d'une plateforme d'orchestration comme Kubernetes est essentielle pour automatiser et gérer efficacement le déploiement des applications.

**Critères de performance**

* L'environnement de pré-production est conforme à celui de production, assurant des tests réalistes.
* Les dysfonctionnements identifiés en pré-production sont remontés aux développeurs pour correction.
* Les containers de l'application sont décrits et gérés sur la plateforme d'orchestration.
* L'application ou sa mise à jour est déployée avec succès en production, assurant sa disponibilité aux utilisateurs.

**Exemple d'application dans un contexte professionnel**

Prenons l'exemple d'une équipe de développement travaillant sur une application de commerce électronique. Après avoir développé de nouvelles fonctionnalités, l'équipe souhaite les déployer en production de manière automatisée.

1. **Préparation de l'environnement de pré-production**:
   * Utilisation de Kubernetes pour créer un cluster de pré-production, reproduisant fidèlement l'environnement de production avec des répliques des services et des données.
2. **Déploiement et test de l'application en pré-production**:
   * Utilisation de flux de déploiement continu (CI/CD) pour déployer automatiquement l'application dans le cluster de pré-production après chaque nouvelle version.
   * Les tests automatisés sont exécutés pour vérifier le bon fonctionnement de l'application, et les erreurs détectées sont signalées aux développeurs pour correction.
3. **Déploiement de l'application en production**:
   * Une fois les tests en pré-production validés, l'application est déployée automatiquement dans le cluster de production, offrant ainsi aux utilisateurs finaux les nouvelles fonctionnalités sans interruption de service.

En résumé, l'automatisation du processus de déploiement avec une plateforme d'orchestration comme Kubernetes permet aux équipes DevOps de déployer rapidement et efficacement des applications tout en garantissant la stabilité et la disponibilité du service pour les utilisateurs finaux.

**Explications et Utilisations en Formation DevOps**

1. **Préparer un environnement de test**:
   * **Formation**: Les étudiants apprennent à configurer des environnements de test identiques à l'environnement de production pour garantir que les tests sont pertinents et fiables. Ils utilisent des outils comme Ansible ou Vagrant pour automatiser cette configuration.
2. **Gérer le stockage des données**:
   * **Formation**: Les étudiants apprennent à configurer et à gérer différentes solutions de stockage de données, en mettant l'accent sur la sécurité et la sauvegarde. Ils utilisent des outils comme Terraform pour gérer les bases de données dans le cloud.
3. **Gérer des conteneurs**:
   * **Formation**: Les étudiants apprennent à utiliser Docker pour créer et gérer des conteneurs, ainsi que Kubernetes pour orchestrer des applications conteneurisées. Ils acquièrent des compétences pratiques en déployant des applications sur des clusters Kubernetes.
4. **Déployer en continu une application**:
   * **Formation**: Les étudiants apprennent à configurer des pipelines CI/CD pour automatiser le déploiement continu des applications. Ils utilisent des outils comme Jenkins, GitLab CI ou CircleCI pour créer des pipelines de déploiement robustes.
5. **Automatiser la mise en production d’une application avec une plateforme**:
   * **Formation**: Les étudiants apprennent à utiliser des plateformes CI/CD pour automatiser la mise en production des applications. Ils configurent des pipelines pour déployer des applications en production de manière automatique et sécurisée.

En intégrant ces compétences dans une formation DevOps, les étudiants sont préparés à gérer des environnements complexes et à automatiser des processus critiques, ce qui est essentiel pour une gestion efficace et moderne des infrastructures et des applications.

**Définir et mettre en place des statistiques de services**

**Explication**: Définir et mettre en place des statistiques de services consiste à collecter, analyser et présenter des données de performance et de disponibilité des services déployés. Cela permet de surveiller l'état des services et d'identifier les éventuels problèmes avant qu'ils n'affectent les utilisateurs.

**Exemple**: Utilisation de Prometheus et Grafana pour collecter et afficher des statistiques de services.

1. **Installer Prometheus**:
   * Télécharger et exécuter Prometheus.

bash

Copier le code

# Télécharger Prometheus

wget https://github.com/prometheus/prometheus/releases/download/v2.26.0/prometheus-2.26.0.linux-amd64.tar.gz

tar xvf prometheus-2.26.0.linux-amd64.tar.gz

cd prometheus-2.26.0.linux-amd64

# Exécuter Prometheus

./prometheus --config.file=prometheus.yml

1. **Configurer un job de scraping dans prometheus.yml**:

yaml

Copier le code

global:

scrape\_interval: 15s

scrape\_configs:

- job\_name: 'node\_exporter'

static\_configs:

- targets: ['localhost:9100']

1. **Installer et configurer Grafana**:
   * Télécharger et installer Grafana.
   * Ajouter Prometheus comme source de données et créer des tableaux de bord pour visualiser les métriques collectées.

**Exploiter une solution de supervision**

**Explication**: Exploiter une solution de supervision consiste à utiliser des outils pour surveiller l'état, les performances et la disponibilité des infrastructures et des services. Cela implique de configurer des alertes et des notifications pour informer les équipes en cas de problèmes.

**Exemple**: Utilisation de Zabbix pour superviser une infrastructure.

1. **Installer Zabbix Server**:
   * Suivre les instructions d'installation de Zabbix pour le système d'exploitation choisi.
2. **Ajouter des hôtes à surveiller**:
   * Connecter des serveurs à Zabbix en installant l'agent Zabbix sur chaque serveur et en configurant Zabbix Server pour surveiller ces agents.

bash

Copier le code

# Installer l'agent Zabbix sur un serveur Linux

sudo apt-get install zabbix-agent

sudo systemctl start zabbix-agent

sudo systemctl enable zabbix-agent

1. **Configurer des éléments de surveillance**:
   * Créer des éléments (items) dans Zabbix pour collecter des métriques spécifiques comme l'utilisation CPU, la mémoire, l'espace disque, etc.
2. **Configurer des alertes et des notifications**:
   * Définir des seuils et des actions à prendre lorsqu'un seuil est dépassé, comme envoyer un email ou un message Slack à l'équipe de support.

**Superviser les services déployés**

**Explication**: Superviser les services déployés consiste à surveiller en temps réel l'état des services pour s'assurer qu'ils fonctionnent correctement. Cela inclut la surveillance des logs, des performances, et des incidents de sécurité, ainsi que la mise en place de solutions pour prévenir et résoudre les problèmes.

**Exemple**: Utilisation d'Elastic Stack (ELK) pour superviser les services déployés.

1. **Installer Elasticsearch, Logstash, et Kibana**:
   * Suivre les instructions d'installation pour chaque composant de la stack ELK.
2. **Configurer Logstash pour collecter les logs**:
   * Créer un fichier de configuration Logstash pour ingérer des logs de différentes sources.

bash

Copier le code

input {

file {

path => "/var/log/nginx/access.log"

start\_position => "beginning"

}

}

filter {

grok {

match => { "message" => "%{COMMONAPACHELOG}" }

}

}

output {

elasticsearch {

hosts => ["localhost:9200"]

index => "nginx-logs-%{+YYYY.MM.dd}"

}

}

1. **Utiliser Kibana pour visualiser et analyser les logs**:
   * Configurer des tableaux de bord dans Kibana pour afficher les logs et les métriques de performance.
   * Définir des alertes pour des anomalies ou des erreurs spécifiques.

**Utilisation de ces compétences en tant qu'Admin Système DevOps**

1. **Définir et mettre en place des statistiques de services**:
   * **Formation**: Les étudiants apprennent à utiliser des outils comme Prometheus et Grafana pour collecter et visualiser des métriques de performance. Ils configurent des job de scraping pour récupérer des métriques de différents services et créent des tableaux de bord pour visualiser les données en temps réel.
2. **Exploiter une solution de supervision**:
   * **Formation**: Les étudiants apprennent à installer et configurer des outils de supervision comme Zabbix. Ils ajoutent des hôtes à surveiller, configurent des éléments de surveillance pour collecter des métriques, et définissent des alertes pour notifier l'équipe en cas de problèmes.
3. **Superviser les services déployés**:
   * **Formation**: Les étudiants apprennent à utiliser des solutions de supervision comme Elastic Stack (ELK) pour collecter et analyser des logs. Ils configurent Logstash pour ingérer des logs, utilisent Kibana pour visualiser les données, et définissent des alertes pour détecter des anomalies ou des erreurs.

En intégrant ces compétences dans une formation DevOps, les étudiants sont préparés à surveiller efficacement les infrastructures et les services, à collecter et analyser des métriques de performance, et à réagir rapidement aux problèmes, ce qui est essentiel pour assurer la fiabilité et la performance des systèmes en production.

4o

9. REAC – explique : FICHE COMPÉTENCE PROFESSIONNELLE N° 9

**Critères de performance et Définition d'indicateurs**

**Critères de performance**

Les critères de performance définissent les standards et les mesures utilisées pour évaluer la performance globale d'un système, d'une application ou d'un service. Ces critères sont essentiels pour garantir la qualité et la fiabilité des systèmes informatiques.

**Exemples d'indicateurs**

1. **Pertinence des indicateurs**:
   * Mesure de la pertinence des indicateurs choisis pour évaluer la performance d'un système ou d'un service. Par exemple, le taux de disponibilité du service est-il un indicateur pertinent pour mesurer la performance d'un site web ?
2. **Conformité aux SLA (Service Level Agreements)**:
   * Vérification si les indicateurs définis sont en accord avec les accords de niveau de service convenus avec les clients ou les utilisateurs. Par exemple, un SLA peut stipuler que le temps de réponse du serveur doit être inférieur à 500 ms.
3. **Exhaustivité des indicateurs**:
   * Assurer que tous les aspects critiques du système sont couverts par les indicateurs. Cela inclut les performances, la sécurité, la disponibilité, etc.

**Savoir-faire techniques, organisationnels, relationnels et savoirs**

**Définition des indicateurs techniques**:

* Savoir identifier et définir les indicateurs techniques pertinents pour évaluer les performances et l'utilisation des ressources des systèmes informatiques. Exemples :
  + Utilisation du CPU
  + Utilisation du stockage
  + Utilisation du réseau

**Définition des indicateurs de sécurité**:

* Capacité à définir des indicateurs liés à la sécurité des systèmes, y compris les mesures d'accès et de détection d'intrusion. Exemples :
  + Nombre de tentatives de connexion infructueuses
  + Nombre d'attaques détectées

**Définition des indicateurs de performance des applications**:

* Compétence à définir des indicateurs pour évaluer les performances des applications, telles que le temps de réponse, la disponibilité, etc. Exemples :
  + Temps de chargement des pages web
  + Nombre de transactions par seconde

**Exemple d'application dans un contexte professionnel**

Prenons l'exemple d'une équipe DevOps responsable de la gestion d'un service de messagerie en ligne. Voici comment ils pourraient définir des indicateurs de performance :

1. **Pertinence des indicateurs**:
   * Mesure de la pertinence des indicateurs de performance comme le temps de réponse du serveur de messagerie, le nombre de messages envoyés par utilisateur, etc.
2. **Conformité aux SLA**:
   * Vérification si les indicateurs définis sont en accord avec les SLA établis avec les clients. Par exemple, le SLA peut garantir un temps de réponse moyen inférieur à 500 ms.
3. **Exhaustivité des indicateurs**:
   * Assurer que tous les aspects critiques du service de messagerie sont couverts par les indicateurs, y compris la sécurité, la disponibilité, etc.

En résumé, la définition d'indicateurs de performance pertinents et conformes aux SLA est essentielle pour évaluer et garantir la qualité et la fiabilité des systèmes informatiques dans un environnement professionnel.

10.FICHE COMPÉTENCE PROFESSIONNELLE N° 10 Exploiter une solution de supervision-

**Savoir-faire techniques, organisationnels, relationnels et savoirs en DevOps**

**Savoir-faire techniques**

1. **Choisir et installer un outil de supervision**:
   * Capacité à évaluer les besoins de supervision et à sélectionner l'outil le plus approprié en fonction des exigences techniques et des contraintes budgétaires. Exemple : Choix de Prometheus pour la supervision des métriques d'une architecture conteneurisée.
2. **Installer et configurer une console de supervision**:
   * Compétence à déployer et à configurer une interface utilisateur pour surveiller les performances et l'état de l'infrastructure. Exemple : Configuration de Grafana pour visualiser les données collectées par Prometheus.
3. **Déployer des agents de collecte de données**:
   * Maîtrise de l'installation et de la configuration d'agents de collecte sur l'ensemble des nœuds de l'infrastructure pour collecter les données de surveillance. Exemple : Installation de l'agent Node Exporter pour collecter les métriques système sur chaque serveur.

**Savoir-faire organisationnels**

1. **S'assurer que les transactions sont sécurisées**:
   * Capacité à garantir la sécurité des données transitant à travers les outils de supervision, en utilisant des protocoles cryptés et des mécanismes d'authentification robustes. Exemple : Configuration de TLS pour chiffrer les communications entre les agents de collecte et la console de supervision.
2. **Surveiller la remontée des indicateurs**:
   * Compétence à surveiller activement les indicateurs de performance et à réagir rapidement en cas de dysfonctionnement ou de dégradation des services. Exemple : Configuration d'alertes dans Prometheus pour être informé en cas de dépassement de seuils critiques.

**Savoir-faire relationnels**

1. **Dialoguer avec les fournisseurs de service cloud**:
   * Capacité à collaborer avec les fournisseurs de services cloud pour obtenir des informations sur les métriques de performance et pour résoudre les problèmes liés à l'infrastructure hébergée dans le cloud. Exemple : Communication avec le support technique d'AWS pour résoudre un problème de connectivité réseau.

**Savoirs**

1. **Connaissance de SNMP**:
   * Compréhension du protocole SNMP (Simple Network Management Protocol) utilisé pour la collecte et le monitoring des informations sur les périphériques réseau. Exemple : Configuration d'un agent SNMP sur un routeur pour collecter des statistiques sur le trafic réseau.
2. **Connaissance de syslog**:
   * Compréhension du protocole syslog utilisé pour la journalisation des événements système. Exemple : Configuration d'un serveur syslog pour centraliser les journaux de plusieurs serveurs dans un environnement distribué.

En résumé, les compétences en DevOps nécessitent à la fois des connaissances techniques solides pour déployer et configurer des outils de supervision, des compétences organisationnelles pour assurer la sécurité et la disponibilité des systèmes surveillés, des compétences relationnelles pour collaborer avec les fournisseurs de services, et une connaissance approfondie des protocoles de supervision comme SNMP et syslog.

REAC table

Activités types :

1 Automatiser le déploiement d’une infrastructure dans le cloud

Compétences professionnelles : 1. Automatiser la création de serveurs à l’aide de scripts

1. Automatiser le déploiement d'une infrastructure
2. Sécuriser l’infrastructure et Mettre l’infrastructure en production dans le cloud.
3. Copilot : **Automatiser la création de serveurs à l’aide de scripts :**
   * Cette compétence consiste à créer des scripts pour automatiser la création de serveurs. Vous pouvez utiliser des outils comme Terraform, Ansible ou CloudFormation pour définir l’infrastructure en tant que code (IaC).
   * **Exemple :** Vous écrivez un script Terraform qui définit un serveur EC2 sur AWS. Lorsque vous exécutez ce script, il crée automatiquement le serveur avec les spécifications que vous avez définies.
4. **Automatiser le déploiement d’une infrastructure :**
   * Cette compétence implique l’automatisation du déploiement d’une infrastructure complète, y compris les serveurs, les bases de données, les réseaux, etc.
   * **Exemple :** Vous utilisez Ansible pour déployer une application web. Ansible configure automatiquement les serveurs, installe les dépendances et déploie l’application.
5. **Sécuriser l’infrastructure :**
   * Pour sécuriser l’infrastructure, vous devez mettre en place des pare-feux, des certificats SSL, des groupes de sécurité, etc.
   * **Exemple :** Vous configurez un groupe de sécurité sur AWS pour autoriser uniquement le trafic HTTP et HTTPS vers vos serveurs.
6. **Mettre l’infrastructure en production dans le cloud :**
   * Cette compétence consiste à déployer l’infrastructure dans un environnement de production, prêt à être utilisé par les utilisateurs finaux.
   * **Exemple :** Vous déployez votre application sur un serveur EC2 dans AWS, configurez un nom de domaine et mettez en place des mécanismes de surveillance pour garantir la disponibilité et les performances.

OBJECTIVE de Stage basé sur Schéma Ciblé de HAS

Cible : mettre en place solution de CI/CD, conteneurisation, outils pour logs et tracking sur 2 applis de la HAS.

 Cibles techniques (parce que l'on a la main dessus) :

* Pacte (possiblement une  PreProd);
* SIAM PreProd

Etape initiale : proposer les solutions technos (pourquoi, les +, les -, plusieurs possibles, pourquoi ce choix), planning / scénario de mise en œuvre + ressources nécessaires en interne pour aider à monter les briques technos et les plateformes.

ARCHITECTURE HAS - Schéma Ciblé

Une image contenant texte, capture d’écran, logiciel, Page web

Description générée automatiquement

Une image contenant Appareils électroniques, texte, capture d’écran, ordinateur

Description générée automatiquement