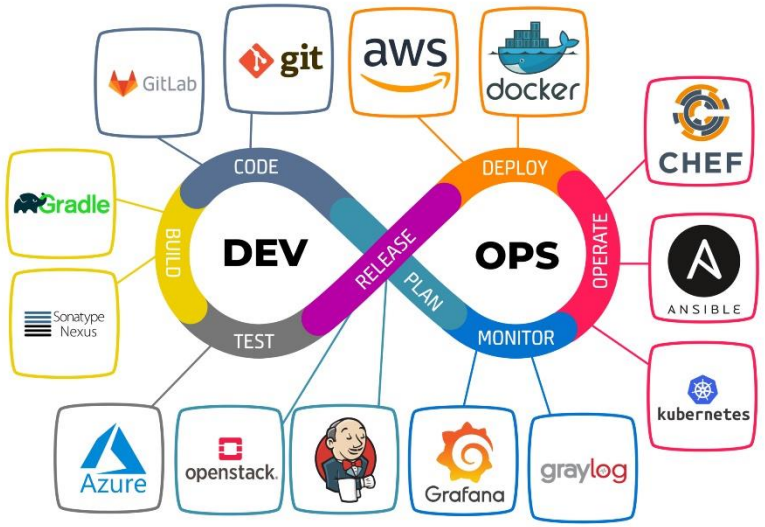
INITIATION AU DEVOPS

CODE :B3COM0105 **Durée** : 20H



Ing. BOGNI-DANCHI T.

OBJECTIFS PEDAGOGIQUES

Prérequis : ***Connaissances de base en administration système et développement logiciel***

Ce cours a pour objectif d'initier les étudiants aux concepts et pratiques de DevOps. Les étudiants apprendront à combiner le développement logiciel et l'administration système pour améliorer la collaboration entre les équipes et automatiser les processus de livraison continue. Le cours couvre les outils, les processus et les méthodologies utilisés dans un environnement DevOps.

À l'issue de ce cours, l'étudiant doit être capable de :

- Comprendre les principes fondamentaux de DevOps et leur rôle dans le développement logiciel.
- Utiliser des outils de gestion de versions, d'intégration continue et de déploiement continu.
- Automatiser les tests et les déploiements à l'aide d'outils comme Jenkins, Docker, et Kubernetes.
- Mettre en place des pipelines CI/CD (Continuous Integration/Continuous Delivery) pour optimiser les processus de développement.

PLAN

- SECTION4: Orchestration et gestion des configuration

  - Architecture Kubernetes
  - Orchestration de conteneurs
  - Concepts : pods, deployments, services
  - Notion sur Ansible
  - Travaux Pratiques (3h)
    - ✓ Installation d'un cluster Kubernetes
    - ✓ Installation et configuration de Ansible
    - ✓ Déploiement d'applications
    - ✓ Mise à l'échelle
    - ✓ Gestion des mises à jour
    - ✓ Configuration de services
    - ✓ Gestion des configurations

- SECTION5: PROJET FINAL (Par l'étudiant)

SECTION 4 : ORCHESTRATION AVEC KUBERNETESE ET GESTION DES CONFIGURATIONS AVEC ANSIBLE

I- ORCHESTRATION AVEC KUBERNETES

I-1- Introduction

- Origine de kubernetes

- **Création par Google** :
  Kubernetes a été initialement développé par **Google** en 2014.
  - ✓ Google possède une expertise significative dans la gestion des conteneurs, ayant utilisé une technologie interne appelée **Borg** pendant plus de dix ans pour orchestrer des millions de conteneurs dans ses propres centres de données.
  - ✓ Kubernetes est l'évolution de cette expérience, conçue pour être open-source et accessible à toute la communauté technologique.
- **Transition vers la CNCF** :
  En 2015, Google a transféré Kubernetes à la **Cloud Native Computing Foundation (CNCF)** , une organisation qui promet l'adoption d'outils natifs pour le cloud.
  Cela a renforcé la communauté open-source et a permis à Kubernetes de devenir un standard dans l'orchestration de conteneurs.
- **Nom et logo** :
  - ✓ **Nom "Kubernetes"** : Il provient du mot grec **κυβερνήτης (kubernetes)** , qui signifie **"pilote"** ou **"gouvernail"** , symbolisant son rôle dans la direction et la gestion des conteneurs.
  - ✓ **Logo** : Une roue de gouvernail avec sept branches, représentant les sept développeurs principaux à l'origine du projet.

SECTION 4 : ORCHESTRATION AVEC KUBERNETESE ET GESTION DES CONFIGURATIONS AVEC ANSIBLE

I-1- Introduction

Kubernetes est conçu pour répondre aux défis liés à la gestion des conteneurs à grande échelle. Voici ses principales fonctionnalités et concepts.

Pourquoi Kubernetes ?

Avant Kubernetes, la gestion des conteneurs (comme ceux créés avec Docker) était principalement manuelle :

- Difficile de déployer et de mettre à jour des applications de manière cohérente.
- Complexe de gérer la haute disponibilité et la tolérance aux pannes.
- Limitations pour la mise à l'échelle automatique.

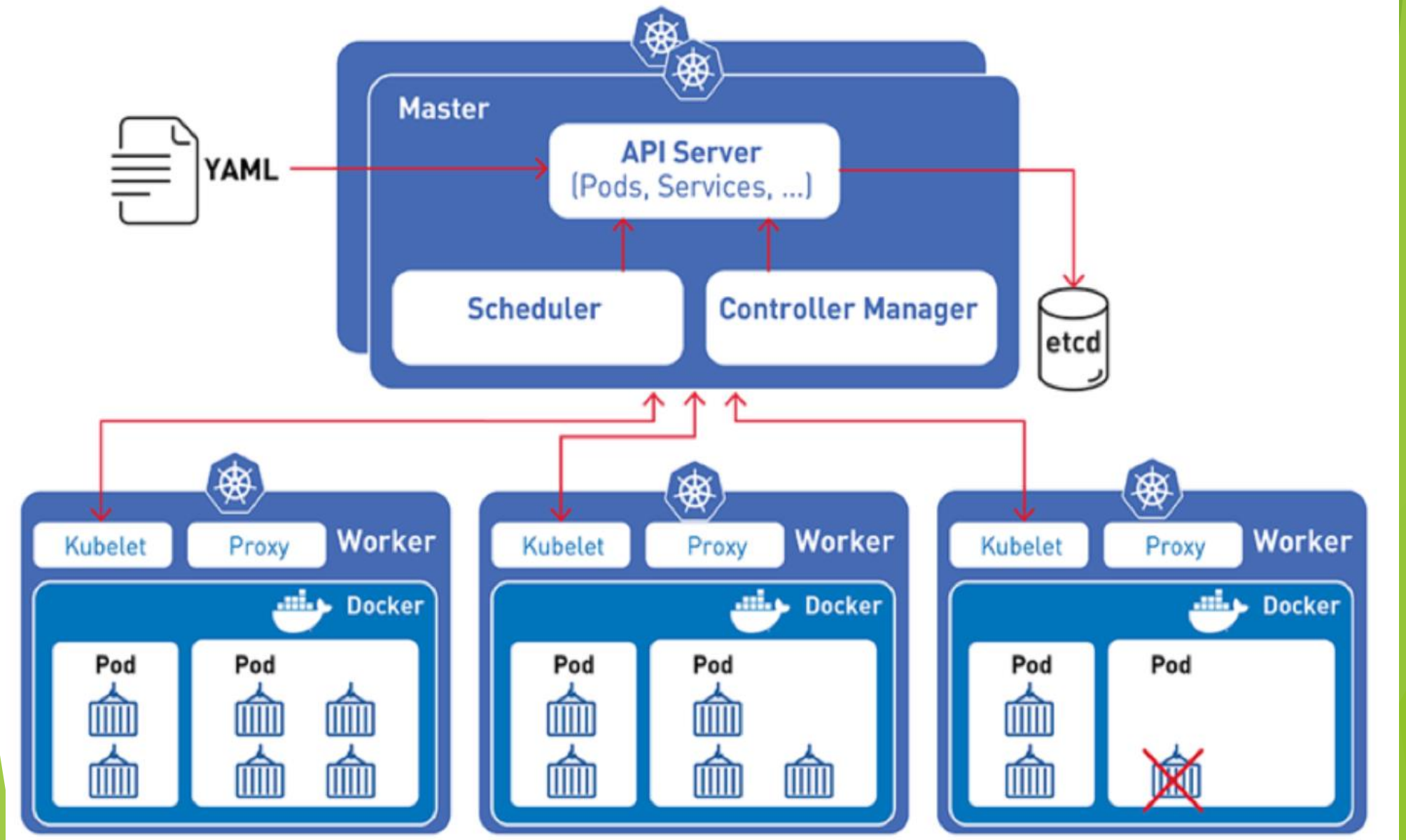
Kubernetes automatise ces tâches et permet de :

- **Orchestrer les conteneurs** sur des clusters de machines.
- **Automatiser le déploiement** et les mises à jour des applications.
- **Répartir la charge** pour optimiser les ressources.
- **Récupérer automatiquement** en cas de panne

Principales fonctionnalités :

- **Mise à l'échelle automatique** : Kubernetes ajuste le nombre de conteneurs en fonction de la charge.
- **Gestion des défauts** : Redémarrage des conteneurs défaillants et remplacement des nœuds non fonctionnels.
- **Réseautage** : Fournit des adresses IP aux conteneurs et connecte automatiquement les Pods.
- **Stockage persistant** : Gère les données des conteneurs à l'aide de solutions comme NFS, EBS ou GCP Persistent Disks.
- **Observabilité** : Surveillez les applications et collectez des métriques pour le diagnostic.

I-2- Architecture de Kubernetes



I-2- Architecture de Kubernetes

L'architecture de Kubernetes repose sur une organisation en **maître-esclave** pour coordonner les tâches.

1.Nœud Master (Plan de Contrôle) :

1. Superviser l'ensemble du cluster.
2. Planifiez les Pods sur les nœuds de travail.
3. Composants clés : **API Server** , **Scheduler** , **Controller Manager** , **etcd** .

2.Nœud worker:

1. Exécuter les applications conteneurisées dans des **Pods** .
2. Composants clés : **Kubelet** , **Kube-Proxy** , et le moteur de conteneurisation (comme Docker).

Cette architecture permet essentiellement de rassembler les machines en un **cluster unique** sur lequel on peut faire tourner des **“charges de calcul” (workloads)** très diverses.
Sur un tel cluster le déploiement d’un workload prend la forme de **ressources (objets k8s)** qu’on **décrit sous forme de code** et qu’on crée ensuite effectivement via l’API Kubernetes.
Pour uniformiser les déploiement logiciel Kubernetes est basé sur le standard des **conteneurs** (défini aujourd’hui sous le nom **Container Runtime Interface**, Docker est l’implémentation la plus connue).
Plutôt que de déployer directement des conteneurs, Kubernetes crée des **aggrégats de un ou plusieurs conteneurs** appelés des **Pods**. Les pods sont donc l’unité de base de Kubernetes.

SECTION 4 : ORCHESTRATION AVEC KUBERNETESE ET GESTION DES CONFIGURATIONS AVEC ANSIBLE

I-3- Les composants de Kubernetes

```
Control Plane:
   ├── API Server
   ├── etcd
   ├── Scheduler
   ├── Controller Manager
   └── Cloud Controller Manager (optionnel)

Worker Node:
   ├── Kubelet
   ├── Kube-Proxy
   └── Runtime de Conteneur (Docker, containerd, etc.)

Objets Kubernetes:
   ├── Pods
   ├── Services
   ├── Deployments
   └── Namespaces
```

1. Composants du Plan de Contrôle (Control Plane)

Ces composants supervisent le cluster et prennent les décisions clés telles que le déploiement et la mise à l'échelle des applications.

API Server

- Point d'entrée principal pour toutes les interactions avec le cluster.
- Expose l'API RESTful que les utilisateurs et composants internes utilisent pour communiquer.
- Valide les requêtes et les transmet à d'autres composants du plan de contrôle,

etcd

•Base de données clé-valeur distribuée qui stocke toutes les informations de configuration du cluster.

•Permet de garantir la **consistance et la disponibilité** des données dans le cluster.

•Contient l'état actuel et désiré des ressources Kubernetes (Pods, Deployments, etc.).

I-3- Les composants de Kubernetes

Controller Manager

•Regroupe plusieurs contrôleurs qui supervisent l'état des objets Kubernetes pour s'assurer qu'ils correspondent à l'état désiré.

•Exemples de contrôleurs :

- **Node Controller** : Surveille la santé des nœuds.
- **Replication Controller** : Gère le nombre de répliques de Pods en fonctionnement.
- **Service Controller** : Supervise les services et leur intégration avec le cloud provider.

Scheduler

•Responsable de l'attribution des Pods aux nœuds disponibles.

•Prend en compte :

- Les ressources disponibles sur chaque nœud (CPU, mémoire).
- Les contraintes définies dans les fichiers de déploiement (affinité, tolérances).

Cloud Controller Manager (optionnel)

•Interagit avec les fournisseurs de cloud pour gérer les ressources spécifiques au cloud (Load Balancers, volumes de stockage, etc.).

2. Composants des Nœuds de Travail (Worker Nodes)

Ces composants assurent l'exécution des applications conteneurisées et la gestion des ressources nécessaires.

I-3- Les composants de Kubernetes

Kubelet

•Agent qui s'exécute sur chaque nœud de travail.
•Assure que les conteneurs des Pods sont lancés correctement et conformes à leurs spécifications.
•Communique avec le serveur API pour recevoir des instructions et rapporter l'état des Pods.

Kube-Proxy

•Composant réseau qui s'exécute sur chaque nœud de travail.
•Maintient les règles réseau pour exposer les services Kubernetes.
•Répartit le trafic entre les Pods associés à un service.

Runtime de Conteneur

•Responsable de l'exécution des conteneurs.
•Kubernetes supporte plusieurs runtimes, tels que :

- **Docker**
- **containerd**
- **CRI-O**

3. Objets Kubernetes Associés

Bien que non directement des composants, les objets Kubernetes jouent un rôle clé dans l'orchestration :

Pod

•Unité de base de Kubernetes.
•Contient un ou plusieurs conteneurs qui partagent le même réseau et stockage.

SECTION 4 : ORCHESTRATION AVEC KUBERNETESE ET GESTION DES CONFIGURATIONS AVEC ANSIBLE

I-3- Les composants de Kubernetes

Service

•Expose les Pods à un réseau interne ou externe.

•Fournit un point d'accès stable, même si les Pods changent.

Namespace

•Segmente les ressources Kubernetes en espaces logiques pour mieux organiser le cluster.

Deployment

•Gère le déploiement et la mise à jour des Pods.

4. Add-ons et Extensions

Ces outils améliorent les capacités natives de Kubernetes.

DNS CoreDNS

•Fournit des services de résolution DNS dans le cluster.

•Permet aux applications de communiquer via des noms DNS.

Dashboard Kubernetes

•Interface utilisateur graphique pour surveiller et gérer le cluster.

Outils de surveillance

•**Prometheus** et **Grafana** pour la collecte et la visualisation des métriques.

Ces composants fonctionnent ensemble pour assurer :

•Une gestion cohérente des ressources du cluster.

•Une orchestration optimale des applications conteneurisées.

•Une résilience et une évolutivité adaptées aux besoins des environnements modernes.

Kubernetes est devenu un outil clé pour l'automatisation et la gestion d'infrastructures conteneurisées.

SECTION 4 : ORCHESTRATION AVEC KUBERNETESE ET GESTION DES CONFIGURATIONS AVEC ANSIBLE

I-4- Les Commandes Kubernetes

1. Commandes de Gestion des Ressources

Création

•kubectl create : Créer une ressource
•kubectl apply -f <fichier> : Créer/Mettre à jour à partir d'un fichier YAML
•kubectl run : Démarrer un pod

Lecture

•kubectl get: Lister les ressources
- •kubectl get pods
- •kubectl get deployments
- •kubectl get services
- •kubectl get nodes
- •kubectl get namespaces

Description Détaillée

•kubectl describe : Afficher les détails d'une ressource
- •kubectl describe pod <nom>
- •kubectl describe deployment <nom>

Suppression

•kubectl delete : Supprimer des ressources
- •kubectl delete pod <nom>
- •kubectl delete deployment <nom>
- •kubectl delete -f <fichier.yaml>

I-4- Les Commandes Kubernetes

2. Commandes de Déploiement

•kubectl create deployment : Créer un déploiement
•kubectl scale deployment : Mise à l'échelle
•kubectl rollout : Gestion des versions
    •kubectl rollout status deployment/<nom>
    •kubectl rollout history deployment/<nom>
    •kubectl rollout undo deployment/<nom>

3. Commandes d'Interaction avec les Pods

•kubectl exec : Exécuter une commande dans un pod
    •kubectl exec -it <pod> -- /bin/bash
•kubectl logs : Afficher les logs
    •kubectl logs <pod>
    •kubectl logs -f <pod> (suivi en temps réel)

4. Commandes de Configuration

•kubectl config : Gérer les configurations du cluster
    •kubectl config view
    •kubectl config current-context
    •kubectl config use-context <contexte>

I-4- Les Commandes Kubernetes

5. Commandes de Réseau et Services

- kubectl expose : Exposer un déploiement comme service
- kubectl port-forward : Redirection de port
  - kubectl port-forward service/<nom> 8080:80

6. Commandes de Gestion des Clusters

- kubectl cluster-info : Informations sur le cluster
- kubectl top : Utilisation des ressources
  - kubectl top nodes
  - kubectl top pods

7. Commandes Avancées

- kubectl patch : Mise à jour partielle d'une ressource
- kubectl edit : Éditer une ressource en direct
- kubectl label : Ajouter/Modifier des labels
- kubectl annotate : Gérer les annotations

I-4- Les Commandes Kubernetes

8. Commandes de Gestion des Volumes

- kubectl volume : Opérations sur les volumes

9. Commandes de Sécurité

- kubectl auth : Vérification des permissions
  - kubectl auth can-i create deployments

```
# Exemple de cmd kubernetes
#Créer un déploiement
kubectl create deployment nginx --image=nginx

# Exposer le déploiement
kubectl expose deployment nginx --port=80 --
type=LoadBalancer

# Mettre à l'échelle
kubectl scale deployment nginx --replicas=3

# Vérifier le statut
kubectl get deployments
kubectl get pods
kubectl get services
```

I-5- Le fichier YAML dans Kubernetes

En Kubernetes, les fichiers **YAML** sont utilisés pour définir les configurations des ressources du cluster. Ces fichiers jouent un rôle clé dans la gestion déclarative des ressources, permettant aux administrateurs et aux développeurs de créer, mettre à jour, et supprimer des objets dans un cluster Kubernetes.
Voici la structure des fichiers YAML dans Kubernetes.

```
apiVersion: <version de l'API>
kind: <type de la ressource>
metadata:
  name: <nom de l'objet>
  namespace: <namespace (facultatif)>
spec:
  <spécifications de la ressource>
```

- **apiVersion** : Indique la version de l'API Kubernetes utilisée pour la ressource (par exemple, v1, apps/v1, etc.).
- **kind** : Spécifie le type de ressource à créer (par exemple, Pod, Deployment, Service, etc.).
- **metadata** : Contient les informations sur la ressource, comme son nom et son namespace.
- **spec** : Définit la configuration spécifique de l'objet, selon le type de ressource

I-4- Le fichier YAML dans Kubernetes

Fichier YAML pour un Pod
Un Pod est la plus petite unité déployable dans Kubernetes, contenant un ou plusieurs conteneurs.

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
  labels:
    app: my-app
spec:
  containers:
  - name: nginx-container
    image: nginx:1.21
    ports:
    - containerPort: 80
```

Explications :
•**metadata.name** : Nom unique du Pod.
•**spec.containers** : Liste des conteneurs dans le Pod.
•**image** : Image Docker utilisée pour le conteneur.
•**ports.containerPort** : Port sur lequel le conteneur écoute.

I-4- Le fichier YAML dans Kubernetes

Fichier YAML pour un Deployment

Un Deployment gère la mise à l'échelle, le déploiement, et les mises à jour des Pods.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-deployment
  labels:
    app: my-app
spec:
  replicas: 3
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
      - name: nginx-container
        image: nginx:1.21
        ports:
        - containerPort: 80
```

Explications :

•**replicas** : Nombre de répliques de Pods souhaitées.

•**template** : Définit la configuration des Pods déployés par ce Deployment.

•**selector.matchLabels** : Associe le Deployment aux Pods ayant ce label.

SECTION 4 : ORCHESTRATION AVEC KUBERNETESE ET GESTION DES CONFIGURATIONS AVEC ANSIBLE

II- ANSIBLE

II-1- Introduction

Ansible est un moteur d'automatisation informatique open source qui automatise le provisionnement, la gestion de la configuration, le déploiement des applications, l'orchestration et de nombreux autres processus informatiques. Son utilisation est gratuite et le projet bénéficie de l'expérience et de l'intelligence de ses milliers de contributeurs.

Red Hat® Ansible Automation Platform regroupe plus d'une douzaine de projets en amont dans une plateforme d'entreprise unifiée et renforcée en termes de sécurité pour l'automatisation des missions critiques. Elle s'appuie sur les fondations du projet open source pour créer une expérience d'automatisation de bout en bout pour les équipes interfonctionnelles.

Ansible fournit une automatisation open source qui réduit la complexité et s'exécute partout. L'utilisation d'Ansible vous permet d'automatiser pratiquement n'importe quelle tâche. Voici quelques cas d'utilisation courants d'Ansible :

- Éliminez les répétitions et simplifiez les flux de travail
- Gérer et maintenir la configuration du système
- Déployer en continu des logiciels complexes
- Effectuez des mises à jour progressives sans interruption de service

Ansible utilise des scripts simples et lisibles par l'homme, appelés playbooks, pour automatiser vos tâches. Vous déclarez l'état souhaité d'un système local ou distant dans votre playbook. Ansible s'assure que le système reste dans cet état.

SECTION 4 : ORCHESTRATION AVEC KUBERNETESE ET GESTION DES CONFIGURATIONS AVEC ANSIBLE

II-1- Introduction

En tant que technologie d'automatisation, Ansible est conçue autour des principes suivants :

- Architecture sans agent

Faible frais de maintenance en évitant l'installation de logiciels supplémentaires sur l'infrastructure informatique.

- Simplicité

Les manuels d'automatisation utilisent une syntaxe YAML simple pour le code qui se lit comme une documentation. Ansible est également décentralisé, utilisant SSH avec les informations d'identification du système d'exploitation existantes pour accéder aux machines distantes.

- Évolutivité et flexibilité

Faites évoluer facilement et rapidement les systèmes que vous automatisez grâce à une conception modulaire qui prend en charge une large gamme de systèmes d'exploitation, de plates-formes cloud et de périphériques réseau.

- Idempotence et prévisibilité

Lorsque le système est dans l'état décrit par votre playbook, Ansible ne change rien, même si le playbook s'exécute plusieurs fois.

II-2- Architecture de Ansible

Ansible est un moteur d'automatisation informatique radicalement simple qui automatise le provisionnement du cloud, la gestion de la configuration, le déploiement d'applications, l'orchestration intra-service et de nombreux autres besoins informatiques.

II-2- Architecture de Ansible

Conçu pour les déploiements à plusieurs niveaux dès le premier jour, Ansible modélise votre infrastructure informatique en décrivant comment tous vos systèmes interagissent, plutôt que de simplement gérer un seul système à la fois.

Il n'utilise aucun agent ni aucune infrastructure de sécurité personnalisée supplémentaire, il est donc facile à déployer - et surtout, il utilise un langage très simple (YAML, sous la forme d'Ansible Playbooks) qui vous permet de décrire vos tâches d'automatisation d'une manière qui se rapproche de l'anglais simple. Cette architecture permet à Ansible d'organiser son système au tour des composantes suivantes:

- Modules
- Module utilities
- Plugins
- Inventory
- Playbooks
- The Ansible search path

- Modules

Ansible fonctionne en se connectant à vos nœuds et en leur envoyant des scripts appelés « modules Ansible ». La plupart des modules acceptent des paramètres qui décrivent l'état souhaité du système. Ansible exécute ensuite ces modules (via SSH par défaut) et les supprime une fois l'opération terminée. Votre bibliothèque de modules peut résider sur n'importe quelle machine, et aucun serveur, démon ou base de données n'est requis.

II-2- Architecture de Ansible

Vous pouvez écrire vos propres modules , mais vous devez d'abord vous demander si vous devez le faire . En général, vous travaillerez avec votre programme de terminal préféré, un éditeur de texte et probablement un système de contrôle de version pour suivre les modifications apportées à votre contenu. Vous pouvez écrire des modules spécialisés dans n'importe quel langage pouvant renvoyer du JSON (Ruby, Python, bash, etc.).

- Module utilities

Lorsque plusieurs modules utilisent le même code, Ansible stocke ces fonctions sous forme d'utilitaires de module afin de minimiser la duplication et la maintenance. Par exemple, le code qui analyse les URL est lib/ansible/module_utils/url.py. Vous pouvez également écrire vos propres utilitaires de module . Les utilitaires de module ne peuvent être écrits qu'en Python ou en PowerShell.

- Plugins

Les plugins augmentent les fonctionnalités de base d'Ansible. Alors que les modules s'exécutent sur le système cible dans des processus distincts (généralement sur un système distant), les plugins s'exécutent sur le nœud de contrôle au sein du/usr/bin/ansibleprocessus. Les plugins offrent des options et des extensions pour les fonctionnalités de base d'Ansible : transformation des données, journalisation des sorties, connexion à l'inventaire, etc. Ansible est livré avec un certain nombre de plugins pratiques, et vous pouvez facilement écrire le vôtre . Par exemple, vous pouvez écrire un plugin d'inventaire pour vous connecter à n'importe quelle source de données qui renvoie du JSON. Les plugins doivent être écrits en Python

II-2- Architecture de Ansible

- Inventory

Par défaut, Ansible représente les machines qu'il gère dans un fichier (INI, YAML, etc.) qui place toutes vos machines gérées dans des groupes de votre choix.

Pour ajouter de nouvelles machines, aucun serveur de signature SSL supplémentaire n'est impliqué, il n'y a donc jamais de problème pour décider pourquoi une machine particulière n'a pas été connectée en raison de problèmes NTP ou DNS obscurs.

Si votre infrastructure contient une autre source de vérité, Ansible peut également s'y connecter. Ansible peut extraire des informations d'inventaire, de groupe et de variable à partir de sources telles qu'EC2, Rackspace, OpenStack, etc.

- Playbooks

Les playbooks peuvent orchestrer finement plusieurs tranches de la topologie de votre infrastructure, avec un contrôle très détaillé sur le nombre de machines à traiter à la fois. C'est là qu'Ansible commence à devenir le plus intéressant.

L'approche d'Ansible en matière d'orchestration est d'une simplicité affinée, car nous pensons que votre code d'automatisation devrait avoir un sens parfait pour vous dans les années à venir et qu'il ne devrait y avoir que très peu de choses à retenir sur la syntaxe ou les fonctionnalités spéciales.

Exemple de fichier yml d’un playbook

II-2- Architecture de Ansible

```
---
- hosts: webservers
  serial: 5 # update 5 machines at a time
  roles:
  - common
  - webapp

- hosts: content_servers
  roles:
  - common
  - content
```

- The Ansible search path

Les modules, les utilitaires de module, les plugins, les playbooks et les rôles peuvent résider à plusieurs emplacements. Si vous écrivez votre propre code pour étendre les fonctionnalités principales d'Ansible, vous pouvez avoir plusieurs fichiers avec des noms similaires ou identiques à différents emplacements sur votre nœud de contrôle Ansible. Le chemin de recherche détermine lesquels de ces fichiers Ansible découvrira et utilisera lors de l'exécution d'un playbook donné

II-3- Le playbook

Création d'un playbook

Les playbooks sont des plans d'automatisation, au YAMLformat, qu'Ansible utilise pour déployer et configurer des nœuds gérés.

Playbook

Une liste de pièces qui définissent l'ordre dans lequel Ansible exécute les opérations, de haut en bas, pour atteindre un objectif global.

Play

Une liste ordonnée de tâches mappées sur des nœuds gérés dans un inventaire.

Task

Une référence à un module unique qui définit les opérations effectuées par Ansible.

Module

Unité de code ou binaire qu'Ansible exécute sur des nœuds gérés. Les modules Ansible sont regroupés dans des collections avec un nom de collection complet (FQCN) pour chaque module.

Suivez les étapes suivantes pour créer un playbook qui envoie des pings à vos hôtes et imprime un message « Hello world » :

1.Créez un fichier nommé playbook.yamldans votre ansible_quickstartrépertoire, que vous avez créé précédemment, avec le contenu suivant :

1. Créez un fichier nommé playbook.yaml dans votre ansible_quickstart répertoire, que vous avez créé précédemment, avec le contenu suivant :

```
- name: My first play
  hosts: myhosts
  tasks:
   - name: Ping my hosts
     ansible.builtin.ping:

   - name: Print message
     ansible.builtin.debug:
       msg: Hello world
```

2. Exécutez votre manuel de jeu.

ansible-playbook -i inventory.ini playbook.yaml

SECTION 4 : ORCHESTRATION AVEC KUBERNETESE ET GESTION DES CONFIGURATIONS AVEC ANSIBLE

II-3- Le playbook

Création d'un playbook

Les playbooks sont des plans d'automatisation, au YAMLformat, qu'Ansible utilise pour déployer et configurer des nœuds gérés.

Playbook

Une liste de pièces qui définissent l'ordre dans lequel Ansible exécute les opérations, de haut en bas, pour atteindre un objectif global.

Play

Une liste ordonnée de tâches mappées sur des nœuds gérés dans un inventaire.

Task

Une référence à un module unique qui définit les opérations effectuées par Ansible.

Module

Unité de code ou binaire qu'Ansible exécute sur des nœuds gérés. Les modules Ansible sont regroupés dans des collections avec un nom de collection complet (FQCN) pour chaque module.

Suivez les étapes suivantes pour créer un playbook qui envoie des pings à vos hôtes et imprime un message « Hello world » :

1.Créez un fichier nommé playbook.yamldans votre ansible_quickstartrépertoire, que vous avez créé précédemment, avec le contenu suivant :

1. Créez un fichier nommé playbook.yaml dans votre ansible_quickstart répertoire, que vous avez créé précédemment, avec le contenu suivant :

```
- name: My first play
  hosts: myhosts
  tasks:
   - name: Ping my hosts
     ansible.builtin.ping:

   - name: Print message
     ansible.builtin.debug:
       msg: Hello world
```

2. Exécutez votre manuel de jeu.

ansible-playbook -i inventory.ini playbook.yaml

SECTION 4 : ORCHESTRATION AVEC KUBERNETESE ET GESTION DES CONFIGURATIONS AVEC ANSIBLE

II-4- Les commandes Ansible

1. Commandes de Base

1.1 Tester la Connectivité

- **Commande** : *ansible all -m ping*
  - **all** : Désigne tous les hôtes de l'inventaire.
  - **-m ping** : Utilise le module ping pour tester la connectivité avec les hôtes cibles.

1.2 Exécuter une Commande Ad-hoc

- **Commande** : *ansible all -m shell -a "uptime"*
  - **-m shell** : Utilise le module shell pour exécuter une commande shell.
  - **-a** : Spécifie l'argument de la commande, ici "uptime".

2. Commandes Liées au Playbook

2.1 Exécuter un Playbook

- **Commande** : *ansible-playbook -i inventory site.yml*
  - **-i inventory** : Spécifie le fichier d'inventaire.
  - **site.yml** : Nom du playbook à exécuter.

2.2 Exécuter un Playbook en Mode Vérification (Dry Run)

- **Commande** : *ansible-playbook -i inventory site.yml --check*
  - **--check** : Simule les changements sans les appliquer.

2.3 Afficher la Sortie Plus Détaillée

- **Commande** : *ansible-playbook -i inventory site.yml -v*
  - **-v** : Affiche une sortie détaillée. Ajouter plus de v (-vvv) pour encore plus de détails.

SECTION 4 : ORCHESTRATION AVEC KUBERNETESE ET GESTION DES CONFIGURATIONS AVEC ANSIBLE

II-4- Les commandes Ansible

3. Commandes Liées à l'Inventaire

3.1 Lister les Hôtes dans un Inventaire

- **Commande** : *ansible-inventory --list -i inventory*
  - **--list** : Affiche tous les hôtes et groupes définis dans l'inventaire.

3.2 Afficher les Groupes d'un Inventaire

- **Commande** : *ansible all --list-hosts -i inventory*
  - **--list-hosts** : Affiche la liste des hôtes sous la cible spécifiée (all ici).

4. Commandes pour la Gestion des Modules

4.1 Rechercher un Module

- **Commande** : *ansible-doc -l*
  - **-l** : Liste tous les modules disponibles.

4.2 Obtenir des Informations sur un Module

- **Commande** : *ansible-doc <nom_du_module>*
  - Exemple : *ansible-doc yum*

5. Gestion de Configuration à Distance

5.1 Installer un Paquet

- **Commande** : *ansible web -m yum -a "name=httpd state=present"*
  - **web** : Groupe défini dans l'inventaire.
  - **-m yum** : Utilise le module yum (gestion des paquets sur les systèmes basés sur RPM).
  - **-a** : Spécifie l'action, ici installer httpd.

SECTION 4 : ORCHESTRATION AVEC KUBERNETESE ET GESTION DES CONFIGURATIONS AVEC ANSIBLE

II-4- Les commandes Ansible

5.2 Copier un Fichier vers une Machine Cible

- **Commande** : *ansible all -m copy -a "src=/local/file dest=/remote/path"*
  - **-m copy** : Utilise le module copy pour copier un fichier.

5.3 Redémarrer un Service

- **Commande** : *ansible web -m service -a "name=httpd state=restarted"*
  - **-m service** : Utilise le module service pour gérer les services.
  - **state=restarted** : Redémarre le service spécifié.

6. Débogage et Journalisation

6.1 Vérifier la Syntaxe d'un Playbook

- **Commande** : *ansible-playbook site.yml --syntax-check*

6.2 Afficher les Faits d'un Hôte (Facts)

- **Commande** *: ansible all -m gather_facts*
  - Permet de récupérer des informations détaillées sur un hôte (comme le système d'exploitation, le CPU, etc.).

6.3 Afficher Seulement un Sous-ensemble des Faits

- **Commande** : *ansible all -m setup -a "filter=ansible_distribution*"*
  - Utilise le module setup pour filtrer des faits spécifiques.

SECTION 4 : ORCHESTRATION AVEC KUBERNETESE ET GESTION DES CONFIGURATIONS AVEC ANSIBLE

II-4- Les commandes Ansible

7. Gestion des Secrets avec Ansible Vault

7.1 Créer un Fichier Chiffré

- **Commande** : *ansible-vault create secret.yml*
  - Crée un fichier YAML protégé par mot de passe.

7.2 Chiffrer un Fichier

- **Commande** : *ansible-vault encrypt site.yml*

7.3 Déchiffrer un Fichier

- **Commande** : *ansible-vault decrypt site.yml*

7.4 Modifier un Fichier Chiffré

- **Commande** : *ansible-vault edit secret.yml*

8. Commandes Avancées

8.1 Limiter l'Exécution à un Hôte ou un Groupe

- **Commande** : *ansible-playbook -i inventory site.yml --limit web*
  - **--limit web** : Exécute uniquement pour le groupe ou hôte web.

8.2 Utiliser un Fichier de Variables

- **Commande** : *ansible-playbook -i inventory site.yml --extra-vars "@variables.json"*
  - **--extra-vars** : Permet d'injecter des variables depuis un fichier externe.
  - .

SECTION 4 : ORCHESTRATION AVEC KUBERNETESE ET GESTION DES CONFIGURATIONS AVEC ANSIBLE

II-4- Les commandes Ansible

7. Gestion des Secrets avec Ansible Vault

7.1 Créer un Fichier Chiffré

- **Commande** : *ansible-vault create secret.yml*
  - Crée un fichier YAML protégé par mot de passe.

7.2 Chiffrer un Fichier

- **Commande** : *ansible-vault encrypt site.yml*

7.3 Déchiffrer un Fichier

- **Commande** : *ansible-vault decrypt site.yml*

7.4 Modifier un Fichier Chiffré

- **Commande** : *ansible-vault edit secret.yml*

8. Commandes Avancées

8.1 Limiter l'Exécution à un Hôte ou un Groupe

- **Commande** : *ansible-playbook -i inventory site.yml --limit web*
  - **--limit web** : Exécute uniquement pour le groupe ou hôte web.

8.2 Utiliser un Fichier de Variables

- **Commande** : *ansible-playbook -i inventory site.yml --extra-vars "@variables.json"*
  - **--extra-vars** : Permet d'injecter des variables depuis un fichier externe.
  - .

SECTION 4 : ORCHESTRATION AVEC KUBERNETESE ET GESTION DES CONFIGURATIONS AVEC ANSIBLE

III- TP

- TP 4.1 Installation
Installation de minikube et Ansible

- TP 4.2 : pod
Creation d un pod sous kubernetes

- TP 4.3 : deployment, services, replicat
Creation d'un deployement, service, replicat

- TP 4.4 : webapp
Deployement de notre application webapp sous un cluster kubernetes: 8180

TP 4.5 : wordpress
Deploiement de wordpress avec kubernetes

TP 4.6 : wordpress
Deploiement de wordpress avec kubernetes via un playbook ansible

REFERENCES BIBLIOGRAPHIQUES

Documentation officielle Docker : https://docs.docker.com/
Hub Docker (images Docker) : https://hub.docker.com/
Blog officiel Docker : https://www.docker.com/blog/
Tutoriels et guides Docker: https://www.digitalocean.com/community/tutorials /tag/docker/
Aide - **mémoire** Docker : https://dockerlabs.collabnix.com/docker/cheatsheet/
Référence API Docker : https://docs.docker.com/engine/api/
Jouer avec Docker (sandbox) : https://labs.play-with-docker.com/
GitHub officiel de Docker : https ://github .com/docker
Documentation de Docker **Compose** : https://docs.docker.com/compose/
kubernetes official documentation : https://kubernetes.io/docs/home/
deprecated api component : https://kubernetes.io/docs/reference/using-api/deprecation-guide/
kubernetes api group : https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.28/
kubernetes example code : https://github.com/kubernetes/examples
kubectl cli : https://kubernetes.io/docs/reference/generated/kubectl/kubectl-commands
https://kubespray.io
jenkins official doc : https://www.jenkins.io/doc/tutorials/
ansible official doc : https://docs.ansible.com/
https://devopssec.fr/
Gene Kim, Patrick Debois, John Willis, *The DevOps Handbook*, IT Revolution Press, 2016.
Len Bass, *DevOps: A Software Architect's Perspective*, Addison-Wesley, 2015.
James Turnbull, *The Docker Book: Containerization is the New Virtualization*, 2014.

Fin

Par Thomas BOGNI-DANCHI