

Devops

dimanche 23 juin 2024 13:50

Modèle en Cascade

- 1- Planification
- 2- Conception
- 3- Implémentation
- 4- Vérification
- 5- Maintenance

Agile

- 1- Définition
- 2- Conception
- 3- Développement
- 4- Démonstration
- 5- Si Oui : Production
- 6- Si Non : on revient à la phase 1

Les Défis rencontrée :

- La taille des projets : Million Lignes Of Code
- Contribution à des projets existants
- Réutilisation de code existant
- Collaboration avec d'autres développeurs
- Utilisateurs ou Client

DEVOPS :

- Ensemble de **techniques** et **d'outils** qui facilitent le passage de la phase **de développement à la production**
- Relation entre le **DEV**(développeur : modification et **réalisation à moindre coût**) et **OPS** (équipes de la **production** des produits : **stabilité** du système , **qualité**)
- **Devops <=> l'automatisation des processus du développement jusqu'à la production(agile)**

CI/CD:

- **Intégration Continu :**

- Intégration automatique du code après modification
- Test automatique des modules intégrée après chaque modification
- Détection rapide des erreurs après modification
- Livraison Continu :
 - Déploiement fiable et rapide des modifications testées
 - Réduction des délais entre développement et production
 - Amélioration et vérification continu avec les besoins du client
- Déploiement Continu
 - Automatisation totale du processus de mise en production
 - Réduction des erreurs et interventions humaines
 - Mise à jour en temps réel et Agilité maximale

Etapas du DevOps :

- 1- Plan
- 2- Code
- 3- Build : génération
- 4- Test
- 5- Release : production des versions
- 6- Deploy : intégration des versions dans l'environnement
- 7- Operate : version opérationnelle
- 8- Monitor : suivi et surveillance

Azure Bords : outil de **planification**

- **Suivi du travail** effectué avec des **tableaux Kanban** , des **backlogs** interactifs

Concepts

Boards

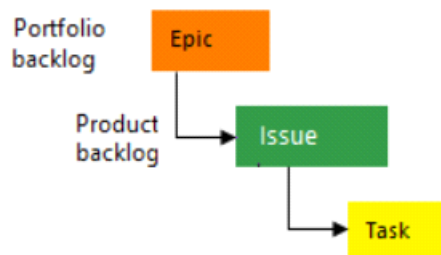
Work item : Epic , task , bug ,userstory

Backlog

Sprint

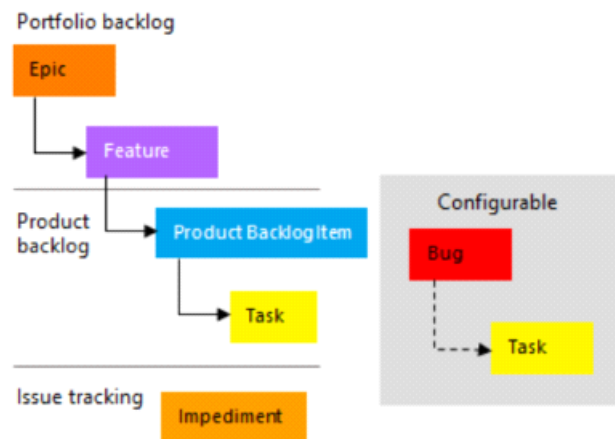
Work item Process

- **Basic**



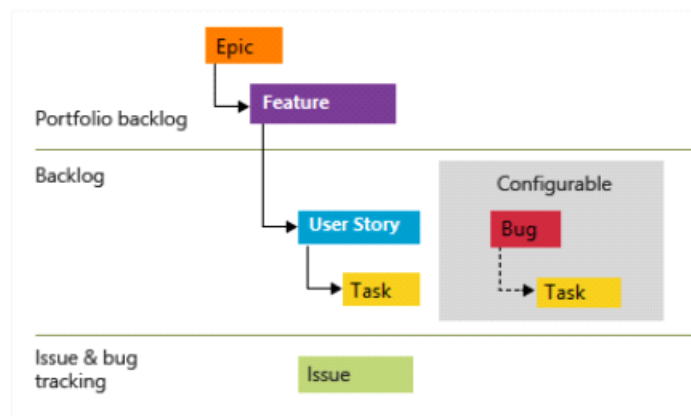
- **Scrum**

- **productBacklogItem<=>userstory**
- **Impediment <=> issue**



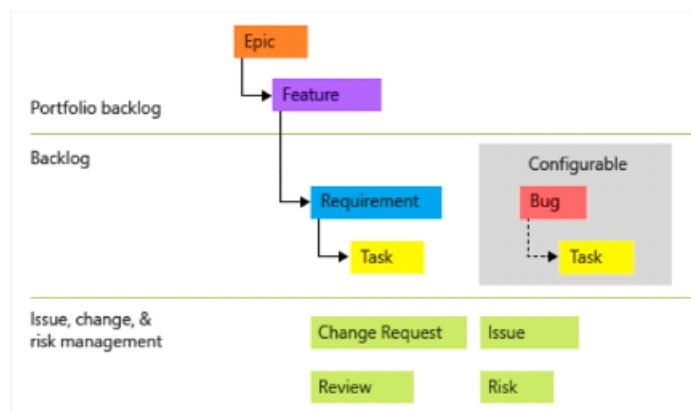
- Agile

▪ userstory



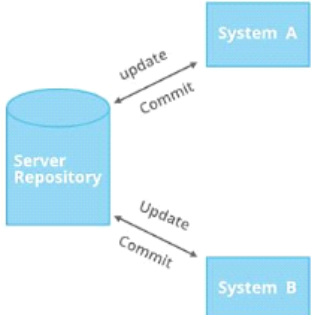
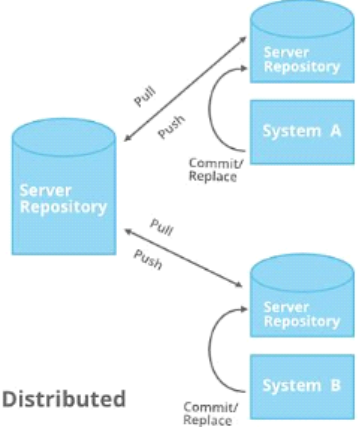
- CMMI

- Requirement
- Risk management



Contrôleur de code Source :

- offre un **système de gestion de version décentralisé**
- Permet le **suivi** et la **gestion** des modifications **du code**
- **permet le retour à une version antérieure**
- **Garde un historique**
- Permet de résoudre les conflits

SCCS Centralisé	SCCS Distribué
Tout l'historique des changements est conservé sur un serveur central (distant)	Chaque usager à une copie locale de tout l'historique (avantages : espace privé - Hors ligne , inconvénient : dépôt centrale de sauvegarde) la copie locale est mis à jour par le serveur distant
 <p>Centralized</p>	 <p>Distributed</p>

Git : un système de contrôle de version /Code Source Distribué Gratuit

GitHub/GitLab : le serveur distant du git

Espaces de travail : répertoire dans votre ordinateur

(Zone 1) **Dépôt local** : le répertoire de travail + (Zone 2) le répertoire git où l'historique est enregistré (.git hidden directory)

Commit : "capture d'écran " de l'état actuelle du code source
Commit = modifications + message de commit

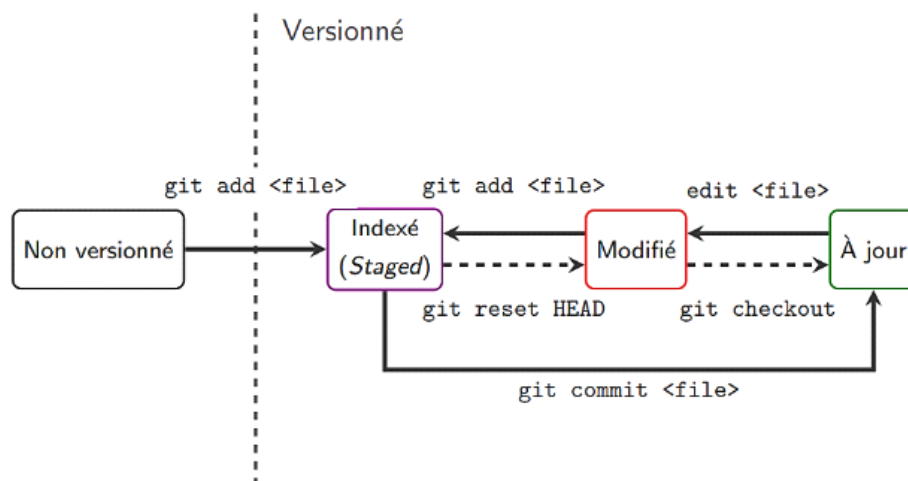
Historique : la chaîne de tous les commit

(Zone 3) Dépôt distant : le repository git qui se trouve dans le serveur distant Github ou GitLab, où le code source est sauvegardé

Commandes du Git :

- **Git log** historique
- **Git push** envoyer au dépôt distant
- **Git Pull** récupérer du dépôt distant
- **Git branch ma_branch** création d'un branche
- **Git branch -m new_branch** renommer la branch
- **Git checkout ma_branch** se positionner dans la branche
- **Git branch -a** lister tous les branches existants dans le dépôt locale
- **Git show-branch** affiche les branches et leur commits
- **Git branch -d ma_branch** supprimer la branche
- **Git add file** ajouter un fichier au dépôt locale
- **Git commit -m "message"** enregistrer un commit
- **Git merge b_source** enregistrer un merge
- **Git Rebase** le réalignement de l'historique des commits

Le cycle de vie d'un fichier



S. BOUHADDOUR

51

Branching :

Branch

- une branche est un pointer sur un commit
- HEAD pointe sur la branche actuelle
- Chaque commit pointe vers le commit précédent

Merge

- Un merge est un commit qui a pour parent les deux branches
- Fusion de deux branches et crée un commit de fusion
- La branche courante avance à ce commit(merge)
- La branche source reste comme il est
- **Git checkout b_destination** se positionner dans la branche qu'on veut avancer
- **Git merge b_source** enregistrer un merge

Rebase

- Historique plus simple
- Autre manière de fusion

Rebase interactif

- Permet de personnaliser les commit

Reword : éditer le message du commit

Squash : fondre le commit de rebase dans le commit précédent

Drop : supprimer un commit

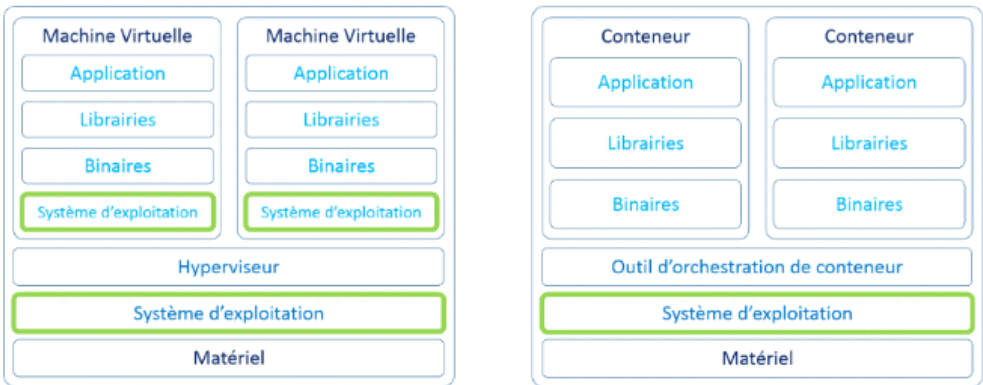
Docker

dimanche 23 juin 2024 13:51

Virtualisation	Cloud Computing
Chaque VM a ses propres ressources	Hébergement en ligne des ressources
Créer plusieurs environnement à partir d'un système physique	Regrouper et automatiser des ressources virtuelles
Fournir pour des utilisateurs spécifique pour une tache spécifique	Fournir pour un groupe d'utilisateurs pour divers tâches
Configurer à partir d'une image	Configurer à partir d'un modèle
Année	Heure ou mois

VM vs Conteneurs

MACHINES VIRTUELLES ≠ CONTENEURS



VM	Conteneur
Chaque vm à son propre SE	Les applications partage un seule SE
Occupe beaucoup d'espace	Relativement plus légers car ils ne contiennent que les bibliothèques
Mise à jour du système doit etre fait individuellement vm par vm	Mise à jour du SE du Hôte seulement (maintenance plus facile)
Hyperviseur	Outil d'orchestration
Passer par le SE principale	Accés direct au matériel
Risque de perte de VM durant la copie ou le déplacement	Faciliter de copie ou déplacement

Docker : outil de conteneurisation

Composants du Docker :

- Daemon Docker
- API de type Rest
- Client en CLI

Fichier docker

Le dockerfile est le code source de l'image Docker

Image Docker

Des plans en lecture seule pour créer des conteneurs

Conteneur

Environnement d'exécution de logiciel

Docker Hub

Dépôt public d'images mises à disposition par Docker

Registre d'image

Bibliothèque d'images disponibles
DockerHub

Etapes de conteneurisation en général



- 1- Créer le dockerfile
- 2- Docker build dockerfile => construction de l'image
- 3- Docker run imageID => lancement du conteneur
- 4- Docker push ConteneurID => dépôt dans le Dockerhub
- 5- Docker pull imagename => récupérer depuis le Dockerhub

Création des images

Manière 1 : Interactivement

Exemple :

- **Docker run -it ubuntu** // un terminal ubuntu est ouvert
- **Apt-get install cowsay**
- **Exit**
- **Docker diff conteneurID** // les fichiers/repas changer auront un C ,
ajouter auront un A
- **Docker commit containerID** // nouvelle imageID est affichée
- **Docker tag imageID cowsay** // changer le nom de la nouvelle image

Manière 2 : Dockerfile

On peut automatiser le processus avec un dockerfile

FROM ubuntu

RUN Apt-get install cowsay

Exemple :

- **Docker build -t cowsay .** // docker cherche le docker file present dans le dossier courant execute l'image et construit le conteneur , effectue les modifications les commits , le conteneur est supprimé et une nouvelle image est générée

Commande Docker File

FROM : image de base

LABEL : association des meta-données

RUN : Commande à exécuter

COPY : ajout d'un fichier

WORKDIR : définition du répertoire de travail

ENV : définition des variables d'environnements

EXPOSE : port exposé

CMD : commande à exécuter dans l'invite de commande

DOCKER CMD

Docker pull

Docker push

Docker images

Docker history

Docker build

Docker run

Docker Commit

Docker tag

Docker Diff

Docker ps

DOCKER-COMPOSE

Fichier de configuration YAML pour faire la :

- Gestion de plusieurs conteneurs (services)
- Gestion des Volumes
- Gestion des Ports
- Gestion des network

```
version: "3.9"
services:
  web:
    build: .
    ports:
      - "5000:5000"
    volumes:
      - ./code
      - logvolume01:/var/log
    depends_on:
      - redis
  redis:
    image: redis
volumes:
  logvolume01:
```

Services	Web	Redis
Image de base	Présente dans le dossier courant et va être construite	Redis
Ports	5000	Aucun port
Volume	2 volumes /code /var/log	Aucun
Ordre (depends_on)	Après Redis	Start first

```

services:
  db:
    image: postgres:latest
    restart: always
    environment:
      - POSTGRES_USER=postgres
      - POSTGRES_PASSWORD=postgres
    ports:
      - 5432:5432
    volumes:
      - db:/var/lib/postgresql/data
    networks:
      - mynet

  web-app:
    image: web-app:latest
    depends_on:
      - db
    networks:
      - mynet
    ports:
      - 8080:8080
    environment:
      DB_HOST: db
      DB_PORT: 5432
      DB_USER: postgres
      DB_PASSWORD: postgres
      DB_NAME: postgres

networks:
  mynet:
    driver: bridge

volumes:
  db:
    driver: local

```

Services	Db	Web-app
Image de base	Postgres:latest	Web-app:latest
Variable d'env	User password	Host Port User Password Name
Ordre	Start first	après db
Ports	5432:5423	8080:8080
Volumes	Db /var/lib ..	Aucun
Network (même network)	Mynet	Mynet

Docker Compose CMD

Docker-compose up -d

Docker-compose ps // voir les conteneur en cours

Docker-compose scale db = 3 // demarrage de 3 conteneur db

Kubernetes

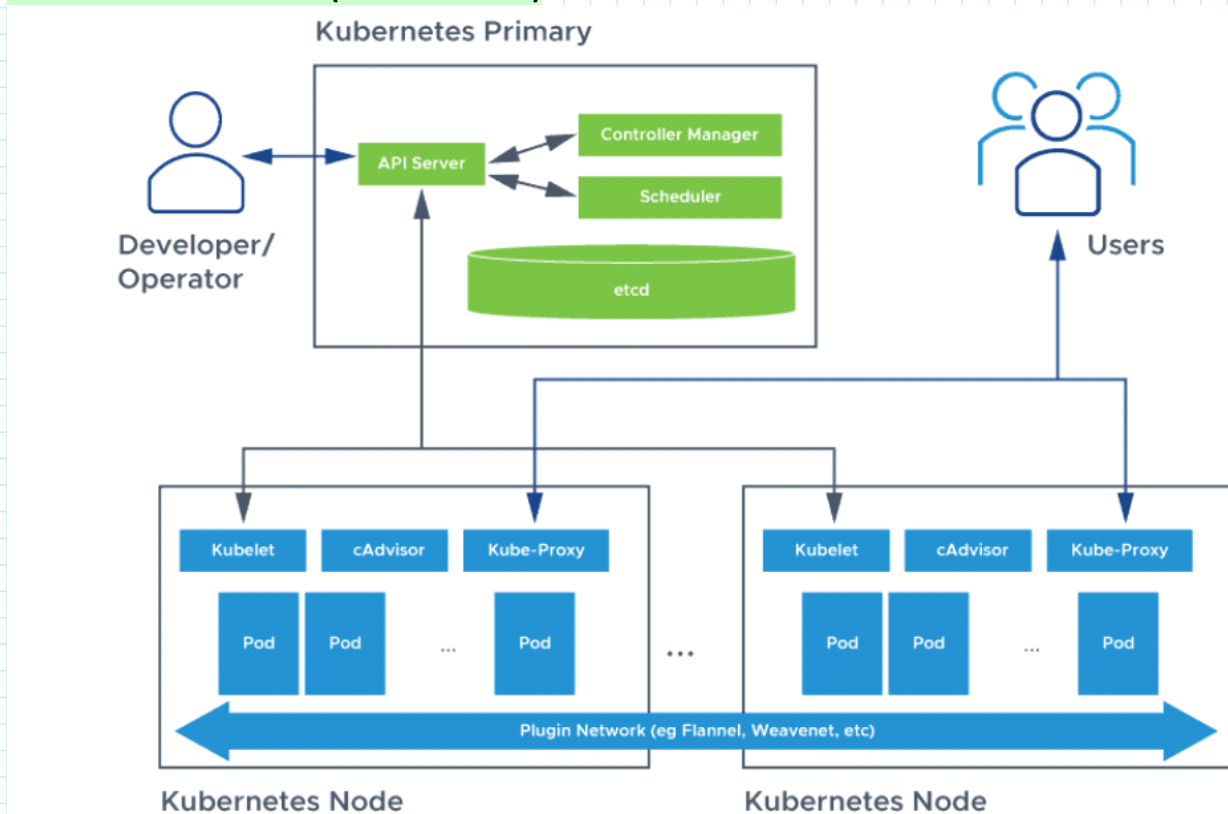
dimanche 23 juin 2024 13:51

But des Kubernetes :

- Gérer les conteneurs dispersés dans un grand nombre de machines
- automatiser le déploiement, la gestion de demande de puissance et la gestion des applications containerisées
- Surveillance

Docker	Kubernetes
une technologie d'exécution de conteneurs qui vous permet de créer, tester et déployer des applications plus rapidement qu'avec des méthodes traditionnelles.	est un outil d'orchestration de conteneurs qui vous permet de mettre à l'échelle vos systèmes de conteneurs afin que vous puissiez gérer, coordonner et planifier les conteneurs à grande échelle.

Architecture Kubernetes (maitre esclave)



- **Master node** : Kubernetes Primary : **administrer** le **cluster** et les Work Node
 - o **Etcd** : stocke l'état du cluster
 - o **API server** : communication avec les composants interne et externe
 - o **Controller Manager**: fait la mise à jour des états à chaque fois un nouveau serveur est crée
 - o **Scheduler** : détermine le Worker node le moins charger
- **Worker Node**: Kubernetes Node : une **VM** qui détient tous les

nécessaires pour les **Pod**

- **Pod = Conteneur**
- **Cadvisor** : envoie les états des Worker Node au Scheduler
- **Kubelet** : responsable de l'état d'exécution
- **Kube-proxy** : le routage du trafic vers le conteneur approprié