Architecture des développements cloud natif

Wiki complet sur l'architecture des développements cloud natif : principes du cloud, applications cloud native, DevOps/SRE, containers, microservices, Kubernetes, applications stateless/stateful, sécurité et CNCF

Systèmes 90 min de lecture Niveau Avancé

Document généré le 27/06/2026 à 21h20 · nouvy.fr/wiki/architecture-cloud-native

Sommaire

45 section(s) · 90 min de lecture

Wiki complet sur l'architecture des développements cloud natif pour concevoir et déployer des applications modernes dans le cloud.

Introduction

Ce wiki couvre les concepts essentiels de l'architecture cloud native, une approche moderne de développement et de déploiement d'applications qui tire parti des avantages du cloud computing.

Objectifs d'apprentissage

- Comprendre les principes généraux du cloud computing
- Concevoir des applications cloud native
- Maîtriser les concepts DevOps et SRE
- Utiliser les containers, microservices et orchestrateurs
- Déployer et gérer des applications sur Kubernetes
- Gérer les applications stateless et stateful
- Sécuriser les applications cloud native
- Connaître l'écosystème CNCF

Prérequis

- Bases en développement logiciel
- Connaissances en systèmes d'exploitation
- Notions de base en réseaux

Partie 1 : Principes généraux du Cloud

1.1 Définition du Cloud Computing

Le **cloud computing** est un modèle de fourniture de services informatiques via Internet, permettant l'accès à des ressources partagées (serveurs, stockage, applications) à la demande.

Caractéristiques essentielles

- **À la demande** : Ressources disponibles instantanément
- **Accès réseau large** : Accessible depuis n'importe où
- **Mise en commun des ressources** : Partage des ressources entre plusieurs clients
- **Élasticité rapide** : Capacité à monter/descendre rapidement
- **Service mesuré** : Facturation à l'usage

1.2 Modèles de service (Service Models)

IaaS (Infrastructure as a Service)

Fournit des ressources informatiques de base (serveurs virtuels, stockage, réseaux).

Exemples :

- Amazon EC2
- Microsoft Azure Virtual Machines
- Google Compute Engine

Avantages :

- Contrôle total sur l'infrastructure
- Flexibilité maximale
- Responsabilité de la gestion du système d'exploitation

PaaS (Platform as a Service)

Fournit une plateforme de développement et de déploiement.

Exemples :

- Heroku
- Google App Engine
- Microsoft Azure App Service

Avantages :

- Pas de gestion de l'infrastructure
- Focus sur le développement
- Scalabilité automatique

SaaS (Software as a Service)

Fournit des applications complètes accessibles via le navigateur.

Exemples :

- Gmail
- Salesforce
- Microsoft 365

Avantages :

- Pas d'installation
- Maintenance gérée par le fournisseur
- Accès depuis n'importe où

1.3 Modèles de déploiement (Deployment Models)

Cloud Public

Infrastructure partagée entre plusieurs organisations.

Avantages :

- Coûts réduits
- Scalabilité élevée
- Maintenance gérée

Inconvénients :

- Moins de contrôle
- Préoccupations de sécurité

Cloud Privé

Infrastructure dédiée à une seule organisation.

Avantages :

- Contrôle total
- Sécurité renforcée
- Conformité facilitée

Inconvénients :

- Coûts plus élevés
- Maintenance interne

Cloud Hybride

Combinaison de cloud public et privé.

Avantages :

- Flexibilité
- Optimisation des coûts
- Meilleur des deux mondes

1.4 Avantages du Cloud Computing

- **Réduction des coûts** : Pas d'investissement initial en matériel
- **Scalabilité** : Adaptation automatique à la demande
- **Disponibilité** : Accès 24/7 depuis n'importe où
- **Performance** : Infrastructure optimisée
- **Sécurité** : Mesures de sécurité avancées
- **Innovation** : Accès aux dernières technologies

Partie 2 : Applications Cloud Native

2.1 Qu'est-ce qu'une application Cloud Native ?

Une **application cloud native** est conçue spécifiquement pour fonctionner dans des environnements cloud, tirant parti des avantages du cloud computing.

Caractéristiques principales

- **Microservices** : Architecture modulaire
- **Containers** : Empaquetage léger et portable
- **Orchestration** : Gestion automatique des conteneurs
- **CI/CD** : Intégration et déploiement continus
- **DevOps** : Collaboration développement/opérations
- **Résilience** : Tolérance aux pannes
- **Scalabilité** : Adaptation à la charge

2.2 Principes de conception

12-Factor App

Méthodologie pour construire des applications SaaS modernes :

1. **Codebase** : Un seul codebase, plusieurs déploiements
2. **Dependencies** : Déclarer explicitement les dépendances
3. **Config** : Stocker la configuration dans l'environnement
4. **Backing services** : Traiter les services externes comme des ressources
5. **Build, release, run** : Séparer strictement les étapes
6. **Processes** : Exécuter l'application comme des processus stateless
7. **Port binding** : Exporter les services via port binding
8. **Concurrency** : Scalabilité via le modèle de processus
9. **Disposability** : Maximiser la robustesse avec des démarrages/arrêts rapides
10. **Dev/prod parity** : Garder les environnements similaires
11. **Logs** : Traiter les logs comme des flux d'événements
12. **Admin processes** : Exécuter les tâches d'administration comme des processus one-off

2.3 Architecture microservices

Définition

Les **microservices** sont une approche architecturale où une application est décomposée en petits services indépendants et déployables.

Avantages

- **Indépendance** : Développement et déploiement indépendants
- **Scalabilité** : Scalabilité granulaire par service
- **Technologie** : Choix de la technologie par service
- **Résilience** : Isolation des pannes
- **Équipes** : Organisation par équipes autonomes

Défis

- **Complexité** : Gestion de la distribution
- **Communication** : Latence réseau
- **Données** : Gestion distribuée des données
- **Tests** : Tests plus complexes

- **Déploiement** : Orchestration nécessaire

2.4 Comparaison Monolithique vs Microservices

Aspect	Monolithique	Microservices
Déploiement	Un seul déploiement	Déploiements multiples
Scalabilité	Scalabilité globale	Scalabilité par service
Technologie	Technologie unique	Technologies variées
Complexité	Simple au début	Complexe dès le départ
Tests	Tests simples	Tests distribués
Déploiement	Facile	Nécessite orchestration

Partie 3 : DevOps et SRE

3.1 DevOps

DevOps est une culture et un ensemble de pratiques qui unifient le développement logiciel (Dev) et les opérations informatiques (Ops).

Principes DevOps

- **Collaboration** : Communication entre Dev et Ops
- **Automatisation** : Automatisation des processus
- **Intégration continue** : Intégration fréquente du code
- **Déploiement continu** : Déploiement automatisé
- **Monitoring** : Surveillance continue
- **Feedback** : Boucle de retour rapide

Outils DevOps

- **CI/CD** : Jenkins, GitLab CI, GitHub Actions
- **Configuration** : Ansible, Puppet, Chef
- **Containers** : Docker, Podman
- **Orchestration** : Kubernetes, Docker Swarm
- **Monitoring** : Prometheus, Grafana
- **Logging** : ELK Stack, Loki

3.2 SRE (Site Reliability Engineering)

SRE est une discipline qui applique les aspects de l'ingénierie logicielle aux problèmes d'opérations.

Principes SRE

- **Service Level Objectives (SLO)** : Objectifs de niveau de service

- **Service Level Indicators (SLI)** : Indicateurs de niveau de service
- **Error Budgets** : Budget d'erreur acceptable
- **Toil Reduction** : Réduction du travail répétitif
- **Automation** : Automatisation maximale
- **Monitoring** : Surveillance proactive

Métriques SRE

- **Availability** : Disponibilité du service
- **Latency** : Temps de réponse
- **Throughput** : Débit de traitement
- **Error Rate** : Taux d'erreur

3.3 CI/CD (Continuous Integration / Continuous Deployment)

Intégration Continue (CI)

Pratique consistant à intégrer fréquemment le code dans un dépôt partagé.

Avantages :

- Détection précoce des bugs
- Réduction des conflits
- Qualité de code améliorée

Pipeline CI typique :

1. Commit du code
2. Build automatique
3. Tests automatiques
4. Analyse de code
5. Génération d'artefacts

Déploiement Continu (CD)

Pratique consistant à déployer automatiquement le code en production.

Avantages :

- Déploiements fréquents
- Réduction des risques
- Feedback rapide

Pipeline CD typique :

1. Tests d'acceptation
2. Déploiement en staging
3. Tests de régression
4. Déploiement en production
5. Monitoring post-déploiement

Partie 4 : Containers, Microservices et Orchestrateurs

4.1 Containers

Définition

Un **container** est une unité d'exécution légère qui empaquette une application avec toutes ses dépendances.

Avantages des containers

- **Portabilité** : Fonctionne partout
- **Isolation** : Isolation des processus
- **Efficacité** : Utilisation optimale des ressources
- **Rapidité** : Démarrage rapide
- **Scalabilité** : Scalabilité facile

Docker

Docker est la plateforme de conteneurisation la plus populaire.

Concepts clés :

- **Image** : Modèle en lecture seule
- **Container** : Instance exécutable d'une image
- **Dockerfile** : Fichier de définition d'une image
- **Docker Compose** : Orchestration multi-conteneurs

4.2 Microservices

Architecture microservices

Chaque microservice est :

- **Autonome** : Peut être développé et déployé indépendamment
- **Spécialisé** : Responsable d'une fonctionnalité métier
- **Découplé** : Communication via APIs
- **Résilient** : Tolérant aux pannes

Communication entre microservices

- **Synchronous** : REST, gRPC
- **Asynchronous** : Message queues (RabbitMQ, Kafka)
- **Service Mesh** : Istio, Linkerd

4.3 Orchestrateurs

Définition

Un **orchestrateur** gère automatiquement le cycle de vie des containers.

Fonctionnalités

- **Scheduling** : Placement des containers

- **Scaling** : Mise à l'échelle automatique
- **Health checks** : Vérification de santé
- **Load balancing** : Répartition de charge
- **Self-healing** : Récupération automatique

Orchestrateurs populaires

- **Kubernetes** : Le plus populaire
- **Docker Swarm** : Intégré à Docker
- **Apache Mesos** : Pour grandes infrastructures
- **Nomad** : Simple et léger

Partie 5 : Les bases de Kubernetes

5.1 Introduction à Kubernetes

Kubernetes (K8s) est un système open source d'orchestration de containers.

Pourquoi Kubernetes ?

- **Scalabilité** : Gestion de milliers de containers
- **Haute disponibilité** : Tolérance aux pannes
- **Portabilité** : Fonctionne sur tous les clouds
- **Écosystème** : Large communauté

5.2 Architecture Kubernetes

Composants du cluster

- **Master (Control Plane)** :
  - API Server
  - etcd
  - Scheduler
  - Controller Manager

- **Nodes (Workers)** :
  - Kubelet
  - Kube-proxy
  - Container Runtime

5.3 Concepts fondamentaux

Pod

Plus petite unité déployable dans Kubernetes. Un Pod peut contenir un ou plusieurs containers.

Deployment

Gère les Pods et leur réplication.

Service

Expose les Pods pour permettre la communication.

Types de Services :

- **ClusterIP** : Accès interne au cluster
- **NodePort** : Accès via port du node
- **LoadBalancer** : IP externe via cloud provider
- **ExternalName** : Alias vers service externe

Namespace

Isolation logique des ressources dans un cluster.

Namespaces par défaut :

- `default` : Ressources par défaut
- `kube-system` : Composants système
- `kube-public` : Ressources publiques
- `kube-node-lease` : Heartbeats des nodes

5.4 Commandes Kubernetes essentielles

Commandes principales :

- `kubectl get pods` : Lister les pods
- `kubectl apply -f deployment.yaml` : Créer une ressource
- `kubectl logs <pod-name>` : Voir les logs
- `kubectl exec -it <pod-name> -- /bin/sh` : Exécuter une commande dans un pod
- `kubectl scale deployment <name> --replicas=5` : Scalabilité
- `kubectl delete -f deployment.yaml` : Supprimer une ressource

5.5 ConfigMaps et Secrets

ConfigMap

Stocke des données de configuration non sensibles (URLs de base de données, niveaux de log, etc.).

Secret

Stocke des données sensibles (mots de passe, clés API, certificats). Les secrets sont encodés en base64 mais ne sont pas chiffrés par défaut dans Kubernetes.

Partie 6 : Applications Stateless et Stateful

6.1 Applications Stateless

Définition

Une application **stateless** ne conserve pas d'état entre les requêtes.

Caractéristiques

- **Sans état** : Chaque requête est indépendante
- **Scalable** : Facile à scaler horizontalement
- **Résiliente** : Pas de perte d'état en cas de panne
- **Simple** : Plus facile à gérer

Exemples

- APIs REST
- Serveurs web statiques
- Services de calcul

Déploiement Kubernetes

Utilisation de **Deployments** pour les applications stateless.

6.2 Applications Stateful

Définition

Une application **stateful** conserve un état entre les requêtes.

Caractéristiques

- **Avec état** : Maintient un état persistant
- **Stockage** : Nécessite un stockage persistant
- **Complexité** : Plus complexe à gérer
- **Scalabilité** : Scalabilité plus délicate

Exemples

- Bases de données
- Systèmes de cache
- Applications avec sessions

Déploiement Kubernetes

Utilisation de **StatefulSets** pour les applications stateful. Les StatefulSets garantissent un ordre de déploiement et des identifiants stables pour chaque pod.

6.3 Volumes Kubernetes

Types de volumes

- **emptyDir** : Volume temporaire
- **hostPath** : Accès au système de fichiers du node

- **PersistentVolume (PV)** : Volume persistant
- **PersistentVolumeClaim (PVC)** : Demande de volume persistant. Les PVC permettent aux pods de réclamer du stockage persistant.

Partie 7 : Sécurité des applications Cloud Native

7.1 Principes de sécurité

Defense in Depth

Approche multicouche de la sécurité.

Couches de sécurité :

- Réseau
- Application
- Données
- Identité

Least Privilege

Principe du moindre privilège : accorder uniquement les permissions nécessaires.

7.2 Sécurité des containers

Bonnes pratiques

- **Images de base** : Utiliser des images officielles et à jour
- **Non-root** : Exécuter en tant qu'utilisateur non-root
- **Secrets** : Ne pas stocker de secrets dans les images
- **Scanning** : Scanner les images pour vulnérabilités
- **Minimal** : Images minimales (Alpine Linux)

7.3 Sécurité Kubernetes

RBAC (Role-Based Access Control)

Contrôle d'accès basé sur les rôles. Permet de définir des rôles et des permissions pour les utilisateurs et les services.

Network Policies

Isolation réseau entre les pods. Permet de contrôler le trafic réseau entrant et sortant des pods.

Pod Security Standards

Standards de sécurité pour les pods :

- **Privileged** : Aucune restriction
- **Baseline** : Restrictions minimales

- **Restricted** : Restrictions maximales

7.4 Secrets Management

Gestion des secrets

- **Kubernetes Secrets** : Stockage natif (base64, non chiffré)
- **External Secrets** : Intégration avec Vault, AWS Secrets Manager
- **Sealed Secrets** : Secrets chiffrés dans Git

Partie 8 : CNCF et son écosystème

8.1 Cloud Native Computing Foundation (CNCF)

La **CNCF** est une fondation qui soutient et développe l'écosystème cloud native.

Mission

- Promouvoir les technologies cloud native
- Faciliter l'adoption
- Standardiser les pratiques

8.2 Projets CNCF

Niveau Graduated (Production Ready)

- **Kubernetes** : Orchestration de containers
- **Prometheus** : Monitoring et alerting
- **Envoy** : Proxy de service mesh
- **CoreDNS** : DNS pour Kubernetes
- **containerd** : Runtime de containers
- **Fluentd** : Collecte de logs
- **gRPC** : Framework RPC
- **Helm** : Gestionnaire de packages Kubernetes

Niveau Incubating

- **Istio** : Service mesh
- **Linkerd** : Service mesh léger
- **Jaeger** : Traçage distribué
- **Vitess** : Base de données scalable
- **OpenTelemetry** : Observabilité

Niveau Sandbox

- **Argo** : Workflows GitOps
- **KubeEdge** : Edge computing
- **KubeVirt** : Virtualisation sur Kubernetes

8.3 Landscape CNCF

Le **CNCF Landscape** catégorise les projets et outils cloud native :

- **Orchestration & Management**
- **Runtime**
- **Provisioning**
- **Observability & Analysis**
- **Service Mesh**
- **API Gateway**
- **Service Proxy**
- **Cloud Native Storage**
- **Database**
- **Streaming & Messaging**
- **Container Registry**
- **Security & Compliance**
- **Key Management**
- **Container Runtime**
- **Software Distribution**

Conclusion

Ce wiki a couvert les essentiels de l'architecture cloud native :

1. **Principes du Cloud** : IaaS, PaaS, SaaS, modèles de déploiement
2. **Applications Cloud Native** : 12-Factor App, microservices
3. **DevOps et SRE** : Culture, pratiques, CI/CD
4. **Containers et Orchestration** : Docker, Kubernetes
5. **Kubernetes** : Pods, Deployments, Services, ConfigMaps, Secrets
6. **Applications Stateless/Stateful** : Déploiements adaptés
7. **Sécurité** : Bonnes pratiques, RBAC, Network Policies
8. **CNCF** : Écosystème et projets

Ces connaissances sont essentielles pour développer et déployer des applications modernes dans le cloud, en tirant parti de la scalabilité, de la résilience et de l'efficacité offertes par les technologies cloud native.

Ressources complémentaires

Documentation officielle

- Kubernetes Documentation
- CNCF Landscape
- 12-Factor App

- Docker Documentation

Plateformes d'apprentissage

- Kubernetes Playground
- CNCF Training
- Kubernetes Tutorials

Livres recommandés

- "Kubernetes Up & Running" - Kelsey Hightower
- "The DevOps Handbook" - Gene Kim
- "Site Reliability Engineering" - Google
- "Cloud Native Patterns" - Cornelia Davis