

Antony Kervazo-Canut

ArgoCD for Teenagers

KUBERNETES PAR LE GITOPS



SOMMAIRE

Introduction	3
Installation D'ArgoCD	4
Configuration d'ArgoCD	5
Configuration d'un repository (SSH)	6
Création d'une application (GUI)	7
Création d'une application (CLI)	10
Kustomize	11
Création d'une application (Helm)	13
Helm et multiples sources	14
Synchronisation	15
SyncOptions	16
Resource Hooks	18
Wave Synchronization	21
Multi-cluster Kubernetes	23
Communauté	24

Introduction



ArgoCD est une plateforme de déploiement continu spécialement conçue pour Kubernetes, qui adopte l'approche GitOps. GitOps est une méthode de gestion des infrastructures et des applications qui utilise Git comme source unique de vérité. Avec GitOps, les changements dans l'infrastructure ou les applications sont effectués par des modifications dans un dépôt Git, assurant ainsi la traçabilité, la révision et la reproductibilité des déploiements.

Les avantages de GitOps avec ArgoCD incluent :

- Automatisation et cohérence : Les déploiements se font automatiquement lorsque le dépôt Git est mis à jour, assurant une cohérence entre le code source et l'état déployé.
- Traçabilité et audit : Chaque modification est enregistrée dans Git, offrant une traçabilité complète.
- Facilité de reprise après incident : En cas de problème, il est facile de revenir à un état précédent en utilisant les fonctionnalités de réversion de Git.
- Sécurité renforcée : Les approbations et les revues de code dans Git contribuent à une meilleure sécurité des déploiements.

L'architecture d'ArgoCD se compose de plusieurs composants clés :

- API Server : Le serveur API est le point central de communication.
- Repository Server : Gère la communication avec les dépôts Git.
- Controller : Surveille l'état des applications et gère la synchronisation.

Installation D'ArgoCD



Vous pouvez installer ArgoCD sur votre cluster Kubernetes en appliquant le manifeste YAML fourni par ArgoCD.

ArgoCD fonctionne également en CLI, mais pour accéder à l'interface web d'ArgoCD, exposez le service argocd-server en dehors du cluster.

Enfin, il faudra installer ArgoCD en CLI pour le gérer de cette manière.

```
● ● ●

# Création du namespace
kubectl create namespace argocd

# Installation d'ArgoCD sur le cluster
kubectl apply -n argocd -f
https://raw.githubusercontent.com/argoproj/argo-
cd/stable/manifests/install.yaml

# Création d'un service pour accéder à l'interface web
kubectl port-forward svc/argocd-server -n argocd 8080:443

# Linux : Télécharger la CLI ArgoCD (remplacer VERSION par la
version désirée)
curl -sSL -o argocd https://github.com/argoproj/argo-
cd/releases/download/vVERSION/argocd-linux-amd64
chmod +x argocd
sudo mv argocd /usr/local/bin/

# Windows : Installer la CLI ArgoCD avec Chocolatey
choco install argocd

# Installer la CLI ArgoCD avec Homebrew
brew install argocd
```

Configuration d'ArgoCD



Une fois la CLI installée, vous pouvez vous connecter à votre serveur ArgoCD. La première étape consiste à récupérer l'adresse IP ou le nom du service argocd-server.

```
● ● ●

# Obtenir l'adresse IP d'ArgoCD (si vous utilisez un LoadBalancer)
kubectl get svc -n argocd argocd-server -o jsonpath='{.status.loadBalancer.ingress[0].ip}'

# Ou utiliser port-forward pour accéder localement
kubectl port-forward svc/argocd-server -n argocd 8080:443

# Obtenir le mot de passe initial de l'admin
kubectl get secret argocd-initial-admin-secret -n argocd -o jsonpath="{.data.password}" | base64 -d; echo

# Connexion à ArgoCD (remplacer ARGOCD_SERVER par l'adresse IP ou le nom de domaine)
argocd login ARGOCD_SERVER --username admin --password <initial_admin_password>
```

Il devient alors possible de se connecter à l'interface web sur le port 8080. Le mot de passe du compte "admin" est celui retourné par la CLI.

Configuration d'un repository (SSH)



Pour configurer un dépôt Git via SSH dans ArgoCD, vous devez suivre plusieurs étapes pour assurer une communication sécurisée entre ArgoCD et votre dépôt Git. Cela implique la création d'une paire de clés SSH, l'ajout de la clé publique à votre dépôt Git et la configuration d'ArgoCD pour utiliser la clé privée lors de l'accès au dépôt.

Le processus de création de paires de clefs SSH ne sera pas détaillé ici.

D'un côté, placez votre clef ssh publique dans une forge de votre choix (azure devops, github, gitlab, etc) et de l'autre allez sur ArgoCD/Settings.Repositories et le bouton "Connect Repo".

CONNECT
SAVE AS CREDENTIALS TEMPLATE
CANCEL

Choose your connection method:

VIA SSH ▾

CONNECT REPO USING SSH

Name (mandatory for Helm)
project1

Project
default

Repository URL
git@ssh.dev.azure.com:v3/ODYCD/ForBabies/ArgoCD_Project1

SSH private key data
mykey

Création d'une application (GUI)



Pour créer une application, vous disposez de deux méthodes : la GUI ou la CLI. Mais il faut d'abord définir ce qu'est une application pour ArgoCD.

Il s'agit d'un repository composé de ressources pour Kubernetes. Donc pour notre exemple, nous allons remplir le repository (vide) que avons connecté dans la page précédente avec un deployment nginx.

```
# fichier: nginx-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  namespace: nginx-namespace
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:latest
          ports:
            - containerPort: 80
```

Création d'une application (GUI)



ArgoCD propose des fichiers yaml propre à son fonctionnement si l'utilisation des champs de texte vous rebute. Ils peuvent d'ailleurs être facilement être utilisés pour être insérés via une CLI et donc pourquoi pas, intégrés à une pipeline de Delivery.

```
● ● ●

apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: myproject
spec:
  destination:
    name: ''
    namespace: nginx-namespace
    server: 'https://kubernetes.default.svc'
  source:
    path: .
    repoURL:
      'git@ssh.dev.azure.com:v3/ODYCD/ForBabies/ArgoCD_Project1'
      targetRevision: HEAD
  sources: []
  project: default
  syncPolicy:
    automated:
      prune: false
      selfHeal: true
  syncOptions:
    - CreateNamespace=true
```

Création d'une application (GUI)



GENERAL

Application Name: myproject

Project Name: default

SYNC POLICY: Automatic

PRUNE RESOURCES ⓘ SELF HEAL ⓘ

SET DELETION FINALIZER ⓘ

SYNC OPTIONS

SKIP SCHEMA VALIDATION AUTO-CREATE NAMESPACE
 PRUNE LAST APPLY OUT OF SYNC ONLY
 RESPECT IGNORE DIFFERENCES SERVER-SIDE APPLY

PRUNE PROPAGATION POLICY: foreground

REPLACE ⚠️ RETRY

SOURCE

Repository URL: git@ssh.dev.azure.com:v3/ODYCD/ForBabies/ArgoCD_Project1

Revision: HEAD

Path: -

Branches: ▾

Une fois l'application créée avec notre repository comme référence, ArgoCD va déployer notre ressource et l'observer sur le cluster Kubernetes.



Création d'une application (CLI)



En CLI, vous disposez de plusieurs moyens pour déployer l'application. L'utilisation d'un fichier YAML, plus pratique à maintenir ou passer directement dans la commande les paramètres de création.

```
● ● ●

# Créer une application directement en utilisant le YAML.
argocd app create --file myproject-application.yaml

# Ou créer une application en utilisant des paramètres
argocd app create myproject \
  --dest-namespace nginx-namespace \
  --dest-server https://kubernetes.default.svc \
  --repo
git@ssh.dev.azure.com:v3/ODYCD/ForBabies/ArgoCD_Project1 \
  --path . \
  --project default \
  --sync-policy automated \
  --sync-option CreateNamespace=true \
  --auto-prune false \
  --self-heal true
```

Kustomize



ArgoCD utilise les fichiers de configuration Kustomize (`kustomization.yaml`) pour générer les ressources Kubernetes finales à partir des templates de base et des overlays de personnalisation. Les overlays permettent de spécifier des modifications pour un environnement spécifique sans avoir besoin de dupliquer l'ensemble des fichiers de configuration.

Les applications Git méritent alors une restructuration et organisation des fichiers.

```
mon-application/
└── base/
    ├── deployment.yaml
    ├── kustomization.yaml
    └── service.yaml
└── overlays/
    ├── production/
    │   └── kustomization.yaml
    │   └── patch-production.yaml
    └── development/
        └── kustomization.yaml
        └── patch-development.yaml
```

Kustomize



Lors de la création de l'objet Application dans ArgoCD pour votre application, spécifiez le chemin vers le dossier de l'overlay correspondant à l'environnement que vous souhaitez déployer.

```
● ● ●

apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: mon-application-production
spec:
  project: default
  source:
    repoURL:
      git@ssh.dev.azure.com:v3/ODYCD/ForBabies/ArgoCD_Project1'
      targetRevision: HEAD
      path: mon-application/overlays/production
  destination:
    server: 'https://kubernetes.default.svc'
    namespace: production
```

Création d'une application (Helm)



Il est possible de déployer des applications en utilisant des Charts Helm avec ArgoCD, combinant la gestion de paquets Helm pour Kubernetes et l'automatisation GitOps pour une mise à jour et une configuration simplifiées des applications.

Cela peut être plus pratique pour déployer une application dans une plusieurs équipes ou plusieurs environnements en changeant juste de la configuration.

```
● ● ●

apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: sonarqube
spec:
  destination:
    name: ''
    namespace: default
    server: 'https://kubernetes.default.svc'
  source:
    path: ''
    repoURL: 'https://SonarSource.github.io/helm-chart-sonarqube'
  targetRevision: 10.4.0+2288
  chart: sonarqube
  sources: []
  project: default
  syncPolicy:
    automated:
      prune: false
      selfHeal: true
  syncOptions:
    - CreateNamespace=true
```

Helm et multiples sources



Avec ArgoCD il est d'ailleurs tout à fait possible de ne stocker sur Git que la configuration et de ne déployer que des releases Helm.
Permettant de ne faire évoluer que la configuration sur une même version.

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: sonarqube
spec:
  project: default
  source:
    repoURL: 'https://SonarSource.github.io/helm-chart-sonarqube'
    targetRevision: 10.4.0+2288
    chart: sonarqube
    helm:
      valueFiles:
        - $values/values-sonar-dev.yaml
  sources:
    - repoURL:
      'git@ssh.dev.azure.com:v3/0DYCD/ForBabies/ArgoCD_ProjectSonar'
        targetRevision: main
        ref: values
  destination:
    server: 'https://kubernetes.default.svc'
    namespace: default
  syncPolicy:
    automated:
      prune: false
      selfHeal: true
  syncOptions:
    - CreateNamespace=true
```

Synchronisation



Automatique vs Manuelle : Vous pouvez configurer une application pour qu'elle se synchronise automatiquement à chaque fois que le dépôt Git change (synchronisation automatique), ou vous pouvez choisir de déclencher la synchronisation manuellement (synchronisation manuelle).

- **Automatique :** Avec l'option `automated`, la synchronisation se fait automatiquement. Vous pouvez également spécifier si les ressources absentes dans le dépôt Git doivent être supprimées du cluster (`prune: true`) et si ArgoCD doit tenter de corriger automatiquement les déviations sans intervention manuelle (`selfHeal: true`).
- **Manuelle :** La synchronisation manuelle nécessite que l'utilisateur déclenche explicitement une synchronisation via l'interface utilisateur d'ArgoCD ou la CLI.
- **Prune :** Lorsqu'elle est activée (`prune: true` dans une politique de synchronisation automatique), cette option indique à ArgoCD de supprimer les ressources Kubernetes qui ne sont plus présentes dans le dépôt Git mais qui existent encore dans le cluster.
- **Self-Heal :** Si activé (`selfHeal: true`), ArgoCD surveillera les modifications apportées aux ressources Kubernetes dans le cluster qui divergent de la configuration Git et les réappliquera automatiquement. Cela garantit que l'état du cluster reste en synchronisation avec le dépôt Git.

SyncOptions



Les syncOptions permettent de personnaliser davantage la synchronisation :

- **CreateNamespace** : Si défini (`CreateNamespace=true`), ArgoCD créera automatiquement le namespace spécifié pour l'application s'il n'existe pas déjà dans le cluster. Cela est utile pour les déploiements qui nécessitent la création de nouveaux namespaces à la volée.
- **Validate** : Contrôle la validation des ressources avant leur application. Vous pouvez désactiver la validation avec `Validate=false` si nécessaire.
- **PrunePropagationPolicy** : Définit la politique de propagation pour les opérations de suppression, par exemple, `PrunePropagationPolicy=Foreground` garantira que toutes les ressources dépendantes sont supprimées avant la ressource elle-même.
- **PruneLast** : Permet de spécifier des ressources qui doivent être supprimées en dernier lors de l'opération de prune, utile pour gérer les dépendances entre ressources.
- **Retry** : Configure les tentatives de réessai en cas d'échec de synchronisation, par exemple:
`Retry=limit:5,backoff:exponential,maxDuration:15m.`

SyncOptions



Le YAML de l'application avec des SyncOptions activés ressemble alors à ceci :

```
● ● ●

apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: myproject
spec:
  project: default
  source:
    repoURL:
      'git@ssh.dev.azure.com:v3/ODYCD/ForBabies/ArgoCD_Project1'
      path: .
      targetRevision: HEAD
  destination:
    server: 'https://kubernetes.default.svc'
    namespace: nginx-namespace
  syncPolicy:
    automated:
      prune: true
      selfHeal: true
  syncOptions:
    - CreateNamespace=true
    - Validate=true
    - PrunePropagationPolicy=Foreground
    - PruneLast=true
    - ApplyOutOfSyncOnly=true
    - SkipDryRunOnMissingResource=true
```

Resource Hooks



Les resource hooks dans ArgoCD sont des outils puissants qui vous permettent d'orchestrer le déploiement de vos applications en exécutant des tâches spécifiques à différents moments du processus de synchronisation. Ces hooks utilisent des ressources Kubernetes standard, comme les Jobs ou les Pods, pour exécuter des scripts ou des commandes avant, pendant, ou après la synchronisation de l'application. Ils sont particulièrement utiles pour gérer les migrations de base de données, les nettoyages, les notifications, et d'autres tâches de gestion ou d'initialisation nécessaires à un déploiement réussi.

ArgoCD supporte plusieurs types de hooks, chacun étant exécuté à un moment spécifique du cycle de vie du déploiement :

- PreSync : Exécuté avant que les modifications ne soient appliquées au cluster.
- Sync : Exécuté après que les hooks PreSync sont terminés mais avant que la synchronisation de l'état désiré soit effectuée.
- PostSync : Exécuté après que toutes les ressources ont été synchronisées avec l'état désiré dans le dépôt Git.
- SyncFail : Exécuté si le processus de synchronisation échoue à n'importe quel moment.

Resource Hooks



Pour utiliser un resource hook, vous devez définir une ressource Kubernetes dans votre dépôt Git qui inclut des annotations spécifiant le type de hook et quand il doit s'exécuter. Voici un exemple de Job Kubernetes configuré comme un hook PreSync :

```
apiVersion: batch/v1
kind: Job
metadata:
  name: example-pre-resources-hook
  annotations:
    argocd.argoproj.io/hook: PreSync
spec:
  template:
    spec:
      containers:
        - name: pre-sync
          image: alpine
          command: ["/bin/sh", "-c"]
          args: ["echo Pre-sync hook; sleep 60"]
      restartPolicy: Never
```

La notion de poids (hook-weight) dans les Resource Hooks peut être utilisée pour contrôler l'ordre d'exécution des hooks lorsqu'il y en a plusieurs à exécuter dans la même phase du cycle de vie du déploiement (par exemple, plusieurs PreSync hooks). Le poids est un nombre entier, et les hooks sont exécutés du poids le plus bas au poids le plus élevé. Les hooks ayant le même poids sont exécutés dans un ordre non déterministe par rapport les uns aux autres.

Resource Hooks



Exemple de 2 resources hooks dont l'ordre d'exécution est géré par le poids :

```
● ● ●

apiVersion: batch/v1
kind: Job
metadata:
  name: pre-sync-database-migration
  annotations:
    argocd.argoproj.io/hook: PreSync
    argocd.argoproj.io/hook-weight: "10"
spec:
  template:
    spec:
      containers:
        - name: migration
          image: myapp/migration:latest
          command: ["/bin/sh", "-c", "perform-migration"]
          restartPolicy: Never
  —
apiVersion: batch/v1
kind: Job
metadata:
  name: pre-sync-seed-database
  annotations:
    argocd.argoproj.io/hook: PreSync
    argocd.argoproj.io/hook-weight: "20"
spec:
  template:
    spec:
      containers:
        - name: seed
          image: myapp/seed:latest
          command: ["/bin/sh", "-c", "seed-database"]
          restartPolicy: Never
```

Wave Synchronization



Les "waves" de synchronisation permettent de regrouper les ressources dans des phases de déploiement séquentielles. Vous pouvez spécifier l'ordre de déploiement en attribuant des valeurs de "wave" aux ressources à l'aide de l'annotation `argocd.argoproj.io-sync-wave`. Les ressources avec une valeur de "wave" inférieure sont déployées avant celles avec une valeur supérieure.

Imaginons que vous déployez une application composée d'une base de données, d'un backend, et d'un frontend. Vous voulez que la base de données soit déployée en premier, suivie du backend, puis du frontend.

```
● ● ●

apiVersion: apps/v1
kind: Deployment
metadata:
  name: database-deployment
  annotations:
    argocd.argoproj.io/sync-wave: "0"
spec:
  # Spécification du déploiement
```

Wave Synchronization



```
● ● ●

apiVersion: apps/v1
kind: Deployment
metadata:
  name: backend-deployment
  annotations:
    argocd.argoproj.io/sync-wave: "1"
spec:
  # Spécification du déploiement
```

```
● ● ●

apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend-deployment
  annotations:
    argocd.argoproj.io/sync-wave: "2"
spec:
  # Spécification du déploiement
```

Vous avez alors un contrôle précis de l'ordre de déploiement, essentiel pour les applications interdépendantes.
Cela réduit les risques de problèmes causés par des dépendances non résolues pendant le déploiement.

Multi-cluster Kubernetes



Le support multi-cluster dans ArgoCD permet de gérer des déploiements dans plusieurs clusters Kubernetes à partir d'une seule instance ArgoCD, offrant une solution puissante pour les organisations qui exploitent des environnements Kubernetes distribués. Cette fonctionnalité facilite la gestion centralisée des applications à travers différents clusters, permettant des opérations cohérentes, une visibilité accrue et une meilleure gouvernance.

Il ne se fait qu'en CLI.

```
● ● ●

# Récupère la liste des différents cluster connus
kubectl config get-contexts

# Ajoute un nouveau cluster géré par ArgoCD
argocd cluster add CONTEXT_NAME --name nom_pour_argocd
```

Nommez clairement vos clusters et applications pour refléter leur environnement et fonction (par exemple, prod, dev, us-east, etc.), facilitant la gestion à grande échelle.

Communauté



La communauté ArgoCD a développé plusieurs outils et extensions pour enrichir et étendre les fonctionnalités de base d'ArgoCD, répondant à divers besoins autour du déploiement, de la gestion et de l'automatisation des applications Kubernetes. On y retrouve parmi eux quelques outils :

- **ArgoCD Notifications**
 - Envoie des notifications configurables pour les événements d'ArgoCD via divers canaux comme Slack, Email, et autres.
- **ArgoCD Image Updater**
 - Automatise la mise à jour des images de conteneurs dans les déploiements gérés par ArgoCD, basé sur des règles définies.
- **ArgoCD Autopilot**
 - Simplifie le démarrage avec GitOps et ArgoCD en automatisant la configuration initiale et la gestion de projets GitOps.
- **ArgoCD Vault Plugin**
 - Intègre ArgoCD avec HashiCorp Vault pour une gestion sécurisée des secrets dans les configurations d'application.
- **ArgoCD Exporter**
 - Fournit des métriques Prometheus pour ArgoCD, permettant de surveiller la santé et la performance de l'instance ArgoCD.
- **ArgoCD Rollouts**
 - Offre des capacités avancées de déploiement, telles que Blue/Green et Canary, pour Kubernetes, intégrables avec ArgoCD pour des stratégies de déploiement plus fines.
- **Argo Workflows**
 - Un moteur de workflow pour Kubernetes qui permet d'exécuter des tâches parallèles et séquentielles dans un cluster, souvent utilisé en tandem avec ArgoCD pour les pipelines CI/CD.

Il en existe d'autres dont la liste complète et détaillée ce trouve ici :

<https://github.com/argoproj>

Dans la même collection

Orchestration et Gestion de Conteneurs



Infrastructure as Code



Sécurité & Gestion des secrets



Développement & CI/CD



↓ FOLLOW ME ↓



[ANTONYCANUT](#)



[ANTONY KERVAZO-CANUT](#)