Menu

# GitOps

## Introduction

### Introduction

GitOps Use Cases

GitOps Advantages

Alauda Container Platform GitOps Advantages

## Install

### Installing Alauda Build of Argo CD

Prerequisites

Procedure

### Installing Alauda Container Platform GitOps

Prerequisites

Installing Alauda Container Platform GitOps cluster plugin

## Upgrade

### Upgrading Alauda Container Platform GitOps

Prerequisites

Upgrading Alauda Container Platform GitOps cluster plugin

## Architecture

### Architecture

GitOps and Argo CD

GitOps Architecture

Alauda Container Platform GitOps Architecture

## Concepts

### GitOps

Introduction

Core Principles

Advantages

Popular GitOps Tools

### Argo CD Concept

### Alauda Container Platform GitOps Concepts

## Guides

### Creating GitOps Application

### GitOps Observability

## How To

### Integrating Code Repositories via Argo CD dashboard

Use Cases

Prerequisites

Procedure

Operation Result

### Creating an Argo CD Application via Argo CD dashboard

Prerequisites

Procedure

### Creating an Argo CD Application via the web console

Use Cases

Prerequisites

Procedure

### How to Obtain Argo CD Access Information

Use Cases

How to Obtain Argo CD Access Information for the GitOps cluster plugin installed on the web console?

How to Obtain Argo CD Access Information from Argo CD Operator?

---

## Troubleshooting

### Troubleshooting

I've deleted/corrupted my repo and can't delete my app?

Why is my application still `OutOfSync` immediately after a successful Sync?

Why is my application stuck in `Progressing` state?

How to disable admin user?

Argo CD cannot deploy Helm Chart based applications without internet access, how can I solve it?

After creating my Helm application with Argo CD I cannot see it with helm ls and other Helm commands?

I've configured cluster secret but it does not show up in CLI/UI, how do I fix it?

Why Is My App Out Of Sync Even After Syncing?

How often does Argo CD check for changes to my Git or Helm repository?

How Do I Fix invalid cookie, longer than max length 4093?

Why Am I Getting rpc error: code = Unavailable desc = transport is closing When Using The CLI?

Why are resources of type SealedSecret stuck in the Progressing state?

How to rotate Redis keys?

# Introduction

GitOps is a modern approach to continuous delivery and operations that leverages Git as the central "Single Source of Truth" (SSOT) for defining and managing infrastructure, application configurations, and deployment workflows. By consolidating application code, configuration files, and Infrastructure as Code (IaC) definitions within a Git repository, GitOps enables comprehensive version control and automated governance of the entire software delivery lifecycle. In this paradigm, development and operations teams collaborate seamlessly throughout the software development, testing, and deployment phases using Git's robust branching, code review, and merge request mechanisms. When changes to code or configurations are pushed to the Git repository, automated tools detect these updates and initiate a cascade of automated processes, including building, testing, and deployment. This workflow facilitates the continuous delivery and continuous deployment (CI/CD) of software, ensuring rapid and reliable releases.

## TOC



## GitOps Use Cases

- **Continuous Delivery of Containerized Applications**: Within a Kubernetes ecosystem, GitOps excels at managing the deployment, updates, and rollbacks of containerized applications. Developers commit application code and Kubernetes configuration files to the

Git repository, and GitOps tools subsequently automate the deployment of these applications to the Kubernetes cluster, synchronizing them with any configuration file modifications.

- **Multi-Environment Management**: GitOps simplifies the management of infrastructure and application configurations across disparate environments, such as development, testing, staging, and production. Through strategic branching and environment tagging, it maintains configuration consistency while accommodating necessary customizations for each environment. Operations of Microservices Architecture: In microservices architectures, GitOps aids teams in efficiently orchestrating the deployment and updates of numerous microservices. Each microservice's code and configurations can be independently stored in the Git repository, allowing GitOps tools to automate deployments and updates based on microservice dependencies and update strategies, thereby ensuring system stability.
- **Infrastructure as Code (IaC) Management**: GitOps seamlessly integrates with IaC tools like Terraform and Ansible to manage cloud infrastructure, server configurations, and network resources. Storing IaC configuration files in a Git repository enables version-controlled and automated infrastructure deployments, enhancing manageability and repeatability.
- **Cross-Team Collaboration and Code Sharing**: In large organizations, multiple teams often need to share code and configurations. GitOps provides a unified platform for teams to collaborate on development, share code, and manage configurations via the Git repository, boosting collaboration efficiency and code reuse.

# GitOps Advantages

- Accelerated Collaboration & Delivery
- Rapid Rollback & Recovery
- Multi-Environment Governance
- Enhanced Security & Compliance

GitOps advantages detailed introduction

# Alauda Container Platform GitOps Advantages

- Enterprise-Grade Argo CD Operator.
- Argo CD Operator Safety Service.
- Visual GitOps Application Multi-Environment Distribution Management.
- Visual GitOps Application Operations and Maintenance.
- Visual GitOps Cluster Configuration Management.
- Closed-Loop GitOps Application Management Integrated with All Platform Products.

Alauda Container Platform GitOps advantages detailed introduction

Menu

# Install

## Installing Alauda Build of Argo CD

Prerequisites

Procedure

## Installing Alauda Container Platform GitOps

Prerequisites

Installing Alauda Container Platform GitOps cluster plugin

# Installing Alauda Build of Argo CD

## TOC



## Prerequisites

1. **Download** the **Alauda Build of Argo CD** Operator installation package corresponding to your platform architecture.
2. **Upload** the installation package using the Upload Packages mechanism.

## Procedure

> Install to the cluster where you want to use GitOps functionality.

### Install Alauda Build of Argo CD Operator

1. Login, go to the **Administrator** page.

2. Click **Marketplace** > **OperatorHub** to enter the **OperatorHub** page.
3. Find the **Alauda Build of Argo CD** Operator, click **Install**, and navigate to the **Install Argo CD** page.

Configuration Parameters:

| Parameter | Recommended Configuration |
| --- | --- |
| **Channel** | The default channel is `alpha`. |
| **Installation Mode** | `Cluster`: All namespaces in the cluster share a single Operator instance for creation and management, resulting in lower resource usage. |
| **Namespace** | Select `Recommended Namespace`: Automatically created if none exists. |
| **Upgrade Strategy** | `Auto`: The OperatorHub will automatically upgrade the Operator to the latest version when a new version is available. |

4. It is recommended to use the suggested default configuration; simply click **Install** to complete the **Alauda Build of Argo CD** Operator installation.

## Create Argo CD Instance

1. Click **Marketplace** > **OperatorHub**.
2. Find the **Alauda Build of Argo CD** Operator, click it to enter the **Argo CD** detail info page.
3. Click **All Instances**,
4. Click **Create Instance**, select **Argo CD** instance card.
5. Click **Create Instance**

> **INFO**
>
> In the configuration instance parameter page, use the default configuration unless there are specific requirements. **Note**: If the global cluster is not highly available (e.g., it has only one control node), please switch to YAML view when creating the instance and set the ha.enabled field value to false.

6. Click **Create**.

## Create AppProject Instance

> **INFO**
>
> **Tip**: If you do not need to use the platform-managed **Cluster Configuration Management** feature, you do not need to perform the following steps.

1. Find the **Alauda Build of Argo CD** operator, click it to enter the **Alauda Argo CD** detail info page.

2. Click **All Instances**, **Create Instance**, select **AppProject** instance card.

3. Switch to YAML view, and overwrite the existing YAML content on the interface with the code below.

```
apiVersion: argoproj.io/v1alpha1
kind: AppProject
metadata:
  name: cpaas-system
  namespace: argocd
spec:
  clusterResourceWhitelist:
  - group: '*'
    kind: '*'
  destinations:
  - namespace: '*'
    server: '*'
  sourceRepos:
  - '*'
```

4. Click **Create**.

After completing the above procedure, you have successfully installed Argo CD. Immediately Creating an Argo CD Application via Argo CD dashboard to begin your GitOps journey.

# Installing Alauda Container Platform GitOps

## TOC



## Prerequisites

1. **Download** the **Alauda Container Platform GitOps** cluster plugin installation package corresponding to your platform architecture.
2. **Upload** the installation package using the Upload Packages mechanism.
3. **Install** the installation package to the `global` cluster using the cluster plugins mechanism.

**INFO**

Upload Packages: **Administrator** > **Marketplace** > **Upload Packages** page. Click **Help Document** on the right to get instructions on how to publish the cluster plugin to `global` cluster. For more details, please refer to CLI.

# Installing Alauda Container Platform GitOps cluster plugin

## Constraints and Limitations

- Only supports installation in the `global` cluster.
- After the plugin is installed, the ArgoCD instance in the `argocd-operator` will be restricted from operations.

## Procedure

1. Login, go to the **Administrator** page.
2. Click **Marketplace** > **Cluster Plugins** to enter the **Cluster Plugins** list page.
3. Find the **GitOps** cluster plugin, click **Install**, and navigate to the **Install GitOps Plugin** page.
4. It is recommended to use the suggested default configuration; simply click **Install** to complete the **Alauda Container Platform GitOps** cluster plugin installation.

The parameter descriptions are as follows:

| Parameter | Description |
| --- | --- |
| **Native Argo CD UI** | Select whether to access the dashboard provided by Argo CD as needed. This interface includes features like monitoring, repository management, and settings, and can be used to manage and monitor the created applications. |
| **Single Sign-On** | It is recommended to enable SSO, which allows for quick access to the Argo CD native UI using platform account information, enhancing the login experience while also improving security and convenience.<br>**Note**: The SSO feature requires the Argo CD native UI feature to be enabled.<br>• Only supports access via HTTPS; SSO will not work if accessed via HTTP. |

| Parameter | Description |
| --- | --- |
| | • After enabling SSO and using the access address to open the Argo CD login interface, click the **LOG IN VIA OIDC** button in the interface for one-click login to the Argo CD native UI. |
| **Access Address** | Recommended: This address is dynamically generated based on the platform address for accessing the Argo CD Dashboad. No manual input is required. |
| **Account** | The account used to log in and access the Argo CD native UI. |
| **Password** | After enabling access to the Argo CD native UI, you can execute the following command in the global cluster's CLI tool to obtain it.<br>Obtain Argo CD Access Information |
| **Resource Quota** | The minimum requirements and recommendations for the platform are as follows:<br>• **Minimum**: CPU requests must not be less than 100 m, and memory requests must not be less than 250 Mi, and request values must not exceed limit values.<br>• **Recommended**: CPU requests should not be less than 250 m, and memory requests should not be less than 500 Mi; CPU limit values should not be less than 2 cores, and memory limit values should not be less than 2 Gi. |

## Verification

1. On the **Administrator** page, under the **Cluster** section in the left navigation, the **Config** entry will be displayed. You can use the capabilities of Cluster Configuration Management.
2. Access the **Container Platform**, the left navigation will display **GitOps Applications** entry, where you can create a GitOps application to immediately experience Creating an Argo CD Application via the web console.

Menu

# Upgrade

# Upgrading Alauda Container Platform GitOps

## TOC



## Prerequisites

1. **Download** the **Alauda Container Platform GitOps** cluster plugin installation package corresponding to your platform architecture.
2. **Upload** the **Alauda Container Platform GitOps** installation package using the Upload Packages mechanism.
3. **Install** the **Alauda Container Platform GitOps** cluster plugin to the `global` cluster using the cluster plugins mechanism.

**INFO**

Upload Packages: **Administrator** > **Marketplace** > **Upload Packages** page. Click **Help Document** on the right to get instructions on how to publish the cluster plugin to `global` cluster. For more details, please refer to CLI.

# Upgrading Alauda Container Platform GitOps cluster plugin

## Constraints and Limitations

- Only supports upgrading in the `global` cluster.

## Procedure

1. Login, go to the **Administrator** page.
2. Click **Clusters** > **Clusters** > **global** > **Functional Components** to enter the components list page.
3. Click the **Upgrade** button, and select the new version of **Alauda Container Platform GitOps** as the target in the **Confirm Component Upgrade** page.
4. Click the **Upgrade** again and the **Upgrade** in the dialog to confirm the upgrading.
5. (The following steps are only required if upgrading from `ACP 3.16.0`). Go back to the cluster `global` page, click **Actions** > **CLI Tools** to enter the CLI Window.
6. In the CLI window, input `kubectl delete cm -n argocd argocd-redis-ha-configmap` to recreate the `argocd-redis-ha-configmap`.
7. In the CLI window, input `kubectl get cm -n argocd argocd-redis-ha-configmap` to ensure the configmap is created.

## Verification

1. On the **Administrator** page, under the **Cluster** section in the left navigation, the **Config** entry will be displayed. You can use the capabilities of Cluster Configuration Management.

2. Access the **Container Platform**, the left navigation will display **GitOps Applications** entry, where you can create a GitOps application to immediately experience Creating an Argo CD Application via the web console.

# Architecture

## TOC



## GitOps and Argo CD

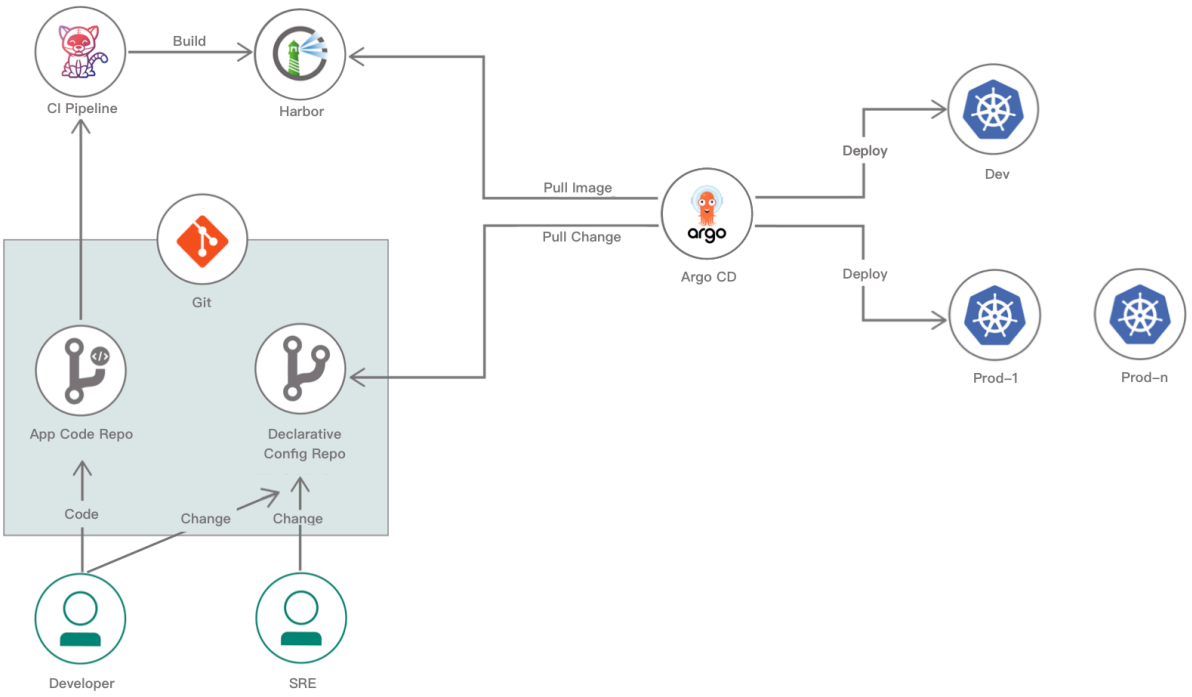
GitOps is a modern theory for continuous delivery and operations, while Argo CD is a powerful tool that implements GitOps by monitoring configuration files in a Git repository and automatically synchronizing them to the target environment. This approach improves software delivery speed, reliability, and security by incorporating the entire delivery process into the Git version control system.

**Alauda Container Platform GitOps**, built on Argo CD, uses the Git repository as the sole trusted source to store application, infrastructure configuration, and other files for rapid and accurate distribution and deployment to one or multiple Kubernetes clusters.

## GitOps Architecture

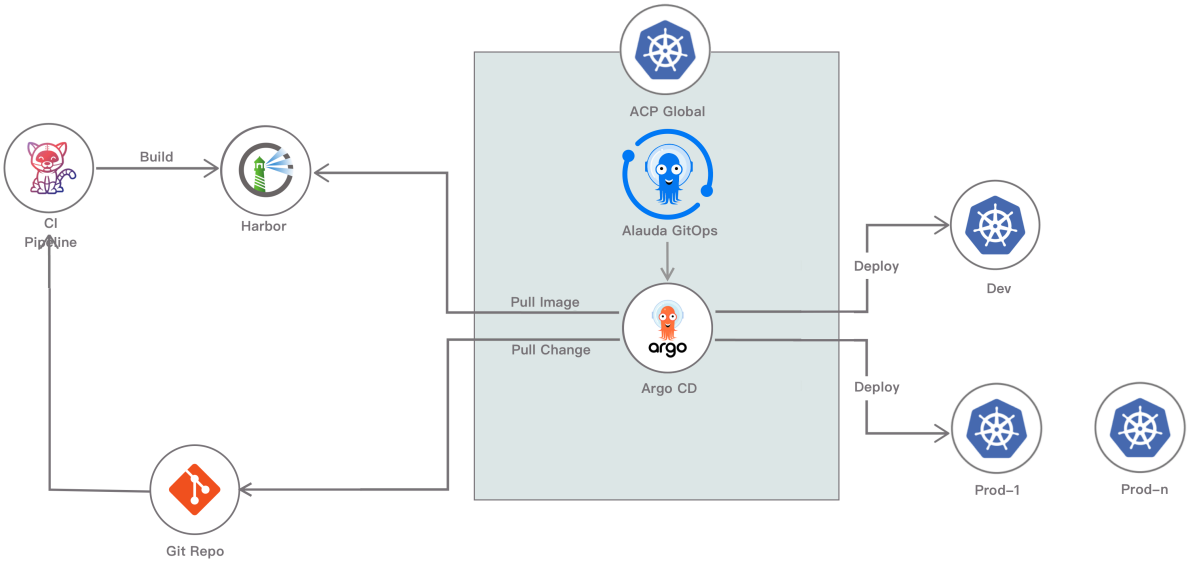
The main differences between GitOps and traditional application management methods are:

- Instead of directly manipulating the runtime environment, GitOps controls it by maintaining an application configuration repository on Git.
- Argo CD continuously pulls the repository and corrects discrepancies between the runtime environment and the application configuration repository, ensuring the environment meets expectations, preventing configuration drift, and enabling rapid recovery in case of failure.




# Alauda Container Platform GitOps Architecture

**Alauda Container Platform GitOps** is installed as a cluster plugin on the `global` cluster and utilizes Argo CD for application distribution and infrastructure provisioning across multiple business clusters.


ACP Global
Alauda GitOps
CI
Pipeline
Build
Harbor
Pull Image
Pull Change
argo
Argo CD
Deploy
Dev
Deploy
Prod-1
Prod-n
Git Repo

Menu

# Concepts

## GitOps

### GitOps

Introduction

Core Principles

Advantages

Popular GitOps Tools

## Argo CD Concept

### Introduction

Summary of Differences Between Application and ApplicationSet

Argo CD Sync Statuses

References

### Application

Introduction

Use Cases for Application

Application Example

Reference

## ApplicationSet

Introduction

Use Cases for ApplicationSet

ApplicationSet Example

References

## Tool

Introduction

Supported Tools

Development Workflow

Feature Comparison

References

## Helm

Introduction

Core Concepts of Helm

Advantages

Use Cases

## Kustomize

Introduction

Core Concepts of Kustomize

Advantages

Use Cases

### Directory

Introduction

Advantages

Use Cases

### Sync

Sync Overview

Sync Status Overview

Sync operation status Overview

Refresh Overview

References

### Health

Introduction

Health Scope

Reference

## Alauda Container Platform GitOps Concepts

### Introduction

Why Argo CD?

Advantages

# Alauda Container Platform GitOps Sync and Health Status

# GitOps

## TOC



## Introduction

GitOps is the practice of using a Git repository as the authoritative source for infrastructure and application configurations. All operational changes are version-controlled, automated, and auditable through Git. It relies on declarative configurations stored in Git, where any modifications must be committed to trigger automated deployment processes.

## Core Principles

- **Declarative Configuration**: GitOps fundamentally requires declarative tools, treating Git as the single source of truth. This enables consistent application deployment across Kubernetes clusters and platform-agnostic recovery in case of failures.
- **Versioned & Immutable State**: Infrastructure and application versions are directly mapped to Git commits. Rollbacks are executed via `git revert`, ensuring immutable version history.

- **Automated Reconciliation**: Merged declarative states are automatically applied to clusters. This eliminates manual intervention, prevents human errors, and supports security approvals in deployment workflows.
- **Self-Healing**: Controllers (e.g., Argo CD) continuously reconcile cluster states with Git-defined states, enabling autonomous system recovery.

# Advantages

- **Accelerated Collaboration & Delivery**: Declarative definitions of infrastructure, configurations, and target states stored in Git enable automated deployments. Teams achieve one-click environment provisioning post-validation, streamlining collaboration and delivery.
- **Rapid Rollback & Recovery**: Leveraging Git's version control, anomalies trigger instant rollbacks. GitOps controllers ensure self-healing through automated reconciliation.
- **Multi-Environment Governance**: Git as the single source of truth, combined with configuration overlays, enables precise bulk deployments across hybrid/multi-cloud environments.
- **Enhanced Security & Compliance**: Git's RBAC, audit logs, branch protections, and encryption secure sensitive configurations, ensuring compliance.

# Popular GitOps Tools

- **Argo CD**: A Kubernetes-native declarative GitOps tool for defining, versioning, and automating application lifecycles with auditability.
- **Flux**: A lightweight Kubernetes GitOps operator that continuously syncs Git repositories to clusters.
- **Jenkins X**: A CI/CD platform with GitOps integration for automated pipelines and Git-driven deployments.

Menu

# Argo CD Concept

## Tool

Introduction

Supported Tools

Development Workflow

Feature Comparison

References

## Helm

Introduction

Core Concepts of Helm

Advantages

Use Cases

## Kustomize

Introduction

Core Concepts of Kustomize

Advantages

Use Cases

## Directory

Introduction

Advantages

Use Cases

## Sync



## Health

# Introduction

Argo CD is a very popular open-source GitOps tool. To use Argo CD, you need to understand the following core concepts:

1. Application: A group of Kubernetes resources as defined by a manifest. This is a Custom Resource Definition (CRD). Application
2. ApplicationSet: A Kubernetes controller supporting the ApplicationSet CRD, enabling bulk generation of Applications from a single template. Think of it as an Application factory that creates instances based on parameters. ApplicationSet
3. Tool: Specifies the configuration management tool for Application sources (e.g., Kustomize, Helm). Tool
4. Sync: The process of reconciling an application's live state with its desired state (e.g., applying changes to Kubernetes clusters). Sync
5. Health: Indicates an application's operational status, including readiness and ability to serve requests. Health

## TOC

# Summary of Differences Between Application and ApplicationSet

| Attribute | Application | ApplicationSet |
|---|---|---|
| Definition | Single application deployment | Template for generating multiple Application instances. |
| Configuration | Static YAML definitions | Dynamic parameter-driven template generation |
| Deployment | Single application | Multiple similar applications |
| Use Cases | Simple single-environment deployments | Complex multi-environment/cluster deployments requiring parameterized instances |
| Core Concepts | Git Repo, target cluster, deployment strategies | Generators, templates, parameters, placeholders |
| Role in Argo CD. | Fundamental deployment unit | Advanced bulk management layer |

# Argo CD Sync Statuses

| Sync Status | Description |
|---|---|
| **Synced** | Application's live state fully matches desired state. |
| **OutOfSync** | Live state diverges from desired state; synchronization required. |
| **Syncing** | Active synchronization in progress; live state converging to desired state. |

# References

- Argo CD Official Documentation ↗

# Application

## TOC



## Introduction

Application is a group of Kubernetes resources as defined by a manifest. This is a Custom Resource Definition (CRD).

## Use Cases for Application

- Single-Component Deployment: Use Application CRD to declaratively manage deployment of atomic workloads within a single namespace.
- Static Configuration: Ideal for applications with deterministic manifests that don't require dynamic templating or multi-environment variations.
- Monocluster Deployment: Targeted deployment to individual Kubernetes clusters through GitOps workflows.

# Application Example

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: guestbook
  namespace: argocd
  finalizers:
    - resources-finalizer.argocd.argoproj.io
  labels:
    name: guestbook
spec:
  project: default

  source:
    repoURL: https://github.com/argoproj/argocd-example-apps.git
    targetRevision: HEAD
    path: guestbook

    chart: chart-name
    helm:
      passCredentials: false
      parameters:
      - name: "nginx-ingress.controller.service.annotations.external-
dns\\.alpha\\.kubernetes\\.io/hostname"
        value: mydomain.example.com
      - name: "ingress.annotations.kubernetes\\.io/tls-acme"
        value: "true"
        forceString: true

      fileParameters:
      - name: config
        path: files/config.json

      releaseName: guestbook

      valueFiles:
      - values-prod.yaml

      ignoreMissingValueFiles: false

      values: |
        ingress:
          enabled: true
          path: /
```

```
        hosts:
          - mydomain.example.com
        annotations:
          kubernetes.io/ingress.class: nginx
          kubernetes.io/tls-acme: "true"
        labels: {}
        tls:
          - secretName: mydomain-tls
            hosts:
              - mydomain.example.com

    valuesObject:
      ingress:
        enabled: true
        path: /
        hosts:
          - mydomain.example.com
        annotations:
          kubernetes.io/ingress.class: nginx
          kubernetes.io/tls-acme: "true"
        labels: {}
        tls:
          - secretName: mydomain-tls
            hosts:
              - mydomain.example.com

    skipCrds: false
    skipSchemaValidation: false
    version: v2
    kubeVersion: 1.30.0
    apiVersions:
      - traefik.io/v1alpha1/TLSOption
      - v1/Service
    namespace: custom-namespace

  kustomize:
    version: v3.5.4
    namePrefix: prod-
    nameSuffix: -some-suffix
    commonLabels:
      foo: bar
    commonAnnotations:
      beep: boop-${ARGOCD_APP_REVISION}
    commonAnnotationsEnvsubst: true
```

```
    forceCommonLabels: false
    forceCommonAnnotations: false
    images:
    - gcr.io/heptio-images/ks-guestbook-demo:0.2
    - my-app=gcr.io/my-repo/my-app:0.1
    namespace: custom-namespace
    replicas:
    - name: kustomize-guestbook-ui
      count: 4
    components:
      - ../component
    patches:
      - target:
          kind: Deployment
          name: guestbook-ui
        patch: |-
          - op: add
            path: /spec/template/spec/nodeSelector/
            value:
              env: "pro"

    kubeVersion: 1.30.0
    apiVersions:
      - traefik.io/v1alpha1/TLSOption
      - v1/Service

  directory:
    recurse: true
    jsonnet:
      extVars:
      - name: foo
        value: bar
      - code: true
        name: baz
        value: "true"
      tlas:
      - code: false
        name: foo
        value: bar
    exclude: 'config.yaml'
    include: '*.yaml'

  plugin:
    name: mypluginname
```

```
    env:
      - name: FOO
        value: bar
    parameters:
      - name: string-param
        string: example-string
      - name: array-param
        array: [item1, item2]
      - name: map-param
        map:
          param-name: param-value

  sources:
    - repoURL: https://github.com/argoproj/argocd-example-apps.git
      targetRevision: HEAD
      path: guestbook
      ref: my-repo

  destination:
    server: https://kubernetes.default.svc
    namespace: guestbook

  info:
    - name: 'Example:'
      value: 'https://example.com'

  syncPolicy:
    automated:
      prune: true
      selfHeal: true
      allowEmpty: false
    syncOptions:
    - Validate=false
    - CreateNamespace=true
    - PrunePropagationPolicy=foreground
    - PruneLast=true
    - RespectIgnoreDifferences=true
    - ApplyOutOfSyncOnly=true
    managedNamespaceMetadata:
      labels:
        any: label
        you: like
      annotations:
        the: same
```

```
        applies: for
        annotations: on-the-namespace

    retry:
      limit: 5
      backoff:
        duration: 5s
        factor: 2
        maxDuration: 3m

  ignoreDifferences:
  - group: apps
    kind: Deployment
    jsonPointers:
    - /spec/replicas
  - kind: ConfigMap
    jqPathExpressions:
    - '.data["config.yaml"].auth'
  - group: "*"
    kind: "*"
    managedFieldsManagers:
    - kube-controller-manager
    name: my-deployment
    namespace: my-namespace
  revisionHistoryLimit: 10
```

# Reference

- Argo CD Official Documentation ↗

# ApplicationSet

## TOC



## Introduction

ApplicationSet controller is a Kubernetes controller that adds support for an ApplicationSet CustomResourceDefinition (CRD). This controller/CRD enables both automation and greater flexibility managing Argo CD Applications across a large number of clusters and within monorepos, plus it makes self-service usage possible on multitenant Kubernetes clusters.

## Use Cases for ApplicationSet

- Deploying multiple similar applications: When you need to deploy multiple applications with similar configurations, you can use ApplicationSet to reduce redundant configurations. For example, you could use ApplicationSet to deploy multiple microservices that utilize the same template, but have different service names and port numbers.
- Multi - cluster deployments: When you need to deploy the same application across multiple Kubernetes clusters, you can use ApplicationSet to simplify configuration. For instance, you

could define an application with ApplicationSet and deploy it across multiple clusters, each using different parameters.

- Dynamically generating applications: When you need to dynamically generate applications based on certain conditions, ApplicationSet can be utilized. For example, you could dynamically generate different application instances based on branches or tags in a Git repository.

# ApplicationSet Example

```yaml
apiVersion: argoproj.io/v1alpha1
kind: ApplicationSet
metadata:
  name: guestbook
spec:
  goTemplate: true
  goTemplateOptions: ["missingkey=error"]
  generators:
  - list:
      elements:
      - cluster: engineering-dev
        url: https://1.2.3.4
      - cluster: engineering-prod
        url: https://2.4.6.8
      - cluster: finance-preprod
        url: https://9.8.7.6
  template:
    metadata:
      name: '{{.cluster}}-guestbook'
    spec:
      project: my-project
      source:
        repoURL: https://github.com/infra-team/cluster-deployments.git
        targetRevision: HEAD
        path: guestbook/{{.cluster}}
      destination:
        server: '{{.url}}'
        namespace: guestbook
```

# References

- Argo CD ApplicationSet Documentation ↗

# Tool

## TOC



## Introduction

**Tool** refers to a utility used to generate or process Kubernetes resource `Manifests`.

## Supported Tools

Argo CD supports several Kubernetes manifest definition approaches:

- **Kustomize Applications** Kustomize
- **Helm Charts** Helm
- **Directory**: Manifests containing `YAML` / `JSON` / `Jsonnet` files, including `Jsonnet` Directory
- **Custom Configuration Management Plugins**: Any custom tool configured as a Config Management Plugin

# Development Workflow

Argo CD allows direct upload of local `manifests`, but this is intended **for development purposes only**. Overriding requires users with permissions (typically administrators) to upload local `manifests`. It supports all aforementioned Kubernetes deployment tools. To upload a local application:

```
argocd app sync APPNAME --local /path/to/dir/
```

# Feature Comparison

| Feature | Helm | Kustomize | Directory (Pure YAML) |
|---|---|---|---|
| **Configuration Method** | Templating (dynamic generation) | Declarative (patches and overlays) | Static YAML files |
| **Reusability** | High (via Charts) | Medium (via base/overlay) | Low |
| **Multi-Environment Support** | High (via values.yaml) | High (via overlays) | Low |
| **Progressive Delivery** | High (complex logic support) | Medium (simple patch support) | Low |
| **Learning Curve** | High (template syntax) | Low (YAML-based) | Low |
| **Argo CD Integration** | Supported | Native Support | Supported |
| **Use Cases** | Complex apps, multi-environment, | Multi-environment, config reuse | Small projects, rapid |

| Feature | Helm | Kustomize | Directory (Pure YAML) |
| --- | --- | --- | --- |
| | distribution | | prototyping |

# References

For more detailed information, please refer to: Tool ↗

# Helm

## TOC



## Introduction

Helm is a package management tool for Kubernetes, enabling users to define, install, and upgrade complex Kubernetes applications. A **Helm Chart** is a templated configuration package containing Kubernetes resource definitions (YAML files).

## Core Concepts of Helm

- **Chart**: A Helm Chart is a templated configuration package containing Kubernetes resource definitions (YAML files).
- **Release**: A Helm Release is an instance of a deployed Helm Chart, representing a specific configuration of Kubernetes resources.
- **Values**: Helm Values are parameterized configurations for a Helm Chart, allowing users to customize Kubernetes resource definitions.

Argo CD's integration with Helm enhances GitOps practices by enabling declarative continuous delivery through web console, Argo CD dashboard, or CLI. Example:

```yaml
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: sealed-secrets
  namespace: argocd
spec:
  project: default
  source:
    chart: sealed-secrets
    repoURL: https://bitnami-labs.github.io/sealed-secrets
    targetRevision: 1.16.1
    helm:
      releaseName: sealed-secrets
  destination:
    server: "https://kubernetes.default.svc"
    namespace: kubeseal
```

OCI Helm Chart Example:

```yaml
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: nginx
spec:
  project: default
  source:
    chart: nginx
    repoURL: registry-1.docker.io/bitnamicharts  # note: the oci:// syntax is not
included.
    targetRevision: 15.9.0
  destination:
    name: "in-cluster"
    namespace: nginx
```

**INFO**

** The Application's lifecycle is managed by Argo CD, not Helm. ** When multiple value sources are provided, the priority order is: `parameters` > `valuesObject` > `values` > `valueFiles` > helm repository `values.yaml`.

# Advantages

- **Templating**: Helm uses the Go template engine (gotpl) to dynamically generate Kubernetes resource files.
- **Package Management**: Helm packages applications as Charts (including templates, default values, and dependencies), simplifying distribution and version control.
- **Dependency Management**: Supports dependencies between Charts.
- **Lifecycle Management**: Provides commands like install, upgrade, and rollback for full lifecycle management.

# Use Cases

- **Complex Application Deployment**: Ideal for scenarios requiring dynamic configuration generation (e.g., environment variables or user input).
- **Multi-Environment Deployments**: Supports environment-specific configurations via values.yaml files.
- **Application Distribution**: Enables packaging Charts for distribution to Helm repositories or OCI registries.

# References

For more detailed information, please refer to: Helm ↗

# Kustomize

## TOC



## Introduction

Kustomize is a Kubernetes-native configuration management tool that enables users to customize Kubernetes resource definitions (YAML files) through overlays and composition without directly modifying original files.

## Core Concepts of Kustomize

- **Base**: Base configurations containing common Kubernetes resource definitions.
- **Overlay**: Customization layers that modify Base configurations.
- **kustomization.yaml**: A configuration file defining how resources are composed and modified.

Argo CD's integration with Kustomize enhances GitOps practices by enabling declarative continuous delivery. Example:

```yaml
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: kustomize-example
spec:
  project: default
  source:
    path: examples/helloWorld
    repoURL: 'https://github.com/kubernetes-sigs/kustomize'
    targetRevision: HEAD
  destination:
    namespace: default
    server: 'https://kubernetes.default.svc'
```

If a `kustomization.yaml` file exists at the `repoURL` and `path` location, Argo CD will render manifests using Kustomize.

Kustomize supports the following configuration options:

- `namePrefix`: Prefix appended to Kustomize-generated resource names.
- `nameSuffix`: Suffix appended to Kustomize-generated resource names.
- `images`: List of Kustomize image overrides.
- `replicas`: List of Kustomize replica overrides.
- `commonLabels`: Map of labels added to all resources.
- `labelWithoutSelector`: Boolean defining whether common labels should apply to resource selectors and templates.
- `forceCommonLabels`: Boolean allowing override of existing labels.
- `commonAnnotations`: Map of annotations added to all resources.
- `namespace`: Kubernetes resource namespace.
- `forceCommonAnnotations`: Boolean allowing override of existing annotations.
- `commonAnnotationsEnvsubst`: Boolean enabling environment variable substitution in annotation values.
- `patches`: List of Kustomize patches supporting inline updates.
- `components`: List of Kustomize components.

To use Kustomize with overlays, point your path to the overlay directory.

## Advantages

- Declarative Configuration: Uses YAML files (via kustomization.yaml) to define resource composition and modifications.
- Template-Free: Customizes configurations through patches and overlays without template engines.
- Kubernetes-Native Integration: Kustomize is built into kubectl, requiring no additional tools.

## Use Cases

- Multi-Environment Distribution: Achieve environment-specific configurations (e.g., apps, clusters) via Base and Overlay.
- Configuration Reuse: Ideal for reusing base configurations across projects.
- Progressive Delivery: Gradually adjust resource configurations through patches.

## References

For more detailed information, please refer to: Kustomize ↗

# Directory

## TOC



## Introduction

**Directory** type application loads `manifests` directly from `.yml`, `.yaml`, or `.json` files. Directory applications can be created via the platform UI, Argo CD Dashboard, CLI, or declaratively. Example declarative syntax:

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: guestbook
spec:
  destination:
    namespace: default
    server: https://kubernetes.default.svc
  project: default
  source:
    path: guestbook
    repoURL: https://github.com/argoproj/argocd-example-apps.git
    targetRevision: HEAD
```

No spec.source.directory field is required unless additional configuration options are needed. Argo CD automatically detects whether the source repository/path contains plain manifest files.

## Advantages

- **Simplicity**: Directly loads resources from manifest files without additional abstraction.
- **Low Maintenance**: No configuration management overhead.

## Use Cases

- Managing multiple Kubernetes resources (e.g., Deployments, Services, ConfigMaps).
- Small-scale projects, minimal resources, or rapid GitOps adoption.
- Deploying raw YAML files without dynamic templating or complex configuration management.

> **WARNING**
>
> Directory type applications **only support plain manifest files**. If Argo CD detects `Kustomize`, `Helm`, or `Jsonnet` files in a Directory path, it will fail to render manifests.

# References

For more detailed instructions, refer to: Directory ↗

# Sync

## TOC



## Sync Overview

Sync is the core functionality of Argo CD, responsible for comparing the **Desired State** of an application with its **Live State** and taking actions to reconcile discrepancies. In essence, Sync ensures that the state of applications in your Kubernetes cluster aligns with the state defined in the Git repository.
You can trigger Sync manually or configure Argo CD to perform it automatically. Auto-Sync can be triggered by monitoring Git repository changes (e.g., commits, tag pushes) or executed at scheduled intervals.

## Sync Status Overview

Sync Status indicates the synchronization state of an application, reflecting whether its **Live State** matches the **Desired State**. Sync Status includes the following states:

- `Synced`: The application's **Live State** exactly matches the **Desired State**.
- `OutOfSync`: The application's **Live State** diverges from the **Desired State**.
- `Syncing`: The application is undergoing synchronization, with the **Live State** converging toward the **Desired State**.

# Sync operation status Overview

Sync Operation Status represents the execution state of a synchronization operation by Argo CD, indicating whether the operation completed successfully. Sync operation status includes the following states:

- `Succeeded`: The synchronization operation completed successfully.
- `Failed`: The synchronization operation failed due to reasons such as Kubernetes resource conflicts, insufficient permissions, etc.
- `Running`: The synchronization operation is in progress.

# Refresh Overview

This operation fetches the latest application configuration from the Git repository and compares it against the actual state in the Kubernetes cluster. Refresh can be triggered manually or configured for automatic execution at defined intervals.

# References

For more detailed information, please refer to: Sync ↗

# Health

## TOC



## Introduction

The health of the application, is it running correctly? Can it serve requests?

## Health Scope

| Health Status | Description |
| --- | --- |
| **Health** | The resource is healthy. |
| **Progressing** | The resource is not healthy yet but still making progress and might be healthy soon. |
| **Degraded** | The resource is degraded. |

| Health Status | Description |
| --- | --- |
| **Suspended** | The resource is suspended and waiting for some external event to resume (e.g. suspended CronJob or paused Deployment). |

# Reference

Argo CD provides built-in health assessment for several standard Kubernetes types, which is then surfaced to the overall Application health status as a whole. Of course, Argo CD also supports custom health checks.

For more detailed explanations, please refer to: Health ↗

Menu

# Alauda Container Platform GitOps Concepts

# Introduction

Alauda Container Platform GitOps is a Kubernetes-native GitOps solution built on Argo CD. It monitors configuration manifests (applications, infrastructure definitions, etc.) in Git repositories and automatically synchronizes them to target Kubernetes clusters, implementing Git-driven continuous delivery. By codifying the entire delivery pipeline in Git's version control system, it enhances deployment velocity, reliability, and security while enabling precise multi-cluster application distribution.

The solution natively integrates the Argo CD Operator to automate deployment lifecycle operations including provisioning, upgrades, and rollbacks.

## TOC



## Why Argo CD?

Argo CD stands as the industry-leading open-source GitOps engine due to its distinctive advantages:

| Technical Advantages | Operational Benefits |
| --- | --- |
| **Declarative GitOps Engine** | **Accelerated Deployment** |

| Technical Advantages | Operational Benefits |
| --- | --- |
| • State reconciliation via CRDs (Application, ApplicationSet)<br>• Multi-source support (Helm, Kustomize, raw YAML) | • 70% faster deployment cycles through Git-driven automation |
| **Kubernetes-Native Architecture**<br>• Deep integration with Kubernetes API server ≈ Native support for Namespace isolation and RBAC | **Enterprise Readiness**<br>• Built-in multi-tenancy and audit capabilities |
| **Multi-Cluster Management**<br>• Centralized control plane for hybrid/multi-cloud deployments<br>• Cluster-specific configuration through ApplicationSets | **Operational Efficiency**<br>• 60% reduction in deployment errors through declarative enforcement |
| **Extensible Plugin System**<br>• Certified integrations with Helm, Kustomize, Istio<br>• Custom Resource Definitions (CRDs) for advanced workflows | **Cost Optimization**<br>• 40% lower cloud costs through precise resource orchestration |
| **Active CNCF Ecosystem**<br>• 3,500+ GitHub stars<br>• 200+ active contributors | **Future-Proof**<br>• Continuous innovation through open-source community |

# Advantages

> In addition to the inherent advantages of GitOps, Alauda Container Platform GitOps offers the following enhanced benefits:

- **Enterprise-Grade Argo CD Operator**
  - Delivers the full suite of functionalities from the native Argo CD Operator, covering application deployment, upgrades, rollbacks, and all core features of Argo CD.
- **Argo CD Operator Safety Service**
  - Offers dedicated technical support for the Argo CD Operator, addressing fault responses, security vulnerability patches, and overall system stability.
- **Visual GitOps Application Multi-Environment Distribution Management**
  - Leverages the platform's multi-cluster management and differentiated configuration capabilities to achieve style-consistent, visual GitOps application management and cluster configuration management, simplifying precise distribution across multi-cloud and multi-environment setups.
- **Visual GitOps Application Operations and Maintenance**
  - Grants direct access to real-time logs and events of Kubernetes Workload resources under GitOps applications. During GitOps application anomalies, users can swiftly analyze and resolve issues using Argo CD's anomaly information and real-time Workload logs without leaving the current interface.
- **Visual GitOps Cluster Configuration Management**
  - Manages cluster configurations through GitOps, achieving unified management and distribution of cluster configurations in a visual manner.
- **Closed-Loop GitOps Application Management Integrated with All Platform Products**
  - Integrates the platform's DevOps capabilities for continuous building, artifact management, and microservice gray release to realize fully automated GitOps application management, forming a complete closed loop of continuous integration and continuous delivery collaboration.
  - Combines with platform products like CrossPlane, MySQL, and Developer Portal to achieve a comprehensive process spanning infrastructure initialization, business

application initialization (including code, pipeline, GitOps application initialization, etc.), and application code development and deployment.

# Alauda Container Platform GitOps Sync and Health Status

Alauda Container Platform GitOps abstracts the state of `Application` resources by leveraging the status of underlying Kubernetes resources. The state of `Application` resources directly governs the state of associated `ApplicationSet` resources.

## TOC



## Sync Status Explanation

Both Kubernetes resources and applications have four sync states: **Sync Failed**, **OutOfSync**, **Syncing**, and **Synced**.

| Sync Status | Description |
| --- | --- |
| **Sync Failed** | Synchronization failed due to network errors, configuration issues, or permissions problems. Check logs for root cause. |

| Sync Status | Description |
| --- | --- |
| **OutOfSync** | Cluster resource state diverges from Git-defined desired state. Manual/auto sync required. |
| **Syncing** | Active reconciliation in progress between cluster state and Git-defined state. |
| **Synced** | Cluster resource state matches Git-defined desired state. |

**INFO**

**Sync Status Display Priority**: Priority order **Sync Failed** > **OutOfSync** > **Syncing** > **Synced**.

**Examples**:

- If an Application has two resources with **Syncing** and **Synced** statuses, its overall status is **Syncing**.
- If an ApplicationSet manages two Applications with **Sync Failed** and **Synced** statuses, its overall status is **Sync Failed**.

# Health Status Explanation

Kubernetes resources and applications have six health states: **Unknown**, **Missing**, **Degraded**, **Paused**, **Progressing**, and **Healthy**.

| Health Status | Description | Reference Solution |
| --- | --- | --- |
| **Unknown** | Unable to determine health state, typically due to controller errors or missing status data. | Inspect resource YAML's `status.conditions` for diagnostic details. |
| **Missing** | Resource not found in cluster. | Initial creation: Wait for reconciliation |

| Health Status | Description | Reference Solution |
| --- | --- | --- |
| | | Accidental deletion: Trigger manual sync. |
| **Degraded** | Workload resources (e.g., Deployment) failed to achieve healthy state within timeout period (default: 10 mins). | Investigate Pod failures (e.g., crashes, resource constraints). |
| **Paused** | Workload resources rollout intentionally paused (e.g., via `kubectl rollout pause`). | Resume rollout if appropriate. |
| **Processing** | Resource created successfully but not fully ready (e.g., Pods initializing). | Monitor until transition to Healthy/Degraded. |
| **Healthy** | Resource operating normally. | - |

**INFO**

**Health Status Priority**:Priority order **Unknown** > **Missing** > **Degraded** > **Paused** > **Progressing** > **Healthy**

**Examples**:

- If an Application has resources with **Healthy** and **Unknown** statuses, its overall health is **Unknown**.
- If an ApplicationSet manages Applications with **Missing** and **Progressing** statuses, its overall health is **Missing**.

# Recognition Rules

**Healthy** status recognition rules for Kubernetes resources:

| Resource Type | Status |
| --- | --- |
| **Deployment** | Rolling update completed with all replicas available. |
| **StatefulSet** | Update completed with all pods ready. |
| **ReplicaSet** | All Pods healthy. |
| **DaemonSet** | Desired number of Pods scheduled and healthy. |
| **Ingress** | LoadBalancer IP/hostname populated in status. |
| **Service** | LoadBalancer IP/hostname populated (if applicable). |
| **PVC** | Status is **Bound**. |
| **Pod** | All containers ready with no restarts exceeding thresholds. |
| **Job** | Job completed successfully ( `.status.succeeded >= 1` ). |
| **HPA** | Successful scaling operation with current replicas matching desired count. |

Menu

# Guides

## Creating GitOps Application

### Creating GitOps Application



### Creating GitOps ApplicationSet



## GitOps Observability

### Argo CD Component Monitoring

## GitOps Applications Ops

Menu

# Creating GitOps Application

# Creating GitOps Application

# Overview

Leverage **Alauda Container Platform GitOps** application management capabilities to visually create Argo CD ApplicationSet for comprehensive lifecycle management of containerized applications through **GitOps Applications**.

---

# TOC



---

# Prerequisites

- **Install Alauda Container Platform GitOps**:

- If not installed, please contact the Administrator to Installing Alauda Container Platform GitOps

- **Git Repository Integration** (Choose one method):

  - Integrating Code Repositories via Argo CD dashboard
  - The Administrator must provision Code Repositories through **DevOps Toolchain** > **Integrate**

# Creating Argo CD Application via web console

> Streamline application distribution through visual management interfaces.

## Procedure

1. **Container Platform**, and navigate to **GitOps Applications**.

2. Click on **Create GitOps Application**.

3. Configure parameters in **Basic Info** and **Code Repository** sections:

| Parameter | Description |
| --- | --- |
| **Type** | **Application**: Argo CD Application object for single namespace deployment<br>**ApplicationSet**: Argo CD ApplicationSet for cross-cluster/cross-namespace deployments with differential configurations |
| **Source** | **Platform integrated**: Pre-configured GitLab/GitHub/Bitbucket repositories<br>**ArgoCD integrated**: GitLab/GitHub/Bitbucket/Gitee/Gitea repositories integrated via Argo CD. Please refer to Integrating Code Repositories via Argo CD dashboard |
| **Integration Project Name** | Toolchain project assigned by the Administrator |

| Parameter | Description |
| --- | --- |
| **Version Identifiers** | Deployment basis: `Branch` / `Tag` / `Commit`<br>**Note**:<br>• `Branch` uses latest commit<br>• `Tag` / `Commit` defaults to latest but configurable |
| **Source File Type** | **Kustomize**: Uses kustomization.yaml for overlay configurations; for more details, please refer to the Kustomize Official Documentation ↗<br>**Helm**: Uses values.yaml for templating; for more details, please refer to the Helm Official Documentation ↗<br>**Directory**: Raw manifests |
| **Source Directory** | Repository path containing base manifests. Supports root directory selection. All resources in this path will be deployed to target clusters |
| **Custom Values** | **Source File Type** is **Helm**, you can select a custom Helm Values file |

4. Configure parameters in **Destination** sections:

   - **Application**: Differential configs don't modify base files in source directory.
   - **ApplicationSet**: Multi-cluster deployment with **Differentiated Configuration**.

   **Note**: **Differentiated Configuration** don't modify base files in **Source Directory**.

5. **Sync Policy** (3-minute reconciliation interval).

| Parameter | Description |
| --- | --- |
| **Manually Sync** | Requires user confirmation when drift detected. |
| **Automatic Sync** | Automatic reconciliation without human intervention. |
| **Sync Ignore Configuration** | Configure using built-in/custom ignore templates, you can View Sync Ignore Configuration Fields in YAML File.<br>**Note**: Custom templates require admin configuration. |

6. Click **Create**.

> **INFO**
>
> **Manual Sync Note**: Choose **Synchronize Immediately** for immediate deployment or **Synchronize Later** to trigger manually via details page.

## View Sync Ignore Configuration Fields in YAML file

After configuring sync ignore rules, verify via:

1. Navigate to **GitOps Application**
2. Select target application
3. Click **Action > Update**
4. Inspect YAML file.

```
ignoreDifferences: # The configuration actually ignored by the selected custom synchronization ignore configuration template
  - group: apps
    kind: Deployment
    jsonPointers:
      - /spec/replicas
```

# Creating Argo CD Application via YAML

## Procedure

1. **Container Platform**, and navigate to **GitOps Applications**.
2. Click on **Create GitOps Application**.
3. Switch to the **YAML** tab.

4. In the **YAML** sections, refer to the following YAML file and configure the relevant information. Replace `namespace` and `project` with your own namespace and project.

```yaml
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: guestbook
  namespace: argocd # Replace with your own namespace
spec:
  project: default # Replace with your own project
  source:
    repoURL: https://github.com/argoproj/argocd-example-apps.git
    targetRevision: master
    path: helm-guestbook
  destination:
    server: https://kubernetes.default.svc
    namespace: guestbook
  syncPolicy:
    automated:
      prune: true
      selfHeal: true
    syncOptions:
      - CreateNamespace=true
```

5. Click **Create**.

# Creating Argo CD Application via CLI

## Prerequisites

The web-cli plugin is installed and the `web-cli` switch is enabled.

```
kubectl apply -f application.yaml
```

# Creating GitOps ApplicationSet

## TOC



## Overview

Leverage **Alauda Container Platform GitOps** application management capabilities to visually create Argo CD ApplicationSet for comprehensive lifecycle management of containerized applications through **GitOps Applications**.

## Prerequisites

- **Installing Alauda Container Platform GitOps**:
  - If not installed, please contact the Administrator to Installing Alauda Container Platform GitOps

- **Git Repository Integration** (Choose one method):
  - Integrating Code Repositories via Argo CD dashboard
  - The Administrator must provision Code Repositories through **DevOps Toolchain** > **Integrate**

# Key Benefits

- **Visual GitOps Application Distribution**: Combines multi-cluster management, differential configurations, and platform-aligned visual operations for simplified multi-cloud/multi-environment deployments.

# Creating GitOps Application

> Streamline application distribution through visual management interfaces.

## Procedure

1. **Container Platform**, and navigate to **GitOps Applications**.
2. Click on **Create GitOps Application**.
3. Configure parameters in **Basic Info** and **Code Repository** sections:

| Parameter | Description |
| --- | --- |
| **Type** | **Application**: Argo CD Application object for single namespace deployment<br>**ApplicationSet**: Argo CD ApplicationSet for cross-cluster/cross-namespace deployments with differential configurations |
| **Source** | **Platform integrated**: Pre-configured GitLab/GitHub/Bitbucket repositories<br>**ArgoCD integrated**: GitLab/GitHub/Bitbucket/Gitee/Gitea |

| Parameter | Description |
| --- | --- |
| | repositories integrated via Argo CD. Please refer to Integrating Code Repositories via Argo CD dashboard |
| **Integration Project Name** | Toolchain project assigned by the Administrator |
| **Version Identifiers** | Deployment basis: `Branch` / `Tag` / `Commit`<br>**Note**:<br>• `Branch` uses latest commit<br>• `Tag` / `Commit` defaults to latest but configurable |
| **Source File Type** | **Kustomize**: Uses `kustomization.yaml` for overlay configurations; for more details, please refer to the Kustomize Official Documentation ↗<br>**Helm**: Uses values.yaml for templating; for more details, please refer to the Helm Official Documentation ↗<br>**Directory**: Raw manifests |
| **Source Directory** | Repository path containing base manifests. Supports root directory selection. All resources in this path will be deployed to target clusters |
| **Custom Values** | **Source File Type** is **Helm**, you can select a custom Helm Values file |

4. Configure parameters in **Destination** sections:

   - **Application**: Differential configs don't modify base files in source directory.
   - **ApplicationSet**: Multi-cluster deployment with **Differentiated Configuration**.

   **Note**: **Differentiated Configuration** don't modify base files in **Source Directory**.

5. **Sync Policy** (3-minute reconciliation interval).

| Parameter | Description |
| --- | --- |
| **Manually Sync** | Requires user confirmation when drift detected |

| Parameter | Description |
| --- | --- |
| **Automatic Sync** | Automatic reconciliation without human intervention |
| **Sync Ignore Configuration** | Configure using built-in/custom ignore templates, you can View Sync Ignore Configuration Fields in YAML File<br>**Note**: Custom templates require admin configuration |

6. Click **Create**.

> **INFO**
>
> **Manual Sync Note**: Choose **Synchronize Immediately** for immediate deployment or **Synchronize Later** to trigger manually via details page.

## View Sync Ignore Configuration Fields in YAML File

After configuring sync ignore rules, verify via:

1. Navigate to **GitOps Application**.
2. Select target application.
3. Click **Action** > **Update**.
4. Inspect YAML file.

```
ignoreDifferences: # The configuration actually ignored by the selected custom synchronization ignore configuration template
  - group: apps
    kind: Deployment
    jsonPointers:
      - /spec/replicas
```

# Managing GitOps Applications

| Action | Description |
| --- | --- |
| **Update** | Initiate updates via:<br>• Edit icon (✎) on **GitOps Application** list<br>• **Action** > **Update** in detail view.<br>• **CAUTION**: This operation will overwrite all created application instances |
| **Manually Sync** | When **Sync Policy** is **Manually Sync** :<br>• Trigger sync via **Action** > **Sync** in detail view upon detecting configuration drift<br>• Propagates latest commits to all managed instances |
| **Delete** | Delete via:<br>• Delete icon (🗑) on list page<br>• **Action** > **Delete** in detail view<br>• **DESTRUCTIVE**: Removes application and ALL child resources |
| **Automatic Sync** | Enable auto-reconciliation to maintain desired state. All instances automatically sync with repo changes every 3 minutes |
| **Source** | For **ApplicationSet** type apps:<br>• Click **Source** link to navigate to parent Application details page. |
| **Application Distribution** | Extend:<br>1. Update existing **ApplicationSet** config<br>2. In **ApplicationSet** details: **Applications** > **Add Distribution** |

Menu

# GitOps Observability

## Argo CD Component Monitoring

Overview

Prerequisites

Viewing the Argo CD component dashboard

## GitOps Applications Ops

Overview

Prerequisites

Alert

Logs

Events

# Argo CD Component Monitoring

## TOC



## Overview

The monitoring dashboard of the web console offers a visual approach to monitor Argo CD components. It proactively observes the resources and operational states of Argo CD components, aiming to ensure their healthiness and availability. Here, the operational states refer to the running conditions and performance metrics of the components in the Kubernetes (K8s) environment. By closely tracking these aspects, we can promptly detect and address any potential issues, maintaining the smooth operation of Argo CD within the K8s cluster.

## Prerequisites

- Installing Alauda Container Platform GitOps
- Installation of Monitoring Plugins

## Viewing the Argo CD component dashboard

1. Login, and navigate to **Administrator**, and select the `global` cluster.

2. Click on **Operations Center** > **Monitoring** > **Monitoring Dashboard**.

3. Click on **Switch** button and select **container-platform** to view the **ArgoCD** dashboard.

4. Click the **ArgoCD** dashboard to view the **Argo CD** component monitoring information.

# GitOps Applications Ops

## TOC



## Overview

The web console's **GitOps Applications** management capabilities enable viewing **GitOps Applications** monitoring, logs, and events. You can also create alerting policies for **GitOps Applications**. When anomalies occur in **GitOps Applications**, proactive alert notifications will be triggered to facilitate rapid issue identification, analysis, and resolution.

## Prerequisites

- A GitOps application has been created on the web console. Creating an Argo CD Application via the web console
- Installation of Monitoring Plugins

# Alert

> Create alerting rule in advance to configure rules. When GitOps applications encounter anomalies, proactive notifications will be triggered to enable quick issue identification, analysis, and resolution.

1. **Container Platform**, Click on **GitOps Applications**.
2. Select the GitOps application name from the list where you want to create an alert rule.
3. Switch to the **Alerts** tab.
4. Click on **Create Rule** and fill in the basic information as required.
5. Click on **Add Alert Condition**, navigate to the **Alert Conditions** page. The corresponding metric descriptions are as follows:

**INFO**

For other parameter configurations and alert settings, refer to Alert Management.

| Metric Name | Rule Description |
| --- | --- |
| GitOps Application Health Status<br>`gitops.applicationset.healthy` | Health status of GitOps application:<br>- 0: Unknown, Lost, Degraded, or Paused<br>- 1: Syncing<br>- 2: Healthy |
| GitOps Application Sync Status<br>`gitops.applicationset.synced` | Sync status of GitOps application:<br>- 0: Sync Failed or Pending<br>- 1: Syncing<br>- 2: Synced |

6. Click **Create**.

# Logs

> View logs for all workload resources created by GitOps applications. Logs enable rapid identification of system failure information without relying on **Log** cluster plugin.

- On the GitOps application details page, under **Kubernetes Resources**, click on any Workload name to view the **Logs** information for that resource on the right side.

# Events

> View events for all resources distributed by **GitOps Applications**. Events enable rapid identification of system failure event information without relying on **Log** cluster plugin.

- On the GitOps application details page, go to the **Events** tab to view aggregated events for all resources.
- On the GitOps application details page, under **Kubernetes Resources**, click any resource name to view events for that resource on the right side.

Menu

# How To

## Integrating Code Repositories via Argo CD dashboard

Use Cases

Prerequisites

Procedure

Operation Result

## Creating an Argo CD Application via Argo CD dashboard

Prerequisites

Procedure

## Creating an Argo CD Application via the web console

Use Cases

Prerequisites

Procedure

## How to Obtain Argo CD Access Information

Use Cases

How to Obtain Argo CD Access Information for the GitOps cluster plugin installed on the web console?

How to Obtain Argo CD Access Information from Argo CD Operator?

# Integrating Code Repositories via Argo CD dashboard

Use the native Argo CD dashboard to integrate code repositories and allocate repositories, enabling developer to manage GitOps applications throughout their entire lifecycle via a visual interface.

## TOC



## Use Cases

- Simplify the creation process of **GitOps Applications** by selecting the associated repository via the web console when creating them.
- When creating an **Application** via the native Argo CD dashboard, you can choose to use the associated repository.

## Prerequisites

- Installing Alauda Container Platform GitOps, and the **Native Argo CD UI** switch has been enabled.
- Access to the **Native Argo CD UI** URL along with username and password.
  - The Administrators can directly access the URL through the GitOps cluster plugin details page.

# Procedure

Follow these steps to utilize the features:

1. Connect Code Repository

   - Log in to **Argo CD** using the access URL.
   - Click on **Settings** in the left navigation bar.
   - Click on the **REPO** card.
   - Click **CONNECT REPO** in the upper left corner of the page.
   - Choose the method to connect the repository and fill in the corresponding parameters as needed.
   - Click **CONNECT**.

2. Associate Project

   - Click on **Settings** in the left navigation bar.
   - Click on the **Projects** card.
   - Click on the project where you need to create the GitOps application.

   Note: Argo CD will automatically sync projects in the cluster, so there is no need to create them manually.

   - Click **EDIT** in the **SOURCE REPOSITORIES** section.

- Click **ADD SOURCE**, enter the repository URL from the **Connect Repository** step, and associate it with the project.
- Click **SAVE**.

# Operation Result

- Return to the web console and navigate to **Container Platform** > **GitOps Applications**. On the **Create** page, you will see the associated repositories.

# Creating an Argo CD Application via Argo CD dashboard

## TOC



## Prerequisites

- Install (Choose one method):
  - Installing Alauda Container Platform GitOps
  - Installing Alauda Build of Argo CD
- Access credentials (URL, username, password) for the Argo CD dashboard have been obtained How to Obtain Argo CD Access Information

## Procedure

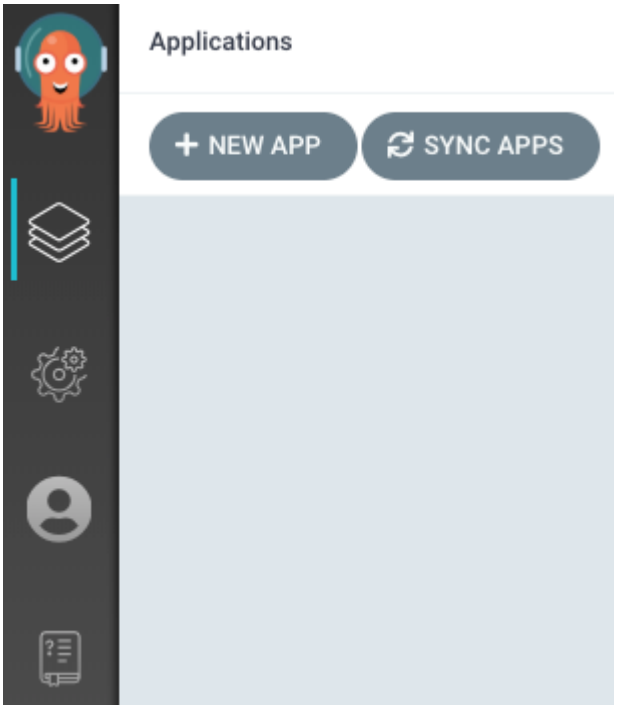
> Follow these steps to utilize the features:

1. Enter the Argo CD dashboard access URL in your browser to open the interface.

> The Administrators can directly access the **Argo CD Native UI** through the `global` cluster plugin details: locate the GitOps cluster plugin and click the access address.

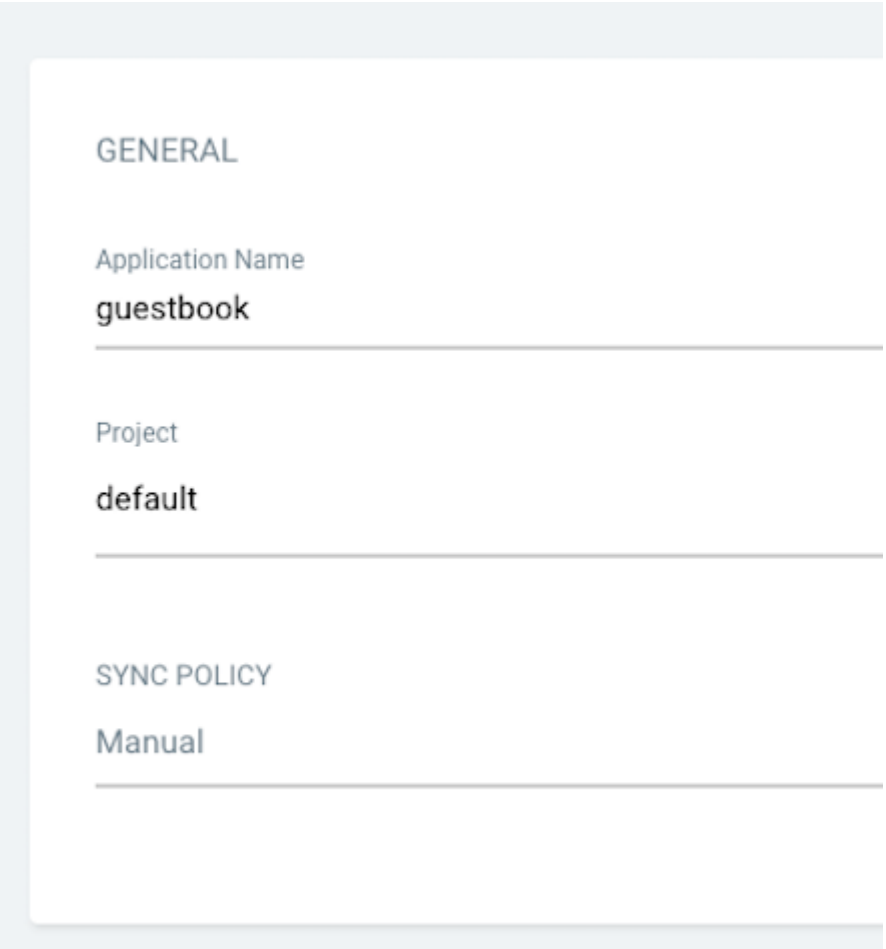
2. Authenticate with your Argo CD credentials and login.

3. Click the **+ NEW APP** button as shown below:



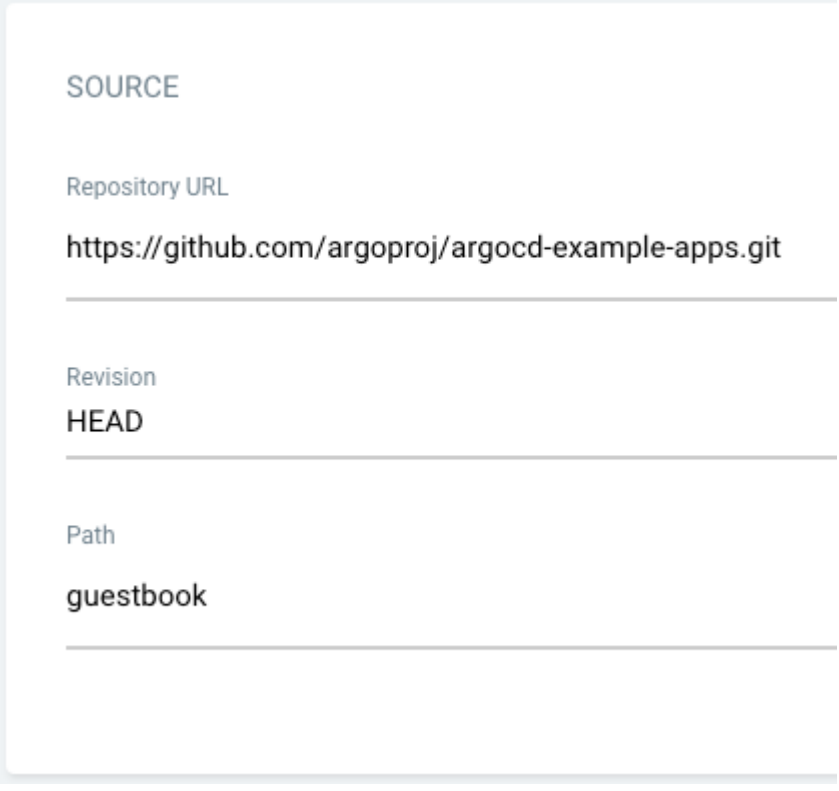

> Configure the application according to the foållowing steps:

**Basic Info Configuration**



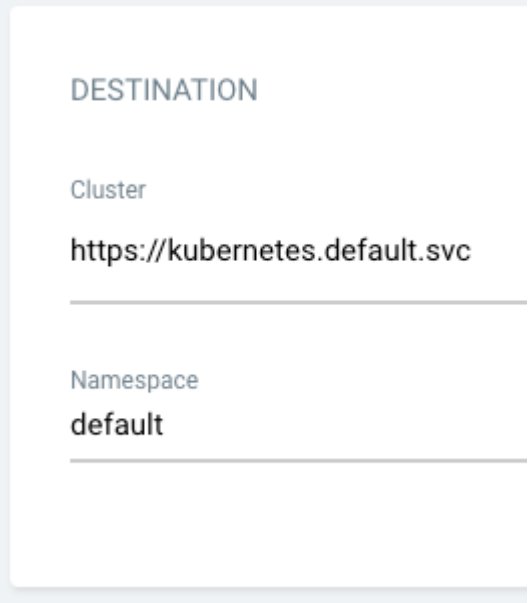

- **Application Name**: Input `guestbook`
- **Project**: Select `default`
- **Sync Policy**: Maintain `Manual` (recommended for initial configuration)

### Source Repo Configuration



- **Repository URL**: Set to `https://github.com/argoproj/argocd-example-apps.git`
- **Revision**: Use default `HEAD`
- **Path**: Specify `guestbook` (directory containing Kubernetes manifests)

**Destination Cluster Configuration**




- **Clusternew**: Set to `https://kubernetes.default.svc` (in-cluster access) or choose a specific cluster name
- **Namespace**: Set to `default` (or specify a target namespace)

4. Create **Application** After completing configurations, click the **Create** button at the top-right corner to initialize the creation of the guestbook application.

# Creating an Argo CD Application via the web console

This article will introduce the complete process of creating an Argo CD Application through the web console's **GitOps Applications**, allowing for GitOps management of the application.

## TOC



## Use Cases

- Create a SpringBoot Argo CD Application using the web console to experience the complete process of managing applications through GitOps.

## Prerequisites

- Installing Alauda Container Platform GitOps
- Projects and namespaces have been allocated

# Procedure

Follow these steps to utilize the features:

## Code Repository Configuration

> If you do not see Integrated Code Repository in the **Create GitOps Applications** details page, you can integrate the code repository first:

- Integrating Code Repositories via Argo CD dashboard

**INFO**

If you don't have an available code repository, you can use the demo repository for demonstration purposes. **Repository URL**: `https://github.com/argoproj/argocd-example-apps.git` **Description**: This repository contains example applications that can be used to demonstrate and test Argo CD functionalities.

## Create Argo CD Application by using GitOps Applications

1. **Container Platform**, click on **GitOps Applications**.
2. Click on **Create GitOps Application**.
3. In the **Basic Info** and **Code Repository** sections, configure the relevant information as per the instructions below.

| Parameter | Input Content |
| --- | --- |
| **Type** | `Application` |
| **Source** | `Argo CD Integration` |
| **Integrated Project Name** | `argocd-example-apps` |
| **Version Identifier** | `Branch`<br>`master` |

| Parameter | Input Content |
| --- | --- |
| **Source File Type** | `Helm` |
| **Source File Directory** | `helm-guestbook` |
| **Custom Values** | `values.yaml` |

4. In the **Distribution**, use the platform's recommended ***Namespace***, or select another namespace.

5. Set the synchronization policy to **Manually Sync** by default.

6. Click on **Create**.

# How to Obtain Argo CD Access Information

> This article details how to acquire access information for the Argo CD, covering both the **Alauda Container Platform GitOps** cluster plugin Argo CD installed on the web console and the one installed via the **Alauda Build of Argo CD** Operator.

## TOC



## Use Cases

- Once you've obtained the Argo CD access information, you can manage all native Argo CD resources via the Argo CD dashboard.

# How to Obtain Argo CD Access Information for the GitOps cluster plugin installed on the web console?

## Prerequisites

- Installing Alauda Container Platform GitOps
- (Option) The CLI plugin is installed, and the `web-cli` switch is enabled
- You possess Administrator permissions

## Procedure

**INFO**

It is advisable to enable the following settings when installing Alauda Container Platform GitOps cluster plugin:

- Enable the **Native Argo CD UI** switch.
- Enable the **Single Sign-On** switch.

> Follow these steps to utilize the features:

1. Login, and navigate to the **Administrator** page.
2. Click on **Marketplace** to access the **Cluster Plugins** list page.
3. Locate the **GitOps** plugin, click on **GitOps**, and a pop-up window will display the **GitOps Cluster Plugin** details.

If it's not enabled: Go back to the Cluster Plugins list page, find the GitOps plugin, click the Actions button, select Update, and enable the Argo CD Native UI switch. If it's enabled: Simply click the Access Address to open the Argo CD Dashboard.

4. Argo CD Native UI

- If not enabled: Navigate to the **Cluster Plugins** list page, find the **GitOps** plugin, click the **Update** button, and enable the **Argo CD Native UI** switch.
- If enabled: Click the **Access Address** directly to open the Argo CD dashboard.

5. **Single Sign-On**

- If enabled: Login to the Argo CD dashboard using the platform account.
- If not enabled: The account defaults to `admin`, and you need to retrieve the password by executing the following command in **Kubectl** Retrieve Argo CD Password.

# How to Obtain Argo CD Access Information from Argo CD Operator?

## Prerequisites

- Installing Argo CD
- (Option) The CLI plugin is installed, and the `web-cli` switch is enabled
- You possess Administrator permissions

## Procedure

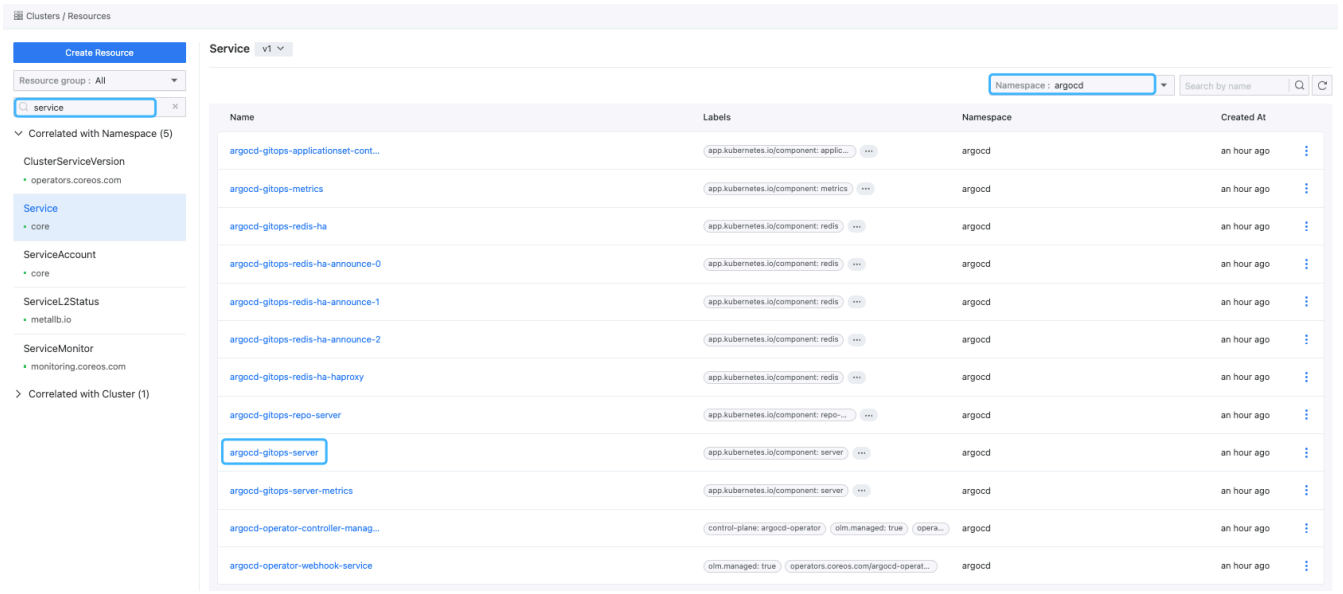
### Obtain Argo CD dashboard URL

1. Login, and navigate to the **Administrator** page.

2. Select **Cluster Management** to enter the **Resource Management** page.

3. In **Resource Group**, search for `Service`, select the ***argocd*** namespace (the namespace where the argocd instance is created). The default namespace for Argo CD installed on the web console is ***argocd***.

4. In the right **Resource List**, find the `argocd-gitops-server`, click the ***Actions*** button, and select **Update** to open the YAML details of argocd-gitops-server, details as shown in the image below.




5. Change the `type` to `NodePort` and record the `nodePort`, then click the **Update** button.

6. In the left sidebar, select **Cluster Management** to enter the **Cluster List** page.

7. Select the cluster where `argocd operator` is installed, enter the **Cluster Details** Page, and select **Nodes**.

8. Obtain the IP address of ***any control plane*** node.

9. Access Argo CD dashboard via `http://{control plane node IP}:{nodePort}`.

### Retrieve Argo CD Password

Execute the following command in **Kubectl** to retrieve the password:

```
kubectl get secret -n argocd argocd-gitops-cluster -o template --template='{{index .data "admin.password"}}'|base64 -d
```

## Update Argo CD admin account password

The default `admin` account password automatically created by installing Argo CD through **Alauda Container Platform GitOps** or **Alauda Build of Argo CD** operator cannot be modified via the **Argo CD dashboard** interface. You can change it by executing the following command in the CLI tool. Here, `newpassword` is the new password you wish to set.

```
kubectl patch -n argocd secrets argocd-gitops-cluster -p '{"stringData":
{"admin.password":"<newpassword>"}}'
```

# Troubleshooting

## TOC



## I've deleted/corrupted my repo and can't delete my app?

Argo CD can't delete an app if it cannot generate manifests. You need to either:

1. Reinstate/fix your repo.
2. Delete the app using `--cascade=false` and then manually deleting the resources.

## Why is my application still `OutOfSync` immediately after a successful Sync?

See Diffing Documentation ↗ for reasons resources can be OutOfSync, and ways to configure Argo CD to ignore fields when differences are expected.

## Why is my application stuck in `Progressing` state?

Argo CD provides health for several standard Kubernetes types. The `Ingress`, `StatefulSet` and `SealedSecret` types have known issues which might cause health check to return `Progressing` state instead of `Healthy`.

- `Ingress` is considered healthy if status.loadBalancer.ingress list is non-empty, with at least one value for `hostname` or `IP`. Some ingress controllers (contour , traefik) don't update `status.loadBalancer.ingress` field which causes `Ingress` to stuck in `Progressing` state forever.
- `StatefulSet` is considered healthy if value of `status.updatedReplicas` field matches to `spec.replicas` field. Due to Kubernetes bug kubernetes#68573 ↗ the status.updatedReplicas is not populated. So unless you run Kubernetes version which include the fix kubernetes#67570 ↗ StatefulSet might stay in Progressing state.
- Your `StatefulSet` or `DaemonSet` is using `OnDelete` instead of `RollingUpdate` strategy.
- For `SealedSecret`, see Why are resources of type `SealedSecret` stuck in the `Progressing` state?

As workaround Argo CD allows providing health check ↗ customization which overrides default behavior.

If you are using Traefik for your Ingress, you can update the Traefik config to publish the loadBalancer IP using publishedservice ↗, which will resolve this issue.

```
providers:
  kubernetesIngress:
    publishedService:
      enabled: true
```

## How to disable admin user?

Add `admin.enabled: "false"` in the `argocd-cm` ConfigMap.

## Argo CD cannot deploy Helm Chart based applications without internet access, how can I solve it?

Argo CD might fail to generate Helm chart manifests if the chart has dependencies located in external repositories. To solve the problem you need to make sure that `requirements.yaml` uses only internally available Helm repositories. Even if the chart uses only dependencies from internal repos Helm might decide to refresh `stable` repo. As workaround override `stable` repo URL in `argocd-cm` config map:

```
data:
  repositories: |
    - type: helm
      url: http://<internal-helm-repo-host>:8080
      name: stable
```

## After creating my Helm application with Argo CD I cannot see it with helm ls and other Helm

## commands?

When deploying a Helm application Argo CD is using Helm only as a template mechanism. It runs `helm template` and then deploys the resulting manifests on the cluster instead of doing `helm install`. This means that you cannot use any Helm command to view/verify the application. It is fully managed by Argo CD. Note that Argo CD supports natively some capabilities that you might miss in Helm (such as the history and rollback commands).

This decision was made so that Argo CD is neutral to all manifest generators.

## I've configured cluster secret but it does not show up in CLI/UI, how do I fix it?

Check if the cluster secret has the label `argocd.argoproj.io/secret-type: cluster`. If the secret has the label but the cluster is still not visible, it may be a permission issue. Try listing the clusters using the `admin` user (e.g.: `argocd login --username admin && argocd cluster list`).

Check if cluster secret has `argocd.argoproj.io/secret-type: cluster` label. If secret has the label but the cluster is still not visible then make sure it might be a permission issue. Try to list clusters using `admin` user (e.g. `argocd login --username admin && argocd cluster list`).

## Why Is My App Out Of Sync Even After Syncing?

In some cases, the tool you use may conflict with Argo CD by adding the `app.kubernetes.io/instance` label. E.g. using Kustomize common labels feature.

Argo CD automatically sets the `app.kubernetes.io/instance` label and uses it to determine which resources form the app. If the tool does this too, this causes confusion. You can change this label by setting the application.instanceLabelKey value in the `argocd-cm`. We recommend that you use `argocd.argoproj.io/instance`.

**INFO**

When you make this change your applications will become out of sync and will need re-syncing.

## How often does Argo CD check for changes to my Git or Helm repository?

The default polling interval is 3 minutes (180 seconds) with a configurable jitter. You can change the setting by updating the `timeout.reconciliation` value and the `timeout.reconciliation.jitter` in the `argocd-cm` config map. If there are any Git changes, Argo CD will only update applications with the `auto-sync` setting enabled. If you set it to 0 then Argo CD will stop polling Git repositories automatically and you can only use alternative methods such as `webhooks` and/or `manual syncs` for creating applications.

## How Do I Fix invalid cookie, longer than max length 4093?

Argo CD uses a JWT as the auth token. You likely are part of many groups and have gone over the 4KB limit which is set for cookies. You can get the list of groups by opening "developer tools -> network":

1. Click login to the Argo CD dashboard How to Obtain Argo CD Access Information
2. Find the call to `<argocd_instance>/auth/callback?code=<random_string>`

Decode the token at jwt.io↗. That will provide the list of teams that you can remove yourself from.

## Why Am I Getting rpc error: code = Unavailable desc = transport is closing When Using The CLI?

Maybe you're behind a proxy that does not support HTTP 2? Try the `--grpc-web` flag:

```
argocd ... --grpc-web
```

# Why are resources of type SealedSecret stuck in the Progressing state?

The controller of the `SealedSecret` resource may expose the status condition on resource it provisioned. Since version `v2.0.0` Argo CD picks up that status condition to derive a health status for the `SealedSecret`.

Versions before `v0.15.0` of the `SealedSecret` controller are affected by an issue regarding this status conditions updates, which is why this feature is disabled by default in these versions. Status condition updates may be enabled by starting the `SealedSecret` controller with the `--update-status` command line parameter or by setting the `SEALED_SECRETS_UPDATE_STATUS` environment variable.

To disable Argo CD from checking the status condition on `SealedSecret` resources, add the following resource customization in your `argocd-cm` ConfigMap via `resource.customizations.health.<group_kind>` key.

```
resource.customizations.health.bitnami.com_SealedSecret: |
  hs = {}
  hs.status = "Healthy"
  hs.message = "Controller doesn't report resource status"
  return hs
```

# How to rotate Redis keys?

- Delete `argocd-redis` secret in the namespace where Argo CD is installed.

  ```
  kubectl delete secret argocd-redis -n <argocd namesapce>
  ```

- If you are running Redis in HA mode, restart Redis in HA.

```
kubectl rollout restart deployment argocd-redis-ha-haproxy
kubectl rollout restart statefulset argocd-redis-ha-server
```

- If you are running Redis in non-HA mode, restart Redis.

```
kubectl rollout restart deployment argocd-redis
```

- Restart other components.

```
kubectl rollout restart deployment argocd-server argocd-repo-server
kubectl rollout restart statefulset argocd-application-controller
```

## How do I fix Manifest generation error (cached)?

`Manifest generation error (cached)` means that there was an error when generating manifests and that the error message has been cached to avoid runaway retries.

Doing a hard refresh (ignoring the cached error) can overcome transient issues. But if there's an ongoing reason manifest generation is failing, a hard refresh will not help.

Instead, try searching the repo-server logs for the app name in order to identify the error that is causing manifest generation to fail.