# Backup and Restore

## Overview

### Overview

Cluster backup and restore

Application backup and restore

Backup considerations

Usage scenarios

## etcd Backup and Restore

### etcd Backup and Restore

Prerequisites

How it works

Configuration Reference

Viewing Backup Records

S3 Backup Configuration

etcd Restore

Configuration Management

# Application Backup and Restore

## Backup repository

- Procedure
- Relevant operations

## Create an application backup s

- Prerequisites
- Procedure
- Relevant operations

## Run an App

- Prerequisites
- Procedure
- What's next

pera

## Image Registry Replacement

- Procedure

## Hooks

- Overview
- Backup Hook
- Restore Hooks

# Overview

Alauda Container Platform provides two types of backup and restore operations:

- **Cluster backup and restore**: Backs up and restores control plane data, including etcd, registry, logging, and monitoring data.
- **Application backup and restore**: Backs up and restores applications and their persistent volumes based on Velero.

## TOC

Cluster backup and restore
etcd backup and restore
Registry backup and restore
Logging backup and restore
Monitoring backup and restore
Application backup and restore
Architecture
Application installation
Backing up and restoring applications
Backup considerations
Immutable OS clusters
Usage scenarios
Backup and restore within the current cluster
Cross-cluster application migration

# Cluster backup and restore

Cluster backup protects the control plane state and platform data.

## etcd backup and restore

etcd is the key-value store for ACP and persists the state of resource objects. etcd backups are required for control-plane state recovery scenarios.

For detailed instructions, see etcd Backup and Restore.

## Registry backup and restore

For registry backup and restore, see Registry Data Backup and Recovery.

## Logging backup and restore

Currently supports ClickHouse. Contact technical support.

## Monitoring backup and restore

For monitoring backup and restore, see VictoriaMetrics Backup and Recovery.

# Application backup and restore

As a cluster administrator, you can back up and restore applications running on ACP by using Velero.

## Architecture

Application backup and restore consists of two components:

- **Alauda Container Platform Data Backup Essentials**: Provides the UI and is installed on the global cluster.
- **Alauda Container Platform Data Backup for Velero**: Provides Velero and is installed on workload clusters.

## Application installation

To enable application backup and restore:

1. Download **Alauda Container Platform Data Backup Essentials** and **Alauda Container Platform Data Backup for Velero** from the Alauda Customer Portal.
2. Upload the packages to the platform.
3. Install Data Backup Essentials on the global cluster.
4. Install Data Backup for Velero on workload clusters.

After installation, configure a backup repository to store backup data.

## Backing up and restoring applications

You back up applications by creating backup schedules and restore applications by executing recovery tasks.

For detailed instructions, see:

- Create application backups
- Restore applications

# Backup considerations

Before configuring backup policies, consider the following architectural factors that affect backup and recovery strategies.

## Immutable OS clusters

For clusters running on Immutable OS, **VM Snapshot is not required or recommended** as a backup strategy for disaster recovery scenarios.

**Why VM Snapshot is not recommended**:

- **Immutable OS design**: The operating system layer is read-only and managed centrally by the platform. When a node fails, the platform automatically creates a new node with the correct configuration.
- **Distributed system nature**: Kubernetes is a distributed system. VM Snapshot cannot capture the consistent state of distributed components like etcd quorum.
- **Disaster recovery limitations**: VM Snapshot is typically stored with original data and cannot handle site-level disasters.

**Recommended backup approach**:

- **Cluster state**: Use etcd backups to capture control plane state
- **Application data**: Use PV snapshots or Restic for persistent volumes
- **Cluster configuration**: Use GitOps/IaC for configuration management

# Usage scenarios

## Backup and restore within the current cluster

Restore applications to the current cluster after accidental deletion or failure. Application resources typically do not require modification during recovery.

## Cross-cluster application migration

Common scenarios include:

- Development and testing across data centers
- Resource migration between clusters
- Replication from production to development/testing clusters

**Considerations**:

- Ensure similar CPU and memory specifications between source and target clusters
- Maintain consistent network modes to avoid resource restoration issues
- If subnets differ, pod IPs will change after recovery
- Assess whether image migration is required before restoring data

# etcd Backup and Restore

The etcd service on the cluster is a distributed key-value store responsible for storing cluster configuration information. etcd is deployed on all control plane nodes of the cluster.

After installing the Alauda Container Platform Cluster Enhancer plugin, an `EtcdBackupConfiguration` resource is automatically created for the cluster configuration. The `EtcdBackupConfiguration` contains backup data sources, control plane node paths, backup data storage locations, and backup methods. Each backup execution based on the policy generates a new backup record, enabling on-demand and scheduled cluster-configuration backups.

## TOC

- Prerequisites
- How it works
- Configuration Reference
  - Schedule and Retention
- Viewing Backup Records
  - Using the Platform UI
  - Using the CLI
- S3 Backup Configuration
  - Prerequisites
  - Step 1: Create S3 Secret
  - Step 2: Configure EtcdBackupConfiguration
  - Step 3: Verify Backup

etcd Restore
Prerequisites
Step 1: Backup Original Data and Modify etcd Configuration
Step 2: Copy Backup Snapshot
Step 3: Restore etcd
Step 4: Distribute Restored Data
Step 5: Restart Cluster Components
Step 6: Verify Recovery
Configuration Management

---

# Prerequisites

To enable etcd backup:

1. Download **Alauda Container Platform Cluster Enhancer** from the Alauda Customer Portal.
2. Upload the package to the platform.
3. Install the plugin on your cluster.

After installation, an EtcdBackupConfiguration resource is automatically created.

# How it works

- etcd backup is provided by **Alauda Container Platform Cluster Enhancer**.
- Supports both local storage and S3-compatible object storage. Local backups are written to a directory on each control plane node — `/var/cpaas/backup` by default. Plan this directory with enough capacity for your etcd data size and retention period. Configuring S3 storage creates an additional copy in the S3 bucket; local backups continue to be generated.
- For Immutable OS clusters, local backup uses `/var/cpaas/backup` by default. S3 storage remains supported through `remoteStorage.s3`.

# Configuration Reference

You can configure the `EtcdBackupConfiguration` resource to customize backup schedules, retention policies, and storage options.

## Schedule and Retention

- **`schedule`**: Defines the backup frequency using standard cron syntax.
  - Example: `0 0 * * *` (Run backup daily at midnight).
- **`localStorage`**: Configures local backup storage.
  - **`path`**: The directory on each control plane node where backups are stored. Default is `/var/cpaas/backup`. Set this to a directory with enough capacity for the etcd data size and the configured retention period; see the storage note below.
  - **`ttl`**: The retention period for backup files in seconds. Backups older than this duration will be automatically deleted.
    - Example: `7776000` (approximately 90 days).
- **`paused`**: Set to `true` to temporarily suspend automatic backups without deleting the configuration.

> **WARNING**
>
> Plan storage capacity for local backups before enabling etcd backup. Local backups accumulate on the control plane nodes and are kept until their `ttl` expires, so `/var/cpaas/backup` must have enough free space for the expected backup size and retention period.

Example configuration:

```yaml
apiVersion: enhancement.cluster.alauda.io/v1
kind: EtcdBackupConfiguration
metadata:
  name: etcd-backup-default
spec:
  schedule: "0 0 * * *"       # Run daily at 00:00
  paused: false               # Enable backups
  localStorage:
    path: /var/cpaas/backup   # Local backup directory on each control pl
ane node
    ttl: "7776000"            # Retain for 90 days
  # ... other fields
```

# Viewing Backup Records

To view etcd backup records, you can use the platform UI or the command line.

## Using the Platform UI

1. In the left navigation bar, click **Operation Center** > **Monitor** > **Dashboards**.
2. Click **Switch** in the upper right corner of the page.
3. Click **Cluster** → **etcd backup** to view the etcd backup records.

## Using the CLI

You can verify backup status and history by checking the `status` field of the `EtcdBackupConfiguration` resource:

```bash
kubectl get etcdbackupconfiguration etcd-backup-default -o yaml
```

The output contains a `status.records` list with details for each backup, including:

- `backupTimestamp`: Time the backup was created.
- `fileName`: Name of the backup file (e.g., `snapshot-etcd-<date>-<time>-<ip>.tar`).

- `result`: Outcome of the backup operation (e.g., `Success`).

# S3 Backup Configuration

To enable S3 storage for etcd backups, follow these steps:

## Prerequisites

- Alauda Container Platform Cluster Enhancer is installed on the cluster.

## Step 1: Create S3 Secret

Prepare your S3 access credentials and create a Kubernetes secret in the `cpaas-system` namespace:

```
export ACCESS_KEY="your-access-key"
export SECRET_KEY="your-secret-key"

kubectl create secret generic etcd-backup-s3-secret \
  --from-literal=ACCESS_KEY="$ACCESS_KEY" \
  --from-literal=SECRET_KEY="$SECRET_KEY" \
  --dry-run=client -n cpaas-system -o yaml | kubectl apply -f -
```

## Step 2: Configure EtcdBackupConfiguration

Modify the `EtcdBackupConfiguration` resource to add the `remoteStorage` field with S3 configuration:

```
spec:
  remoteStorage:
    s3:
      endpoint: "your-s3-endpoint"  # e.g.: https://s3.bucket.com
      region: "your-s3-region"
      bucket: "your-s3-bucket"
      dir: "your-s3-bucket-dir"
      skipTLSVerify: false  # Set to true only for self-signed certificat
es
      secretRef: etcd-backup-s3-secret
```

## Step 3: Verify Backup

Trigger a manual etcd backup to verify the configuration:

```
# Set environment variables
token="your-platform-token"
platform_url="your-platform-url"
cluster_name="your-cluster-name"

# Trigger backup
curl $platform_url/kubernetes/$cluster_name/apis/enhancement.cluster.alau
da.io/v1/etcdbackupconfigurations/etcd-backup-default/exec \
  -k -H "Authorization: Bearer $token"
```

After the backup completes, verify that backup files exist in your S3 bucket.

For restore preparation, use the `fileName` shown in the backup record and the configured `remoteStorage.s3.dir` value to download the tar archive from S3:

```
aws --endpoint-url <s3-endpoint> s3 cp \
  s3://<bucket>/<dir>/<backup-file-name> \
  /tmp/snapshot-etcd.tar

mkdir -p /tmp/etcd-restore
tar -xf /tmp/snapshot-etcd.tar -C /tmp/etcd-restore
find /tmp/etcd-restore -name 'snapshot-etcd-*.db' -type f
```

# etcd Restore

> **Warning:**
>
> - This operation performs a destructive recovery of the etcd cluster. It will overwrite the existing data. Ensure you have a valid backup snapshot before proceeding.
> - This procedure entails significant risks. If you are unsure about the operation, contact technical support.
> - During the recovery process, the Kubernetes API Server will be unavailable.

## Prerequisites

- The Kubernetes cluster is deployed using hostnames (`kubectl get node` shows hostnames as node names).
- An etcd backup snapshot is available.
- The cluster is malfunctioning due to the failure of etcd nodes (e.g., more than half of the control plane nodes are down).
- This recovery procedure is specifically designed for a **3-node control plane** cluster. If your cluster has 5 or more control plane nodes, contact technical support for assistance.
- On Immutable OS, confirm that `/var/lib/etcd`, `/etc/kubernetes/manifests/etcd.yaml`, and `/etc/kubernetes/pki/etcd/` exist on each control plane node before starting restore.
- On Immutable OS, `etcdctl` might not be available in the host `PATH`. Locate it under `/var/lib/containerd`, or use a container image that includes a compatible `etcdctl` binary.

## Step 1: Backup Original Data and Modify etcd Configuration

Execute the following commands on **all control plane nodes**:

```
# Create backup directory
mkdir -p /root/backup_$(date +%Y%m%d%H)/old-etcd/

# Stop kubelet
systemctl stop kubelet

# Prepare etcdctl binary
ETCDCTL_PATH="$(find /var/lib/containerd/ -name etcdctl -type f | tail -
1)"
if [ -z "$ETCDCTL_PATH" ]; then
  echo "etcdctl was not found under /var/lib/containerd. Use a container
image that includes a compatible etcdctl binary."
  exit 1
fi
cp "$ETCDCTL_PATH" /root/etcdctl
chmod +x /root/etcdctl

# Remove etcd containers
crictl ps -a | grep etcd | awk '{print $1}' | xargs -r crictl rm -f

# Backup etcd data and kubernetes configuration
cp -a /var/lib/etcd/* /root/backup_$(date +%Y%m%d%H)/old-etcd/
rm -rf /var/lib/etcd/*
cp -r /etc/kubernetes/ /root/backup_$(date +%Y%m%d%H)/old-etcd/

# Modify etcd.yaml to use existing cluster state
sed -i /initial-cluster-state=/d /etc/kubernetes/manifests/etcd.yaml
sed -i '/initial-cluster=/a\    - --initial-cluster-state=existing' /etc/
kubernetes/manifests/etcd.yaml
```

**Note**: Verify the indentation of `--initial-cluster-state=existing` in `/etc/kubernetes/manifests/etcd.yaml`.

## Step 2: Copy Backup Snapshot

Copy the selected etcd backup snapshot to the `/tmp` directory on the **first control plane node** and name it `snapshot.db`.

If the backup tar archive is available locally, extract it from `/var/cpaas/backup`:

```
mkdir -p /tmp/etcd-restore
tar -xf "$(ls -1t /var/cpaas/backup/snapshot-etcd-*.tar | head -1)" -C /t
mp/etcd-restore
cp "$(find /tmp/etcd-restore -name 'snapshot-etcd-*.db' -type f | head -
1)" /tmp/snapshot.db
```

If the backup tar archive is stored in S3, download it first, and then extract the snapshot file:

```
aws --endpoint-url <s3-endpoint> s3 cp \
  s3://<bucket>/<dir>/<backup-file-name> \
  /tmp/snapshot-etcd.tar

mkdir -p /tmp/etcd-restore
tar -xf /tmp/snapshot-etcd.tar -C /tmp/etcd-restore
cp "$(find /tmp/etcd-restore -name 'snapshot-etcd-*.db' -type f | head -
1)" /tmp/snapshot.db
```

## Step 3: Restore etcd

Execute the following script on the **first control plane node** to restore the snapshot.

> **Note:** The following script assumes a 3-node control plane cluster. If your cluster has 5 or more nodes, contact technical support.

```
#!/usr/bin/env bash

# Set etcd node IPs (Replace with actual IPs)
export ETCD_1=1.1.1.1
export ETCD_2=2.2.2.2
export ETCD_3=3.3.3.3

# Set corresponding node hostnames (Replace with actual hostnames)
export ETCD_1_HOSTNAME=etcd-1
export ETCD_2_HOSTNAME=etcd-2
export ETCD_3_HOSTNAME=etcd-3

export ETCDCTL_API=3

# Loop for 3 nodes. Adjust '1 2 3' if you have a different number of node
s.
for n in 1 2 3; do
  ip_var=ETCD_${n}
  host_var=ETCD_${n}_HOSTNAME

  ip=${!ip_var}
  host=${!host_var}

  echo "Restoring for node: ${host} (${ip})..."

  rm -rf /tmp/etcd
  /root/etcdctl snapshot restore /tmp/snapshot.db \
    --cert=/etc/kubernetes/pki/etcd/server.crt \
    --key=/etc/kubernetes/pki/etcd/server.key \
    --cacert=/etc/kubernetes/pki/etcd/ca.crt \
    --skip-hash-check=true \
    --data-dir=/tmp/etcd \
    --name "${host}" \
    --initial-cluster \
      ${ETCD_1_HOSTNAME}=https://${ETCD_1}:2380,\
${ETCD_2_HOSTNAME}=https://${ETCD_2}:2380,\
${ETCD_3_HOSTNAME}=https://${ETCD_3}:2380 \
    --initial-advertise-peer-urls https://"${ip}":2380 && \
    mv /tmp/etcd /root/etcd_"${host}"

  echo "Restoration for ${host} completed. Data directory: /root/etcd_${h
ost}"
done
```

After the script completes, three directories ( `etcd_$host` ) are generated in the `/root` directory.

## Step 4: Distribute Restored Data

1. Transfer the restored data directories to the corresponding control plane nodes. Use `scp` or a similar tool to copy the directories generated in Step 3 ( `/root/etcd_<hostname>` ) from the first node to the others.
   For example, transfer to **etcd-2** and **etcd-3**:

   ```
   # Replace <etcd-2-ip> and <etcd-3-ip> with actual IPs
   scp -r /root/etcd_etcd-2 root@<etcd-2-ip>:/root/
   scp -r /root/etcd_etcd-3 root@<etcd-3-ip>:/root/
   ```

2. Restore the data to the etcd data directory ( `/var/lib/etcd` ) on **each control plane node**.

   ```
   # On etcd-1:
   cp -r /root/etcd_etcd-1/member/* /var/lib/etcd/

   # On etcd-2:
   cp -r /root/etcd_etcd-2/member/* /var/lib/etcd/

   # On etcd-3:
   cp -r /root/etcd_etcd-3/member/* /var/lib/etcd/
   ```

## Step 5: Restart Cluster Components

Execute the following commands on **all control plane nodes**:

```
# Remove Kubernetes control plane containers
crictl ps -a | grep -E "kube-api|kube-sche|kube-contro" | awk '{print
$1}' | xargs -r crictl rm -f

# Restart kubelet
systemctl restart kubelet
```

### Step 6: Verify Recovery

1. Check if the etcd cluster is healthy. You can execute this command inside any `etcd` pod or using the `etcdctl` binary on the host:

```
# Using etcdctl on the host
export ETCDCTL_API=3
/root/etcdctl --cacert=/etc/kubernetes/pki/etcd/ca.crt \
  --cert=/etc/kubernetes/pki/etcd/server.crt \
  --key=/etc/kubernetes/pki/etcd/server.key \
  --endpoints=https://127.0.0.1:2379 \
  endpoint health
```

2. Check if the Kubernetes pods are running correctly:

```
kubectl get po -n kube-system
```

3. Restart kubelet on **all nodes** (both control plane and worker nodes) to ensure all components reconnect to the recovered etcd:

```
systemctl restart kubelet
```

## Configuration Management

To modify the default etcd backup configuration, contact technical support for detailed configuration options and advanced settings.

# Application Backup and Restore

Backup repository

Create an application backup s

Run an App

Image Registry Replacement

Hooks

# Backup repository

The backup repository stores application and configuration data (for example, namespaces and applications) and enables rapid restoration from backup files.

## TOC

Procedure

Relevant operations

Delete a backup repository

## Procedure

1. In the left navigation, click **Clusters** > **Backup and Recovery**.
2. Switch to the **Backup Repository Management** tab.
3. Click **Create Backup Repository** and configure the parameters as follows.

| Parameter | Description |
| --- | --- |
| **Region** | The region of the object storage data center. **Tip**: For MinIO, enter "minio". |
| **Addressing Style** | Addressing method. Defaults: virtual-hosted–style for object storage; path-based for file storage and MinIO integration. |

| Parameter | Description |
| --- | --- |
| **Endpoint** | The external access address (HTTP REST API) of the object storage. Use the service endpoint, for example `https://cos.<region>.myqcloud.com`. **Caution**: Do not use the public bucket domain (which includes the bucket name), for example `https://<bucket name>.cos.<region>.myqcloud.com`. |
| **Bucket** | Name of the storage bucket. |
| **Folder** | Bucket folder prefix used to store backup data, for example `backups`. |
| **Secret Id/Secret Key** | Access keys for authenticating to the object storage. |
| **Skip TLS Verify** | If the external storage does not use a valid certificate, enable this option to skip certificate verification. |
| **CA Certificate** | If CA verification is required, upload the Base64-encoded CA certificate. |

4. Click **Create**.

# Relevant operations

## Delete a backup repository

If a backup schedule references the repository, deleting it will cause backup failures when the schedule runs. Update the backup schedule to reference another repository before deletion.

1. In the left navigation, click **Clusters** > **Backup and Recovery**.
2. Switch to the **Backup Repository Management** tab.
3. Click the **Delete** button on the right of the backup repository, and then click **Delete** Confirm the deletion.

# Create Application Backup

Create an application backup schedule to define the scope of data to back up (by namespace), the backup storage location, method, and related parameters. Each run based on the schedule generates a new backup record, enabling automatic backups for application resources in selected namespaces on demand or at a defined frequency.

## TOC

- Prerequisites
- Procedure
  - Basic information
  - Backup resources
  - Method
  - Advanced configuration
- Relevant operations
  - Run the backup schedule manually
  - Export backup task logs

## Prerequisites

- The backup components are installed. For installation instructions, see application installation.

- The backup repository is configured.

# Procedure

> **WARNING**
>
> - To back up application data, include PersistentVolumeClaims (PVCs). PVCs bound to `hostPath` PersistentVolumes are not supported.
> - To ensure reliability and data integrity, do not back up database data (for example, MySQL-PXC, Redis). Use Data Services for database backups.
> - Avoid reads, writes, updates, and deletes in the namespaces being backed up to prevent drift and inconsistencies after migration.

## 1 Basic information

1. In the left navigation bar, click **Clusters** > **Backup and Recovery**.
2. Switch to the **Backup Management** tab.
3. Click **Create Backup Policy** > **Create Application Backup** and configure the parameters as follows.
   - **Backup Resource Type**: **Kubernetes Resources** includes all Kubernetes resource files in the selected namespaces. **PVCs** are persistent volume claims used to back up application data bound to persistent volumes. PVCs bound to `hostPath` volumes are not supported.
     - If the storage resource used by your PVC has a **Recycle Strategy** of **Retain**, only Kubernetes resources need to be backed up.
     - If the storage resource used by your PVC has a **Recycle Strategy** of **Delete**, both Kubernetes resources and PVCs need to be backed up.
   - **Backup Repository**: Select a repository that has passed connectivity verification, or click Create Backup Repository. After creating a repository, click **OK and Create Application Backup** to return and continue.

4. After configuring the basic information, click **Next**.

## 2 Backup resources

Back up application resources under the selected namespaces.

Namespaces that have not been imported in the cluster are not displayed. To back up such namespaces, import them into the project first.

1. Select one or more **Namespaces** to be backed up.
2. **(Optional)** Configure resource filtering options to refine the backup scope:

   - **Included Namespace Resources**: Select specific namespace-scoped resource types to include in the backup. Only the selected resource types within the chosen namespaces will be backed up. This field supports fuzzy search and multi-selection.
   - **Included Cluster Resources**: Select specific cluster-scoped resource types to include in the backup. Only explicitly selected cluster-scoped resources will be backed up. By default, no cluster-scoped resources are backed up unless included. This field supports fuzzy search and multi-selection.
   - **Excluded Namespace Resources**: Select namespace-scoped resource types to exclude from the backup. Velero will not back up the selected resource types. This field supports fuzzy search and multi-selection.
   - **Label Selectors**: Add label selectors to filter resources. Only resources matching the configured expressions will be backed up. When multiple selectors are configured, they are combined with OR logic (backed by Velero's `orLabelSelectors` field). Note: this is mutually exclusive with using a single `labelSelector` — only one mode can be used per backup. Each selector supports two matching methods:
     **matchLabels** - Simple key-value matching:

     | Parameter | Type | Description |
     | --- | --- | --- |
     | Key | string (required) | Label key |
     | Value | string (required) | Label value |

**matchExpressions** - Complex conditional matching:

| Parameter | Type | Description |
| --- | --- | --- |
| Key | string (required) | Label key |
| Operator | string (required) | One of `In`, `NotIn`, `Exists`, `DoesNotExist` |
| Values | array (conditional) | Array of values. Required for `In` / `NotIn`, must be empty for `Exists` / `DoesNotExist` |

To match resources by label key only (without specifying a value), use **matchExpressions** with the `Exists` operator.

> **NOTE**
>
> If a matched resource has mounted PVCs and the backup mode includes PVC data (based on `defaultVolumesToFsBackup`), the mounted PVC data will also be backed up unless excluded by Pod annotations (for example, `backup.velero.io/backup-volumes-excludes`). This annotation only excludes volumes from filesystem backup—if volume snapshots are configured, excluded volumes may still be backed up via snapshots.

3. Click **Next**.

## 3 Method

Configure the backup schedule.

- **Backup once only**: Executes immediately after creation. After setting the **Backup retention period**, backups exceeding the retention period are cleaned up automatically.
- **Scheduled**: Set a **Backup rule** to execute the policy periodically. A crontab expression is supported. You can use the platform's preset **Backup rule templates**, then adjust as needed. Recommended minimum frequencies: once per day for

**Backup Kubernetes resources and Persistent Volume Claims**; once per hour for **Backup Kubernetes resources**.

### 4 Advanced configuration

If required, configure custom hooks to run during the backup process.

# Relevant operations

## Run the backup schedule manually

Manually execute a created schedule (including those with periodic rules). Each execution generates a new backup record.

1. On the left navigation bar, click **Clusters** > **Backup and Recovery**.
2. Switch to the **Backup Management** tab.
3. Click **Execute Backup** on the right of the schedule, then confirm.

## Export backup task logs

Manually export the backup task log for a specified schedule. **Log export is not supported while the backup task is in progress.**

**Procedure**

1. On the left navigation bar, click **Clusters** > **Backup and Recovery**.
2. Switch to the **Backup Management** tab.
3. Click the **Backup Schedule Name** to view the backup record, then in the **Backup Records** area click **Export Log** on the right of the record.

# Run an Application Restore Task

You can quickly restore the application to the target namespace by performing an application recovery task based on existing application backup records in the following scenarios:

- Kubernetes resources were accidentally deleted and need to be restored.
- Application data needs to be migrated to other namespaces in the same cluster.
- Application resources need to be migrated to namespaces in other clusters on the platform.
- Kubernetes resources backed up in the production cluster need to be restored to the disaster recovery cluster.

## TOC

Prerequisites
Procedure
What's next
Relevant operations
Try again
Procedure
Download the application recovery task log
Procedure

# Prerequisites

- A successful application backup exists in the object storage where the cluster stores backup files.
- If the PersistentVolume has a **reclaimPolicy** of **Retain**, delete `spec.claimRef.uid` and `spec.claimRef.resourceVersion` on the corresponding PersistentVolume (PV) before restoring data to the bound PVC.
- For cross-cluster recovery (data migration), ensure that the target cluster and the cluster where the backup file resides can read the same backup file. This can be achieved in two ways:
  - The target cluster and the cluster where the backup file resides are connected to the same object storage.
  - Copy the required backup files to the object storage connected to the target cluster in advance.
- If **Backup Kubernetes resources and persistent volume claims** was selected as the resource type for the application backup, ensure that the target cluster **StorageClass** name matches that of the source cluster. If not, configure the mapping between the source and target storage classes in **Advanced Recovery Target Settings**.

# Procedure

1. In the left navigation bar, click **Clusters** > **Backup and Recovery**.
2. Switch to the **Recovery Management** tab.
3. Click **Execute Application Recovery Task**.
4. Configure the parameters as follows.

| Parameter | Description |
| --- | --- |
| **Backup Repository** | Select a repository that has passed connectivity verification, or click Create Backup Repository. **Tip**: After creating a repository, click **OK and Create Application Backup** to return and continue, or click **Create** to return to the repository list. |

| Parameter | Description |
| --- | --- |
| **Recovery File Configuration** | Select the backup file that stores the backup data. **Tip**: The file name prefix is the name of the backup policy; only successfully backed up files can be selected. |
| **Recovery Target Configuration** | **Namespaces**: The namespace where data recovery is performed and the source namespace of the backup data. The optional range is the namespace set in the backup policy. The system restores backup data to the same namespace based on your selection. **Tip**: To restore to other namespaces in the cluster, configure **Advanced Recovery Target Settings**. |
| **Advanced Recovery Target Settings** | Restore backup data originally scheduled for the source namespace to any namespace in the cluster (existing or newly created). **Source Namespace**: The selected namespace. **Target Namespace**: The namespace where data recovery is performed; either an existing namespace or a new one created by entering a non-existent name. **Tip**: If **Backup Kubernetes Resources and Persistent Volume Claims** was selected as the application backup resource type, ensure that the target cluster **StorageClass** name matches the source. If not, configure the source and target storage class names in the advanced options; the platform stores data using the new storage class. |

5. Click **YAML** in the upper-right corner to switch to YAML editing mode. To configure commands that run during recovery, see Configuring Hooks.
   **Caution**: By default, the backup file is compared with resources in the target namespace. Only data that exists in the backup file but is missing in the namespace is restored. Resources with the same name or incremental resources (existing in the namespace but missing in the backup file) are not overwritten.
   To overwrite resources with the same name that already exist in the namespace:

   1. Click **YAML** in the upper-right corner to switch to YAML editing mode.
   2. Add `existingResourcePolicy: update` under `.spec`, and add `pods` to `excludedResources` as follows: `excludedResources: ["pods"]`.

   **Tip**: This method cannot overwrite application data in persistent volumes bound to PVCs.

6. Click **Start**.

# What's next

1. **Namespace Import**: After cross-cluster or cross-platform migration, manually import the namespace into the corresponding project in **Project Management**. Otherwise, the recovered application may not be visible in the platform UI.
2. **Reconfigure Fixed IP**: After cross-cluster or cross-platform migration, fixed IP addresses of containers in compute components change. If necessary, manually update the **Static IP Address** parameter of the container group for deployments, daemon sets, and stateful sets.

# Relevant operations

## Try again

If the recovery task fails, retry the task.

> **TIP**
>
> Retrying will generate a new recovery record, and you can view the execution status of the task through the new recovery record.

### Procedure

1. In the left navigation bar, click **Clusters** > **Backup and Recovery**.
2. Switch to the **Recovery Management** tab.
3. Click **Retry** on the right of the failed recovery record, and confirm.

## Download the application recovery task log

Each time a recovery task is executed, a recovery record is generated. View the execution status and details (YAML) through the recovery record, and manually download the operation

log of the application recovery task.

## Procedure

1. In the left navigation bar, click **Clusters** > **Backup and Recovery**.
2. Switch to the **Recovery Management** tab.
3. Click **Export Log** on the right of the recovery record.

# Image Registry Replacement

For cross-cluster and cross-platform recovery scenarios, use this procedure to restore application images so they can pull from the correct repository after migration.

- **Image repository migration**: When the image repository changes, update application image references to the new repository.
- **Image synchronization**: When synchronizing images to other projects within the repository (for example, promoting from test to production), update application image references to the new version.

## TOC

Procedure

## Procedure

1. Before performing the recovery task, create a ConfigMap that maps old to new image repositories. Replace the placeholders in the following example and save as `change-registry-config.yaml`.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: change-registry-config
  namespace: cpaas-system
  labels:
    velero.io/plugin-config: ""
    alauda.io/change-registry: RestoreItemAction
data:
  old: <Old Image Repository Address>
  new: <New Image Repository Address>
```

2. On a control-plane node of the target cluster, create the ConfigMap:

```
kubectl apply -f change-registry-config.yaml
```

3. After creation, continue with the application recovery task.

# Hooks

## TOC

Overview

Backup Hook

Restore Hooks

## Overview

Custom hooks are extension commands that run in containers within the cluster. By configuring hooks, you can perform tailored operations during backup and restore. For special configuration requirements, contact technical support.

## Backup Hook

During backup execution, when a Pod is being backed up, one or more commands can be executed in containers within the Pod. These commands can run before ( `pre` ) or after ( `post` ) the Pod backup operation completes.

Hooks can be configured under the `.spec.template.spec.hooks` field of the **schedule** resource or the `.spec.hooks` field of the **backup** resource.

Configuration example and parameter descriptions:

```
hooks:
    resources:
      -
        # Hook name, displayed in logs.
        name: my-hook
        # (Optional) Namespaces to run the hook in (array). If not set, t
he hook runs in all namespaces.
        includedNamespaces:
        - '*'
        # (Optional) Namespaces to exclude (array).
        excludedNamespaces:
        - some-namespace
        # Resources to run the hook on. Currently supports only pods.
        includedResources:
        - pods
        # (Optional) Label selector for target resources.
        labelSelector:
          matchLabels:
            app: velero
            component: server
        # Pod actions to execute before backup (array). Only exec hooks a
re supported.
        pre:
          -
            exec:
              # (Optional) Name of container to execute the command in. I
f not set, defaults to the first container in the Pod.
              container: my-container
              # Command to execute (array). Required.
              command:
                - /bin/uname
                - -a
              # (Optional) Strategy for failed command execution: Continu
e or Fail. Default: Fail.
              onError: Fail
              # (Optional) Timeout for command execution. Default: 30s.
              timeout: 10s
        # Pod actions to execute after backup (array). Only exec hooks ar
e supported.
        post:
          # Same as pre
```

# Restore Hooks

The backup and restore component supports executing custom operations through hooks during or after the restore task.

Two types of restore hooks are supported:

- initContainer restore hook: adds an init container to the restored Pod to perform necessary setup before the application container starts.
- exec restore hook: used to execute commands or scripts in the container of the restored Pod.

Restore hooks can be set in the `.spec.hooks` field of the **restore** resource. Configuration example and parameter descriptions: