OPERATIONS GUIDE

A guide for administrators of CircleCI Server v3.3.0 on AWS or GCP

docs@circleci.com

Version 3.3.0, 01/21/2022: FINAL

CircleCI Server v3.x Operations Overview

The following guide contains information useful for CircleCI server Operators, or those responsible for ensuring CircleCI server 3.x is running properly through maintenance and monitoring.

It is assumed that you have already read the Server 3.x Overview.

CircleCI server schedules CI jobs using the Nomad scheduler. The Nomad control plane runs inside of Kubernetes, while the Nomad clients are provisioned outside the cluster. The Nomad clients need access to the Nomad control plane, output processor, and VM service.

CircleCI server can run Docker jobs on the Nomad clients, but it can also run jobs in a dedicated VM. These VM jobs are controlled by the Nomad clients, therefore the Nomad clients must be able to access the VM machines on port 22 for SSH and port 2376 for remote Docker jobs.

Job artifacts and outputs are sent directly from jobs in Nomad to object storage (S3, GCS, or other supported options). Audit logs and other items from the application are also stored in object storage, so both the Kubernetes cluster and the Nomad clients will need access to object storage.

Execution Environment

CircleCI server 3.x uses Nomad as the primary job scheduler. Refer to our Introduction to Nomad Cluster Operation to learn more about the job scheduler and how to perform basic client and cluster operations.

By default, CircleCI Nomad clients automatically provision compute resources according to the executors configured for each job in a project's `.circleci/config.yml` file.

Nomad Clients

Nomad Clients run without storing state, enabling you to increase or decrease the number of containers as needed.

To ensure enough Nomad clients are running to handle all builds, track the queued builds, and then increase the number of Nomad client machines as needed to balance the load. For more on tracking metrics see the Metrics and Monitoring section.

If a job's resource class requires more resources than the Nomad client's instance type has available, it will remain in a pending state. Choosing a smaller instance type for Nomad clients is a way to reduce cost, but will limit the Docker resource classes CircleCI can use. Review the available resource classes to decide what is best for you. The default instance type will run up to `xlarge` resource classes.

See the Nomad Documentation for options on optimizing the resource usage of Nomad clients.

The maximum machine size for a Nomad client is 128GB RAM/64 CPUs. Contact your CircleCI account representative to request use of larger machines for Nomad clients.

For more information on Nomad port requirements, see the Hardening Your Cluster section.

GitHub

CircleCI uses GitHub or GitHub Enterprise as an identity provider. GitHub Enterprise can, in turn, use SAML or SCIM to manage users from an external identity provider.

CircleCI does not support changing the URL or backend GitHub instance after it has been set up.

The following table describes the ports used on machines running GitHub to communicate with the services and Nomad client instances.

Source	Ports	Use
Services	22	Git access
Services	80 or 443	API access
Nomad Client	22	Git access
Nomad Client	80 or 443	API access

CircleCI Server v3.x Metrics and Monitoring

Metrics such as CPU or memory usage and internal metrics are useful in:

- Quickly detecting incidents and abnormal behavior
- Dynamically scaling compute resources
- Retroactively understanding infrastructure-wide issues

Metrics Collection

Scope

Your CircleCI server installation collects a number of metrics and logs by default, which can be useful in monitoring the health of your system and debug issues with your installation.

Data is retained for a maximum of 15 days.

Prometheus Server is not limited to only scrape metrics from your CircleCI server install. It will scrape metrics from your entire cluster by default. You may disable Prometheus from within the KOTS Admin Console config if needed.

Prometheus

Prometheus is a leading monitoring and alerting system for Kubernetes. Server 3.x ships with basic implementation of monitoring common performance metrics.

KOTS Admin - Metrics Graphs

By default, an instance of Prometheus is deployed with your CircleCI server install. Once deployed, you may provide the address for your Prometheus instance to the KOTS Admin Console. KOTS will use this address to generate graph data for the CPU and memory usage of containers in your cluster.

The default Prometheus address is `http://prometheus-server`

From the KOTS dashboard, select "configure graphs". Then enter `http://prometheus-server` and KOTS will generate resource usage graphs.

Telegraf

Most services running on server will report StatsD metrics to the Telegraf pod running in server. The configuration is fully customizable, so you can forward your metrics from Telegraf to any output that is supported by Telegraf via output plugins. By default, it will provide a metrics endpoint for Prometheus to scrape.

Use Telegraf to forward metrics to Datadog

The following example shows how to configure Telegraf to output metrics to Datadog:

Open the management console dashboard and select **Config** from the menu bar. Locate the **Custom**

Telegraf config section under **Observability and monitoring**. There is an editable text window where you can configure plugins for forwarding Telegraf metrics for your server installation. To forward to Datadog, add the following code, substituting `my-secret-key` with your Datadog API key:

```
[[outputs.datadog]]
  ## Replace "my-secret-key" with Datadog API key
  apikey = "my-secret-key"
```

For more options, see the Influxdata docs.

CircleCI Server v3.x Configuring a Proxy

Depending on your security requirements, you might want to install CircleCI server behind a proxy. Installing behind a proxy gives you the power to monitor and control access between your installation and the broader Internet.

Installation and configuration

There are two stages to installing CircleCI server behind a proxy. First, at the point of installation, the proxy addresses need to be specified, along with any addresses that should not be behind the proxy.

Installing behind a proxy

The installation process is described in detail in the CircleCI Server v3.x Installation guide. Both proxy and non-proxy addresses should be supplied using the arguments described here. The installation command should be in the format:

```
kubectl kots install circleci-server --http-proxy <my-http-proxy-uri> --https-proxy <my-https-proxy>
--no-proxy <my-no-proxy-list>
```

Configuring your proxy

Once you have installed server and accessed the management console, there are some fields that need to be completed in the configuration section, as shown in the screenshot below. These fields will not be automatically populated, and so the same proxy and no-proxy addresses you supplied during installation will need to be supplied here. If your proxy requires authentication in the form of a username and password, check the **HTTP Proxy authenticated** option to add the credentials.

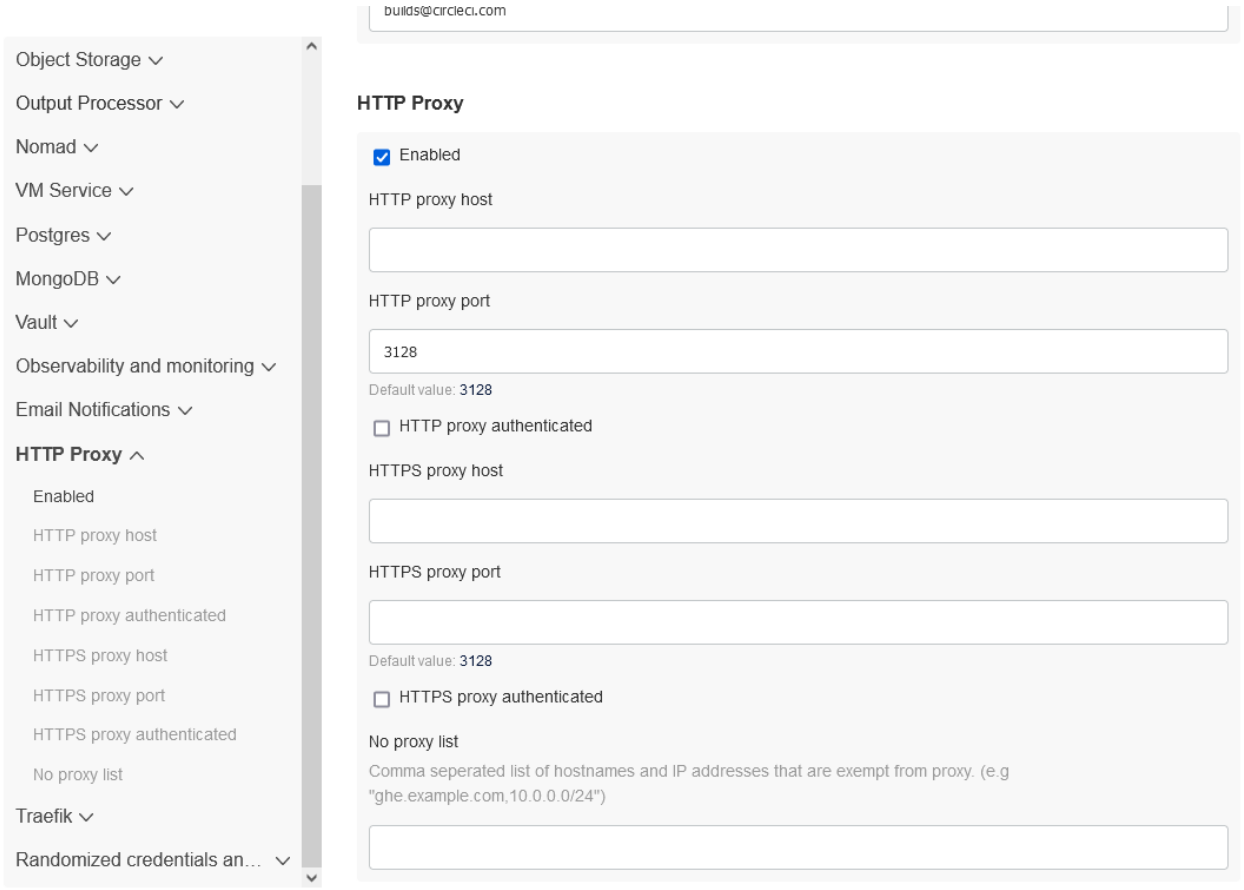


Figure 1. CircleCI Server v3.x Configuring a Proxy

Known limitations

- Installing behind a proxy will prevent the use of CircleCI runner.
- Some additional configuration will be required to import orbs when installed behind a proxy. See Orbs on Server docs for more information.
- The JVM only accepts proxies that run over HTTP, not HTTPS, and therefore proxy URIs must be of the form `http://user:password@host:port` rather than `https://user:password@host:port`.
- If your GitHub instance is running outside of the proxied environment (either GitHub.com or GitHub Enterprise), you must ensure that SSH traffic from our application (inside the Kubernetes cluster) and from our Nomad node can reach your instance without additional configuration. Our SSH agents do not respect proxy settings because it uses a different network protocol. If a proxy is the only way that any traffic can reach outside the proxied environment, this means it will block SSH traffic and our application will fail.
- The load balancer endpoints must be added to the no-proxy list for the following services: `output processor` and `vm-service`. This is because the no-proxy list is shared between the application and build-agent. The application and build-agent are assumed to be behind the same firewall and therefore cannot have a proxy between them.
- The KOTS Admin Console cannot be upgraded if proxy setting were configured. The proxy settings will be deleted and cause the KOTS Admin Console to break.
- If you install server behind a proxy, you may need to provide a custom image for VM service. Visit the CircleCI Linux Image Builder repo for further information.

- If object storage is outside the proxy, no job features that use object storage will work. This includes:
  - Artifacts
  - Test results
  - Cache save and restore
  - Workspaces

  Users can get around this restriction by setting environment variables on their jobs. For example:

```
jobname:
  docker:
  - image: ubuntu:latest
    environment:
      HTTP_PROXY: http://proxy.example.com:3128
      HTTPS_PROXY: http://proxy.example.com:3128
      NO_PROXY: whatever.internal,10.0.1.2
```

It is crucial that these environment variables are set in this particular location because its the only location that propagates them to the correct service.

CircleCI Server v3.x User Accounts

This section provides information to help operators manage user accounts. For an overview of user accounts, see the Admin settings overview from the CircleCI app by clicking on your profile in the top right corner and selecting **Admin**.

Suspending Accounts

This section covers how to suspend new, active, or inactive accounts.

New Accounts

Any user associated with your GitHub organization can create a user account for your CircleCI Server installation. To control who has access, you can choose to automatically suspend all new users, requiring an administrator to activate them before they can log in. To access this feature:

1. Navigate to your CircleCI Admin Settings
2. Select **System Settings** from the Admin Settings menu
3. Set **Suspend New Users** to **True**

Active Accounts

When an account is no longer required, you can suspend the account. It will no longer be active and will not count against your license quota. To suspend an account:

1. Navigate to your CircleCI Admin Settings
2. Select **Users** from the Admin Settings menu
3. Scroll to locate the account in either the Active or Inactive window
4. Click **Suspend** next to the account name and the account will appear in the Suspended window

Inactive Accounts

Inactive accounts are those that have been approved by the administrator of the server installation but have not logged into the system successfully. These accounts do not count against your available server seats.

Reactivating Accounts

This section covers how to reactivate new or previously active accounts.

New Accounts

To activate a new account that was automatically suspended and allow the associated user access to your installation of CircleCI Server:

1. Navigate to your CircleCI Admin Settings
2. Select **Users** from the Admin Settings menu
3. View the **Suspended New Users** window

4. Click on **Activate** next to the User you wish to grant access and the account will appear in the Active Window

Previously Active Accounts

To reactivate an account that has been suspended:

1. Navigate to your CircleCI Admin Settings
2. Select **Users** from the Admin Settings menu
3. View the Suspended window
4. Click on **Activate** next to the User you wish to grant access and the account will appear in the Active window.

Limiting Registration by GitHub Organization

When using GitHub.com, you can limit who can register with your CircleCI install to people with some connection to your approved organizations list. To access this feature:

1. Navigate to your CircleCI Admin Settings page
2. Select System Settings from the Admin Setting menu
3. Scroll down to Required Org Membership List
4. Enter the organization(s) you wish to approve. If entering more than one organization, use a comma-delimited string.

CircleCI Server v3.x Orbs

This section describes how to manage orbs for an installation of server v3.x. Server installations include their own local orb registry. All orbs referenced in configs refer to the orbs in the server orb registry. You are responsible for maintaining orbs. This includes copying orbs from the public registry, updating orbs that may have been previously copied, and registering your company's private orbs, if they exist.

For information on orbs and related use cases, see the Orb docs.

If you are looking for information on creating an orb, see the Introduction to Authoring Orbs.

Managing Orbs

Orbs are accessed via the CircleCI CLI. Orbs require your CircleCI user to be an admin. They also require a personal API token. Please ensure that you are using a personal API token generated *after* your user account is made an admin.

Providing a local repository location using the `--host` option allows you to access your local server orbs, rather than public cloud orbs. For example, if your server installation is located at `http://circleci.somehostname.com`, then you can run orb commands local to that orb repository by passing `--host http://cirlceci.somehostname.com`.

List available orbs

To list available public orbs, visit the orb directory, or run the following command:

```
circleci orb list
```

To list available private orbs (registered in your local server orb repository), run the following command:

```
circleci orb list --host <your server install domain> --token <your api token>
```

Import a public orb

To import a public orb to your local server orb repository, run the following command:

```
circleci admin import-orb ns[orb[@version]] --host <your server installation domain> --token <your
api token>
```

Fetch a public orb's updates

To update a public orb in your local server orb repository with a new version, run:

```
circleci admin import-orb ns[orb[@version]] --host <your server installation domain> --token <your
api token>
```

Using orbs behind a proxy

When importing orbs, the CLI must be able to talk to the server installation and to circleci.com. If you want to do this when using a server installation behind a proxy, the CLI needs to be configured to use the proxy to make those requests to `circleci.com`, rather than proxying requests to the server install. For example:

```
export NO_PROXY=server.example.com
export HTTPS_PROXY=http://proxy.example.com:3128
export HTTP_PROXY=http://proxy.example.com:3128
circleci admin import-orb ns[orb[@version]] --host <your server installation domain> --token <your
api token>
```

CircleCI Server v3.x VM Service

CircleCI server's VM service controls how `machine` executor (Linux and Windows images) and Remote Docker jobs are run.

This section describes the available configuration options for VM service. These config options are all accessible from the KOTS Admin Console by choosing the **Config** tab from your dashboard.

We recommend that you leave these options at their defaults until you have successfully configured and verified the core and build services of your server installation. Steps to set up VM service are provided in the server 3.x installation guide for AWS and GCP.

VM service settings

You will need to provide the hostname for your VM service load balancer:

1. Once your server installation is up and running, run the following command:

```
kubectl get svc/vm-service
```

2. Enter the address listed under **External IP** in the **VM Service Load Balancer Hostname** field.

There is also an option to change the port used for VM service. The default is `3000` and this should only be changed if you are guided to do so by a CircleCI support engineer.

VM provider

The following configuration options are for the VM provider: either AWS or GCP.

AWS EC2

You will need to complete the following fields to configure your VM Service to work with AWS EC2.

At this point you can uncheck the **Assign Public IPs** check box if you need VMs to use private IP addresses.

- **AWS Region** (required): This is the region in which the application is hosted.
- **AWS Linux AMI ID** (optional): If you wish to provide a custom AMI for Linux `machine` executors you can supply an AMI ID here. To create a Linux image, use the CircleCI Server Linux Image Builder. If you leave this field blank, a default AMI will be used.
- **AWS Windows AMI ID** (optional): If you require Windows executors, you can supply an AMI ID for them here. To create a Windows image, use the CircleCI Server Windows Image Builder. Leave this field blank if you do not require Windows executors.
- **Subnets** (required): Choose subnets (public or private) where the VMs should be deployed. Note that all subnets must be in the same availability zone.
- **Security Group ID** (required): This is the security group that will be attached to the VMs.

  The recommended security group configuration can be found in the Hardening Your Cluster section.

- **Number of <VM-type> VMs to keep prescaled**: By default, this field is set to 0, which will create and provision instances of a resource type on demand. You have the option of preallocating up to 5 instances per resource type. Preallocating instances lowers the start time, allowing for faster machine and `remote_docker` builds.

Preallocated instances are always running and could potentially increase costs. Decreasing this number may also take up to 24 hours for changes to take effect. You have the option of terminating those instances manually, if required.

If Docker Layer Caching (DLC) is to be used, VM Service instances need to be spun up on demand. To ensure this can happen, **either** check any preallocated instances are in use, **or** set both remote Docker and `machine` preallocated instance fields to `0`.

When using preallocated instances, be aware that a cron job is scheduled to cycle through these instances once per day to ensure they don't end up in an unworkable state.

Authentication

One of the following is required. Either select "IAM Keys" and provide:

- **Access Key ID** (required): Access Key ID for EC2 access.

The Access Key and Secret Key used by VM Service differs from the policy used by object storage in the previous section.

- **Secret Key** (required): Secret Key for EC2 access.

Or select "IAM role" and provide:

- **Role ARN** (required): Role ARN for Service Accounts (Amazon Resource Name) for EC2 access.

Default AWS AMI list

The default AMIs for server v3.x are based on Ubuntu 20.04.

```
"us-east-1" "ami-04f249339fa8afc90"
"ca-central-1" "ami-002f61fb4f6cd4f04"
"ap-south-1" "ami-0309e6438340ff3f5"
"ap-southeast-2" "ami-03ac956e1d298b76a"
"ap-southeast-1" "ami-0272b002478c96552"
"eu-central-1" "ami-07266a91e4ef7e3e8"
"eu-west-1" "ami-0bc8a965f9ae82e44"
"eu-west-2" "ami-0bcbed1cffe3866c2"
"sa-east-1" "ami-05291e231356c0387"
"us-east-2" "ami-08735066168c5c8e9"
"us-west-1" "ami-035e0e862838fcb21"
"us-west-2" "ami-0b4970c467d8baaef"
"ap-northeast-1" "ami-0b9233227f60abc2c"
"ap-northeast-2" "ami-08e7a9df6ab2f6b9d"
"eu-west-3" "ami-07f0d51c7621f0c39"
"us-gov-east-1" "ami-0f68718afd37587ae"
"us-gov-west-1" "ami-8e2106ef"
```

Google Cloud Platform

You will need the following fields to configure your VM service to work with Google Cloud Platform (GCP).

We recommend you create a unique service account used exclusively by VM Service. The Compute Instance Admin (Beta) role is broad enough to allow VM Service to operate. If you wish to make permissions more granular, you can use the Compute Instance Admin (beta) role documentation as reference.

At this point you can uncheck the **Assign Public IPs** check box if you need VMs to use private IP addresses.

- **GCP project ID** (required): Name of the GCP project the cluster resides.
- **GCP Zone for your VMs** (required): GCP zone the virtual machines instances should be created in, for example `us-east1-b`.
- **GCP Windows Image** (optional): If you require Windows executors, you can supply an AMI ID for them here. To create a Windows image, use the CircleCI Server Windows Image Builder. Leave this field blank if you do not require Windows executors.
- **GCP VPC Network** (required): Name of the VPC Network. If you are deploying CircleCI server in a shared VPC, use the full network endpoint for the host network rather than the name, for example:

  ```
  https://www.googleapis.com/compute/v1/projects/<host-project>/global/networks/<network-name>
  ```

- **GCP VPC Subnet** (optional): Name of the VPC Subnet. If using auto-subnetting, leave this field blank. If you are deploying CircleCI server in a shared VPC, use the full network endpoint for the shared

subnetwork rather than the name, for example:

```
https://www.googleapis.com/compute/v1/projects/<service-project>/regions/<your-region>/subnetworks/<subnetwork-name>
```

- **GCP Service Account JSON file** (required): Copy and paste the contents of your service account JSON file.
- **Number of <VM-type> VMs to keep prescaled**: By default, this field is set to 0, which will create and provision instances of a resource type on demand. You have the option of preallocating up to 5 instances per resource type. Preallocating instances lowers the start time allowing for faster machine and `remote_docker` builds.

Preallocated instances are always running and could potentially increase costs. Decreasing this number may also take up to 24 hours for changes to take effect. You have the option of terminating those instances manually, if required.

If Docker Layer Caching (DLC) is to be used, VM Service instances need to be spun up on demand. To ensure this can happen, **either** ensure any preallocated instances are in use, **or** set both remote Docker and `machine` preallocated instance fields to `0`.

When using preallocated instances be aware that a cron job is scheduled to cycle through these instances once per day to ensure they do not end up in an unworkable state.

CircleCI Server v3.x Configuring External Services

This document describes how to configure the following external services for use with a CircleCI server 3.x installation. The settings described in this guide can be found in the KOTS Admin Console. Access the KOTS Admin Console by running the following command, substituting your namespace: `kubectl kots admin-console -n <YOUR_CIRCLECI_NAMESPACE>`

- PostgreSQL
- MongoDB
- Vault

PostgreSQL

If using your own PostgreSQL instance, it needs to be version 12.1 or greater.

Postgres

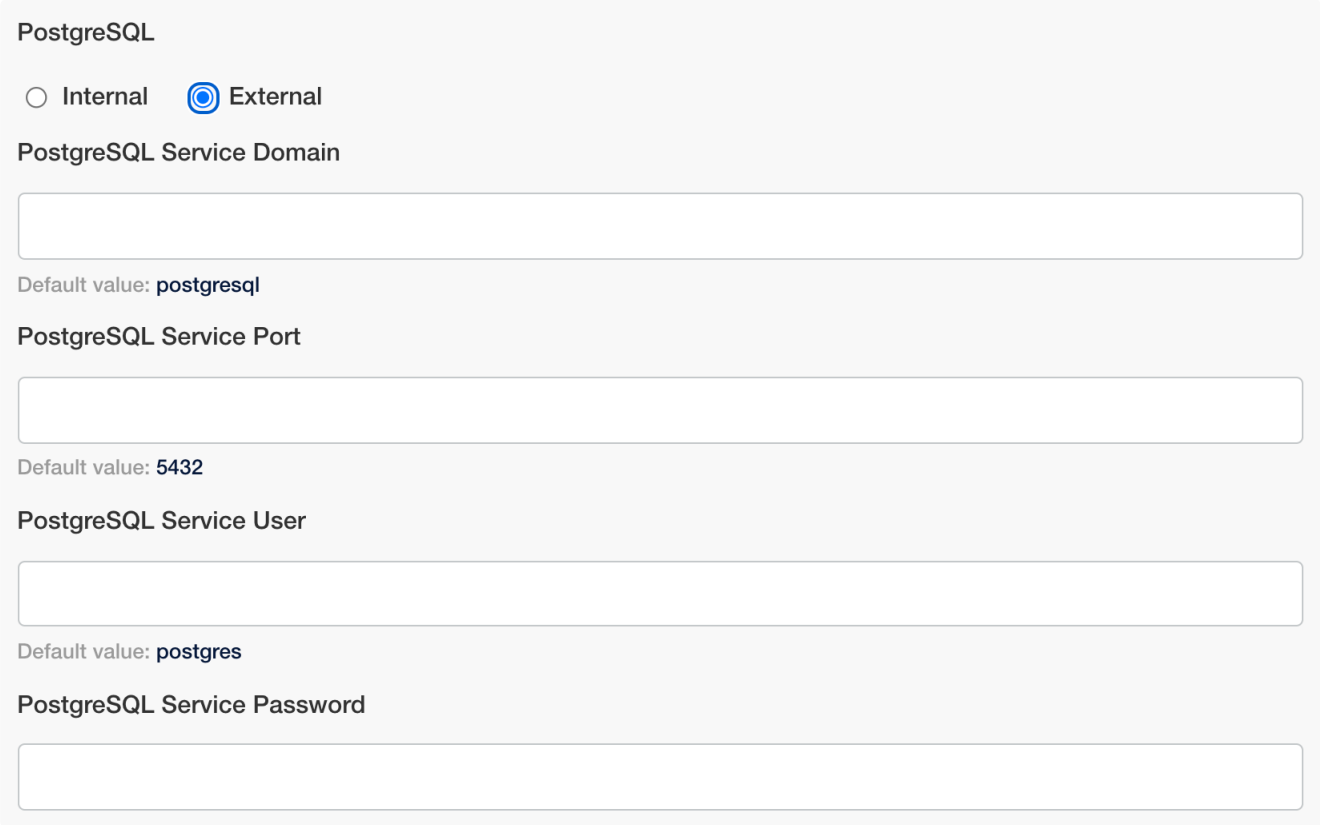


Figure 2. External PostgreSQL

If you choose to use an external PostgreSQL instance, complete the following fields:

- **PostgreSQL Service Domain (required)** - The domain or IP address of your PostgreSQL instance.
- **PostgreSQL Service Port (required)** - The port of your PostgreSQL instance.
- **PostgreSQL Service Username (required)** - A user with the appropriate privileges to access your PostgreSQL instance.

- **PostgreSQL Service Password (required)** - The password of the user used to access your PostgreSQL instance.

Best Practices for your PostgreSQL

Consider running at least two PostgreSQL replicas to allow recovery from primary failure and for backups. The table below shows the recommended specifications for PostgreSQL machines:

# of Daily Active Users	# of PostgreSQL Replicas	CPU	RAM	Disk	NIC Speed
<50	2	8 Cores	16 GB	100 GB	1 Gbps
50 - 250	2	8 Cores	16 GB	200 GB	1 Gbps
250 - 1000	3	8 Cores	32 GB	500 GB	10 Gbps
1000 - 5000	3	8 Cores	32 GB	1 TB	10 Gbps
5000+	3	8 Cores	32 GB	1 TB	10 Gbps

Backing Up PostgreSQL

PostgreSQL provides official documentation for backing up and restoring your PostgreSQL 12 install, which can be found here.

We strongly recommend the following:

- Taking daily backups
- Keeping at least 30 days of backups
- Using encrypted storage for backups as databases might contain sensitive information
- Performing a backup before each upgrade of CircleCI server.

MongoDB

If using your own MongoDB instance, it needs to be version 3.6 or greater.

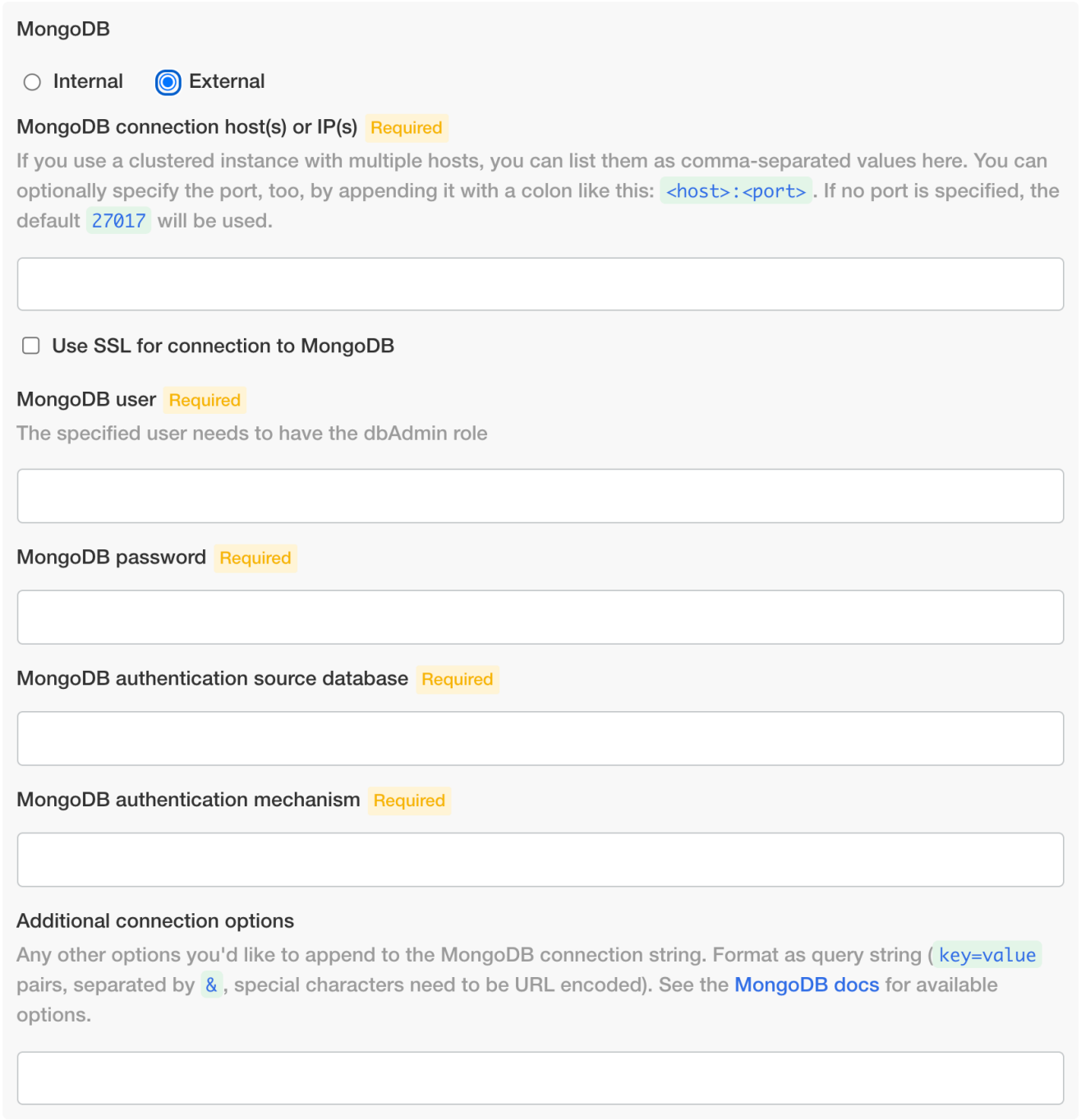


Figure 3. External MongoDB

If you choose to use an external MongoDB instance, complete the following fields:

- **MongoDB connection host(s) or IP(s) (required)** - The hostname or IP of your MongoDB instance. Specifying a port using a colon and multiple hosts for sharded instances are both supported.
- **Use SSL for connection to MongoDB (required)** - Whether to use SSL when connecting to your external MongoDB instance.
- **Allow insecure TLS connections (required)** - If you use a self-signed certificate or one signed by a custom CA, you will need to enable this setting. However, this is an insecure setting and you should use a TLS certificate signed by a valid CA, if you can.
- **MongoDB user (required)** - The username for the account to use. This account should have the

dbAdmin role.

- **MongoDB password (required)** - The password for the account to use.
- **MongoDB authentication source database (required)** - The database that holds the account information, usually admin.
- **MongoDB authentication mechanism (required)** - The authentication mechanism to use, usually SCRAM-SHA-1.
- **Additional connection options (optional)** - Any other connection options you would like to use. This needs to be formatted as a query string (`key=value` pairs, separated by `&`. Special characters must be URL encoded). See the MongoDB docs for available options.

Vault

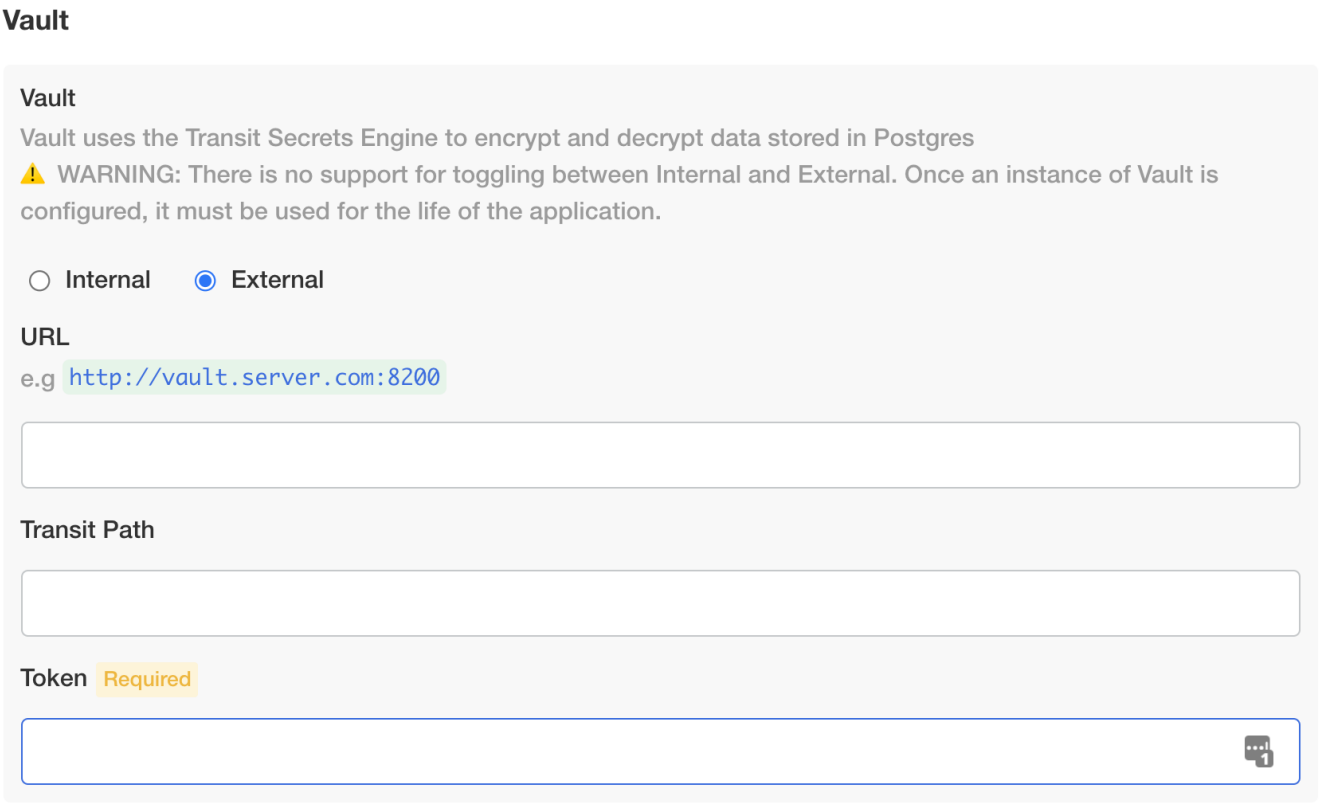


Figure 4. External Vault

If you choose to use an external Vault instance, complete the following fields:

- **URL** - The URL to your Vault service.
- **Transit Path** - Your Vault secrets transit path.
- **Token** - The access token for vault.

CircleCI Server v3.x Internal Database Volume Expansion

Overview

Persistent volume expansion of MongoDB and Postgres is available for server v3.2.0 and up.

If you have chosen to deploy either of the CircleCI databases (MongoDB or Postgres) within the cluster, rather than externally provisioning these databases, then there may come a point at which the storage space initially made available to these databases is no longer sufficient. Internal databases in your Kubernetes cluster make use of persistent volumes for persistent storage. The size of these volumes is determined by persistence volume claims (PVCs). These PVCs request storage space based on what has been made available to the nodes in your cluster.

This document runs through the steps required to increase PVCs to expand the space available to your internally deployed databases. This operation should not require any downtime, unless you need to restart your database pods.

Expanding persistent volumes does not affect the size of the storage attached to your nodes. Expanding node storage remains within the limitations of your cloud provider. Please refer to the docs for your chosen cloud provider for details on how to expand the storage attached to your cluster's nodes.

Resizing persistent volume claims

Below are the steps detailing how to resize the persistent volume claims for Postgres and MongoDB. We will confirm the size of the claims and the disk space made available to our databases before and after this operation.

As a precaution, it's always good to create a backup of your cluster first.

Step 0 - Confirm current volume size

By default, the persistent volume claims used by our internal databases have a capacity of 8Gi. However, this initial value can be set at the time of first deployment from the KOTS Admin Console. We can confirm the size of our persistent volume claim capacity using the command: `kubectl get pvc <pvc-name>`.

For postgres:

```
circleci-user ~ $ kubectl get pvc data-postgresql-0
NAME                STATUS   VOLUME                                     CAPACITY   ACCESS MODES
STORAGECLASS   AGE
data-postgresql-0   Bound    pvc-c2a2d97b-2b7d-47d3-ac77-d07c76c995a3   8Gi        RWO
gp2            1d
```

For mongodb:

```
circleci-user ~ $ kubectl get pvc datadir-mongodb-0
NAME                STATUS   VOLUME                                     CAPACITY   ACCESS MODES
STORAGECLASS   AGE
datadir-mongodb-0   Bound    pvc-58a2274c-31c0-487a-b329-0062426b5535   8Gi        RWO
gp2            1d
```

We can also confirm this capacity is made available to a database by checking the size of its data directory.

For postgres, the directory is `/bitnami/postgresql`. We can confirm its size using the command below.

```
circleci-user ~ $ kubectl exec postgresql-0 -- df -h /bitnami/postgresql
Filesystem      Size  Used Avail Use% Mounted on
/dev/nvme4n1    7.8G  404M  7.4G   3% /bitnami/postgresql
```

For mongodb, the directory is `/bitnami/mongodb`.

```
circleci-user ~ $ kubectl exec mongodb-0 -- df -h /bitnami/mongodb
Filesystem      Size  Used Avail Use% Mounted on
/dev/nvme1n1    7.8G  441M  7.4G   3% /bitnami/mongodb
```

From the examples above, the capacities are still 8Gi. The following steps show how to increase this to 10Gi.

Step 1 - Confirm volume expansion is allowed

First, confirm that volume expansion is allowed in your cluster.

```
circleci-user ~ $ kubectl get sc
NAME            PROVISIONER             RECLAIMPOLICY   VOLUMEBINDINGMODE      ALLOWVOLUMEEXPANSION
AGE
gp2 (default)   kubernetes.io/aws-ebs   Delete          WaitForFirstConsumer   false
1d
```

As we can see, our default storage class does not allow volume expansion. However, we can change this with the `kubectl patch` command:

```
circleci-user ~ $ kubectl patch sc gp2 -p '{"allowVolumeExpansion": true}'
storageclass.storage.k8s.io/gp2 patched
circleci-user ~ $ kubectl get sc
NAME            PROVISIONER             RECLAIMPOLICY   VOLUMEBINDINGMODE      ALLOWVOLUMEEXPANSION
AGE
gp2 (default)   kubernetes.io/aws-ebs   Delete          WaitForFirstConsumer   true
1d
```

Now we may proceed to expanding our volumes.

Step 2 - Delete the database's stateful set

In this step we will delete the stateful set, which controls our database pod. The command below will delete the referenced database's stateful set without deleting the pod. We do not want to delete the pod itself as this would cause downtime. In the following steps, we will be redeploying our stateful set. You might chose to delete one or both stateful sets, depending on which database volumes you wish to expand. The `--cascade=false` flag is most important here.

For postgres:

```
kubectl delete sts postgresql --cascade=false
```

For mongodb:

```
kubectl delete sts mongodb --cascade=false
```

Step 3 - Update the size of the database's PVC

Now that the stateful set has been removed, we can increase the size of our persistent volume claim to 10Gi.

For postgres:

```
kubectl patch pvc data-postgresql-0 -p '{"spec": {"resources": {"requests": {"storage": "10Gi"}}}}'
```

For mongodb:

```
kubectl patch pvc datadir-mongodb-0 -p '{"spec": {"resources": {"requests": {"storage": "10Gi"}}}}'
```

Step 4 - Update KOTS Admin Console with the new PVC size

Now we need to head over to the KOTS Admin Console to persist our changes. In the config section, we will update the values for our PVC size to 10Gi as shown below.

Postgres

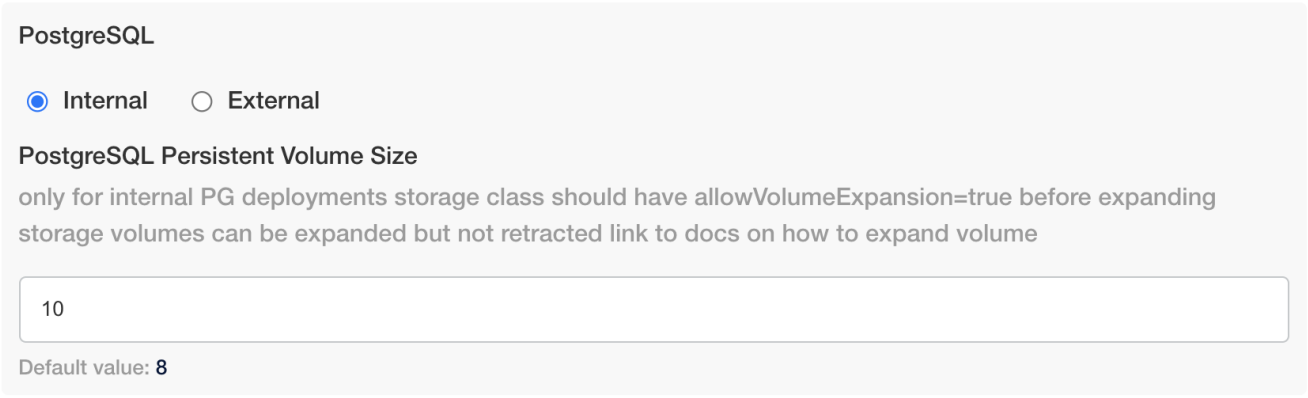


Figure 5. Postgres

MongoDB

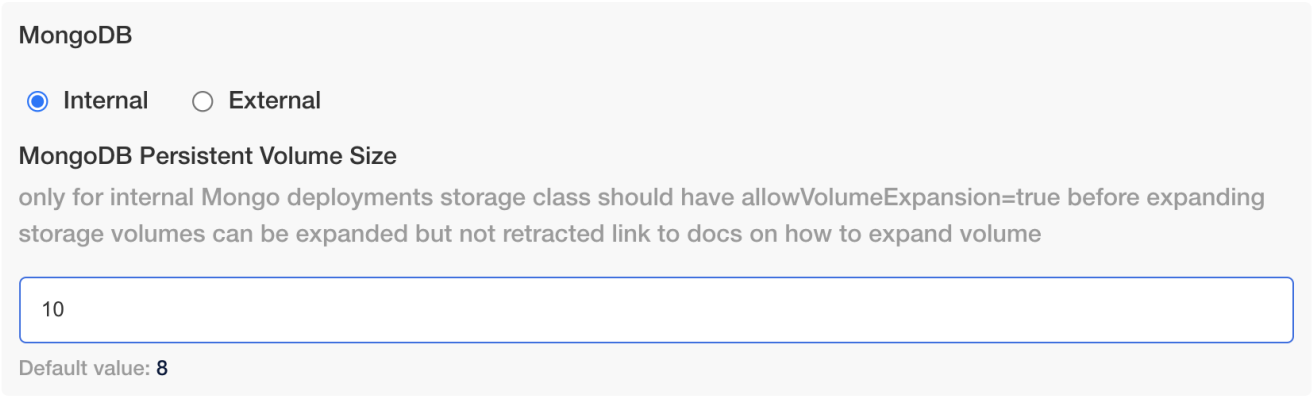


Figure 6. MongoDB

Now save and deploy your changes. This will recreate the stateful set(s) that we destroyed earlier, but with the new PVC sizes, which will persist through new releases.

Step 5 - Validate new volume size

Once deployed, we can validate the size of the data directories assigned to our databases.

For postgres the directory is `/bitnami/postgresql`.

```
circleci-user ~ $ kubectl exec postgresql-0 -- df -h /bitnami/postgresql
Filesystem      Size  Used Avail Use% Mounted on
/dev/nvme4n1    9.8G  404M  9.4G   5% /bitnami/postgresql
```

For mongodb the directory is `/bitnami/mongodb`.

```
circleci-user ~ $ kubectl exec mongodb-0 -- df -h /bitnami/mongodb
Filesystem      Size  Used Avail Use% Mounted on
/dev/nvme1n1    9.8G  441M  9.3G   5% /bitnami/mongodb
```

As we can see, the size of our directories has been increased.

When completing these steps, if you find, as expected, that the new pods *do* show the resized volumes, it is still worth checking with the `kubectl describe` commands shown below. In some instances the resize will fail, but the only way to know is by viewing an event in the output from `kubectl describe`.

For postgres:

```
kubectl describe pvc data-postgresql-0
```

For mongodb:

```
kubectl describe pvc datadir-mongodb-0
```

Success looks like this example:

```
Events:
Type    Reason                      Age   From     Message

Normal  FileSystemResizeSuccessful  19m   kubelet  MountVolume.NodeExpandVolume succeeded for volume
"pvc-b3382dd7-3ecc-45b0-aeff-45edc31f48aa"
```

Failure might look like this example:

```
Warning  VolumeResizeFailed  58m   volume_expand  error expanding volume "circleci-server/datadir-
mongodb-0" of plugin "kubernetes.io/aws-ebs": AWS modifyVolume failed for vol-08d0861715c313887 with
VolumeModificationRateExceeded: You've reached the maximum modification rate per volume limit. Wait
at least 6 hours between modifications per EBS volume.
status code: 400, request id: 3bd43d1e-0420-4807-9c33-df26a4ca3f23
Normal   FileSystemResizeSuccessful  55m (x2 over 81m)  kubelet        MountVolume.NodeExpandVolume
succeeded for volume "pvc-29456ce2-c7ff-492b-add4-fcf11872589f"
```

Troubleshooting

If you find that after following these steps, the disk size allocated to your data directories has not increased, then you may need to restart your database pods. This will cause downtime of 1-5 mins while the databases restart. You can use the commands below to restart your databases.

For postgres:

```
kubectl rollout restart sts postgresql
```

For mongodb:

```
kubectl rollout restart sts mongodb
```

CircleCI Server v3.x Load Balancers

CircleCI server uses load balancers to manage network traffic entering and leaving the Kubernetes services cluster. The three load balancers managing traffic to the Nomad cluster are internal to the VPC and they manage the distribution of jobs to the various compute resources.

The frontend load balancer manages traffic coming from developers and your VCS, including through the API, the CLI, and CircleCI app. The frontend load balancer is public by default, but can be made private.

Make the frontend load balancer private

Webhooks: If you choose to make the frontend load balancer private, the following conditions must be met, dependent on VCS, for webhooks to work:

- **GitHub Enterprise** – your CircleCI server installation must be in the same internal network as GHE.
- **GitHub.com** – set up a proxy for incoming webhooks and set it as override for the webhook host URL. This setting can be found under **Admin Settings** > **System Settings** > **Override webhook host URL** from the CircleCI app.

The Private load balancers option only works with installations on CircleCI server on GKE or EKS.

1. From the management console, select **Config** from the menu bar and locate the **Private load balancers** option under **General Settings**.
2. Check the box next to **Private load balancers**.

If you are using Let's Encrypt TLS certificates, this box will not be visible as Let's Encrypt doesn't work with private installations. Uncheck the box for Let's Encrypt to make the Private load balancer option appear.

If you are changing this setting after the initial deployment of CircleCI server, you may need to delete the old public load balancer so that Kubernetes will request a new load balancer with the new configuration.

CircleCI Server v3.x Authentication

CircleCI server currently supports OAuth through GitHub or GitHub Enterprise.

The default method for user account authentication in CircleCI server is through GitHub.com/GitHub Enterprise OAuth.

After your installation is up and running, provide users with a link to access the CircleCI application - for example, `<your-circleci-hostname>.com` – and they will be prompted to set up an account by running through the GitHub/GitHub Enterprise OAuth flow before being redirected to the CircleCI login screen.

Using Docker Authenticated Pulls

This document describes how to authenticate with your Docker registry provider to pull images.

Authenticated pulls allow access to private Docker images. It may also grant higher rate limits, depending on your registry provider.

CircleCI has partnered with Docker to ensure that our users can continue to access Docker Hub without rate limits. As of November 1st 2020, with few exceptions, you should not be impacted by any rate limits when pulling images from Docker Hub through CircleCI. However, these rate limits may be implemented for CircleCI users in the future. This is why we are encouraging you and your team to add Docker Hub authentication to your CircleCI configuration and consider upgrading your Docker Hub plan, as appropriate, to prevent any impact from rate limits in the future.

Docker executor

For the Docker executor, specify a username and password in the `auth` field of your config.yml file. To protect the password, place it in a context, or use a per-project Environment Variable.

Server 2.x customers may instead set up a Docker Hub pull through a registry mirror. Pulls through Docker Hub registry mirrors are not yet available on server 3.x.

Contexts are the more flexible option. CircleCI supports multiple contexts, which is a great way modularize secrets, ensuring jobs can only access what they *need*.

In this example, we grant the "build" job access to Docker credentials context, `docker-hub-creds`, without bloating the existing `build-env-vars` context:

```
workflows:
  my-workflow:
    jobs:
      - build:
          context:
            - build-env-vars
            - docker-hub-creds

jobs:
  build:
    docker:
      - image: acme-private/private-image:321
        auth:
          username: mydockerhub-user  # can specify string literal values
          password: $DOCKERHUB_PASSWORD  # or project environment variable reference
```

If you have two-factor authentication set up on Docker Hub, you can simply use your personal access token for the `password` key instead. For example:

```
- image: acme-private/private-image:321
  auth:
    username: mydockerhub-user
    password: $DOCKERHUB_ACCESS_TOKEN
```

You can also use images from a private repository like gcr.io or quay.io. Ensure you supply the full registry/image URL for the `image` key, and use the appropriate username/password for the `auth` key. For example:

```
- image: quay.io/project/image:tag
  auth:
    username: $QUAY_USERNAME
    password: $QUAY_PASSWORD
```

Machine executor (with Docker orb)

Alternatively, you can utilize the `machine` executor to achieve the same result using the Docker orb:

```
version: 2.1
orbs:
  docker: circleci/docker@1.4.0

workflows:
  my-workflow:
    jobs:
      - machine-job:
          context:
            - build-env-vars
            - docker-hub-creds

jobs:
  machine-job:
    machine: true
    steps:
      - docker/check:
          docker-username: DOCKERHUB_LOGIN  # DOCKER_LOGIN is the default value, if it exists, it
automatically would be used.
          docker-password: DOCKERHUB_PASSWORD  # DOCKER_PASSWORD is the default value
      - docker/pull:
          images: 'circleci/node:latest'
```

Machine executor (with Docker CLI)

Or with the CLI:

```
version: 2
jobs:
  build:
    machine: true
    working_directory: ~/my_app
    steps:
      # Docker is preinstalled, along with docker-compose
      - checkout

      # start proprietary DB using private Docker image
      - run: |
          docker login -u $DOCKER_USER -p $DOCKER_PASS
          docker run -d --name db company/proprietary-db:1.2.3
```

AWS ECR

CircleCI now supports pulling private images from Amazon's ECR service.

You can pull your private images from ECR repositories in any regions. However, for the best experience, we strongly recommend you make a copy of your image in `us-east-1` region, and specify that `us-east-1` image for the Docker executor. Our job execution infrastructure is in the `us-east-1` region, so using `us-east-1` images accelerates the process of spinning up your environment.

You can start using private images from ECR in one of two ways:

1. Set your AWS credentials using standard CircleCI private environment variables.
2. Specify your AWS credentials in `.circleci/config.yml` using `aws_auth`:

```
version: 2
jobs:
  build:
    docker:
      - image: account-id.dkr.ecr.us-east-1.amazonaws.com/org/repo:0.1
        aws_auth:
          aws_access_key_id: AKIAQWERVA  # can specify string literal values
          aws_secret_access_key: $ECR_AWS_SECRET_ACCESS_KEY  # or project UI envar reference
```

Both options are virtually the same. However, the second option enables you to specify the variable name you want for the credentials. This can be useful where you have different AWS credentials for different infrastructures. For example, your SaaS app runs the speedier tests and deploys to staging infrastructure on every commit, while for git tag pushes, we run the complete test suite before deploying to production:

```yaml
version: 2
jobs:
  build:
    docker:
      - image: account-id.dkr.ecr.us-east-1.amazonaws.com/org/repo:0.1
        aws_auth:
          aws_access_key_id: $AWS_ACCESS_KEY_ID_STAGING
          aws_secret_access_key: $AWS_SECRET_ACCESS_KEY_STAGING
    steps:
      - run:
          name: "Every Day Tests"
          command: "testing...."
      - run:
          name: "Deploy to Staging Infrastructure"
          command: "something something darkside.... cli"
  deploy:
    docker:
      - image: account-id.dkr.ecr.us-east-1.amazonaws.com/org/repo:0.1
        aws_auth:
          aws_access_key_id: $AWS_ACCESS_KEY_ID_PRODUCTION
          aws_secret_access_key: $AWS_SECRET_ACCESS_KEY_PRODUCTION
    steps:
      - run:
          name: "Full Test Suite"
          command: "testing...."
      - run:
          name: "Deploy to Production Infrastructure"
          command: "something something darkside.... cli"

workflows:
  version: 2
  main:
    jobs:
      - build:
          filters:
            tags:
              only: /^\d{4}\.\d+$/
      - deploy:
          requires:
            - build
          filters:
```

```
            branches:
              ignore: /.*/
            tags:
              only: /^\d{4}\.\d+$/
```

CircleCI Server v3.x build artifacts

Build artifacts persist data after a job is completed. They can be used for longer-term storage of your build process outputs. For example, when a Java build/test process finishes, the output of the process is saved as a `.jar` file. CircleCI can store this file as an artifact, keeping it available long after the process has finished.

Safe and unsafe content types

By default, only predefined artifact types are allowed to be rendered. This protects users from uploading, and potentially executing, malicious content. The 'allowed-list' is as follows:

Category	Safe Type
Text	Plain
Application	json
Image	png
Image	jpg
Image	gif
Image	bmp
Video	webm
Video	ogg
Video	mp4
Audio	webm
Audio	aac
Audio	mp4
Audio	mpeg
Audio	ogg
Audio	wav

Also, by default, the following types will be rendered as plain text:

Category	Unsafe Type
Text	html
Text	css
Text	javascript
Text	ecmascript
Application	javascript
Application	ecmascript
Text	xml

Allow unsafe types

You can choose to allow unsafe types to be rendered, if required, by checking the **Server Unsafe Artifacts** checkbox within the **Frontend Settings** in the KOTS Admin Console.

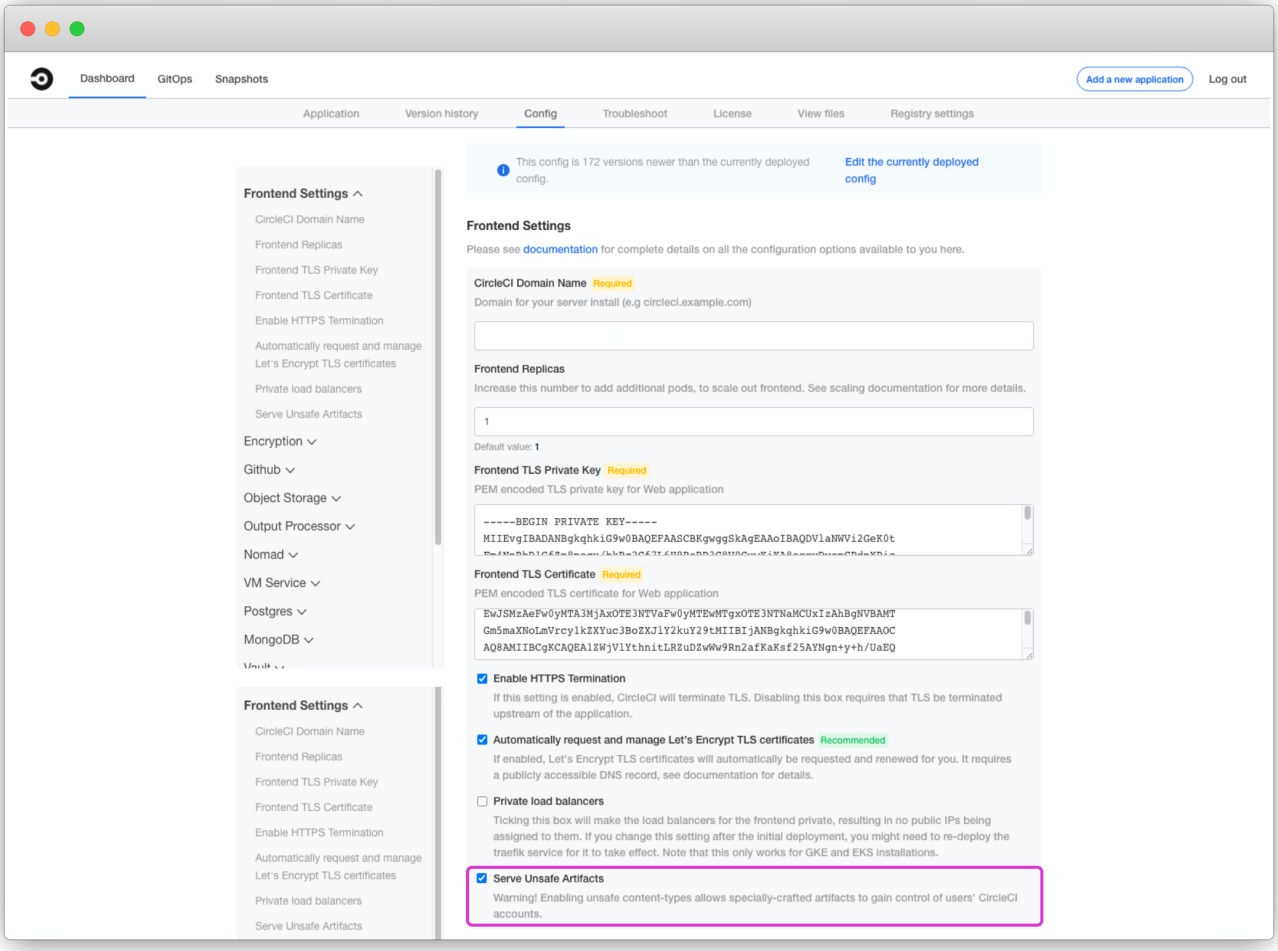


Figure 7. CircleCI Server v3.x Artifacts

CircleCI Server v3.x Usage Data

CircleCI typically collects usage data such as logs and other aggregated data for the purpose of improving our products and services. We will never collect personally identifiable information or information that is specific to your projects or accounts.

Current Data Collected

At this time, Server 3.0 does not include the data collection service. The service will be included in a future release. With any release, we will communicate what additional data will be collected.

Security

This document outlines security features built into CircleCI and related integrations.

Overview

Security is our top priority at CircleCI, we are proactive and we act on security issues immediately. Report security issues to security@circleci.com with an encrypted message using our security team's GPG key (ID: 0x4013DDA7, fingerprint: 3CD2 A48F 2071 61C0 B9B7 1AE2 6170 15B8 4013 DDA7).

Encryption

CircleCI uses HTTPS or SSH for all networking in and out of our service, including from the browser to our services application, from the services application to your builder fleet, from our builder fleet to your source control system, and all other points of communication. None of your code or data travels to or from CircleCI without being encrypted, unless you have code in your builds that does so at your discretion. Operators may also choose to bypass our SSL configuration or not use TLS for communicating with underlying systems.

The nature of CircleCI is that our software has access to your code and whatever data that code interacts with. All jobs on CircleCI run in a sandbox (specifically, a Docker container or an ephemeral VM) that stands alone from all other builds and is not accessible from the Internet or from your own network. The build agent pulls code via git over SSH. Your particular test suite or job configurations may call out to external services or integration points within your network, and the response from such calls will be pulled into your jobs and used by your code at your discretion. After a job is complete, the container that ran the job is destroyed and rebuilt. All environment variables are encrypted using Hashicorp Vault. Environment variables are encrypted using AES256-GCM96 and are unavailable to CircleCI employees.

Sandboxing

With CircleCI, you control the resources allocated to run the builds of your code. This will be done through instances of our builder boxes that set up the containers in which your builds will run. By their nature, build containers will pull down source code and run whatever test and deployment scripts are part of the codebase or your configuration. The containers are sandboxed, each created and destroyed for one build only (or one slice of a parallel build), and they are not available from outside themselves. The CircleCI service provides the ability to SSH directly to a particular build container. When accessing a container this way, a user will have complete access to any files or processes being run inside that build container. Only provide CircleCI access to those also trusted with your source code.

Integrations

A few different external services and technology integration points touch CircleCI. The following list explains those integration points.

- **Web Sockets** We use Pusher client libraries for WebSocket communication between the server and the browser. However, for installs we use an internal server called Slanger, so Pusher servers have no access to your instance of CircleCI nor your source control system. This is how we, for instance, update the builds list dynamically, or show the output of a build line-by-line as it occurs. We send build status and lines of your build output through the web socket server (which unless you have configured your installation to run without SSL is done using the same certs over SSL), so it is encrypted in transit.

- **Replicated** We use Replicated to manage the installation wizard, licensing keys, system audit logs, software updates, and other maintenance and systems tasks for CircleCI. Your instance of CircleCI communicates with Replicated servers to send license key information and version information to check for updates. Replicated does not have access to your data or other systems, and we do not send any of your data to Replicated.
- **Source Control Systems** To use CircleCI you will set up a direct connection with your instance of GitHub Enterprise or GitHub.com. When you set up CircleCI, you authorize the system to check out your private repositories. You may revoke this permission at any time through your GitHub application settings page and by removing Circle's Deploy Keys and Service Hooks from your repositories' Admin pages. While CircleCI allows you to selectively build your projects, GitHub's permissions model is "all or nothing" — CircleCI gets permission to access all of a user's repositories or none of them. Your instance of CircleCI will have access to anything hosted in those git repositories and will create webhooks for a variety of events (for example, when code is pushed, when a user is added, etc.) that will call back to CircleCI, triggering one or more git commands that will pull down code to your build fleet.
- **Dependency and Source Caches** Most CircleCI customers use S3 or equivalent cloud-based storage inside their private cloud infrastructure (Amazon VPC, etc) to store their dependency and source caches. These storage servers are subject to the normal security parameters of anything stored on such services, meaning in most cases our customers prevent any outside access.
- **Artifacts** It is common to use S3 or similar hosted storage for artifacts. Assuming these resources are secured per your normal policies, they are as safe from any outside intrusion as any other data you store there.

Audit Logs

The Audit Log feature is only available for CircleCI installed on your servers or private cloud.

CircleCI logs important events in the system for audit and forensic analysis purposes. Audit logs are separate from system logs that track performance and network metrics.

Complete Audit logs may be downloaded as a CSV file from the Audit Log page within the Admin section of the application. Audit log fields with nested data contain JSON blobs. Please note the audit log download can take a very long time to start; we recommend clicking the Download button once and leaving it to run.

Note: In some situations, the internal machinery may generate duplicate events in the audit logs. The `id` field of the downloaded logs is unique per event and can be used to identify duplicate entries.

Audit Log Events

The following are the system events that are logged. See `action` in the Field section below for the definition and format.

- context.create
- context.delete
- context.env_var.delete
- context.env_var.store
- project.env_var.create
- project.env_var.delete

- project.settings.update
- user.create
- user.logged_in
- user.logged_out
- workflow.job.approve
- workflow.job.finish
- workflow.job.scheduled
- workflow.job.start

Audit Log Fields

- **action:** The action taken that created the event. The format is ASCII lowercase words separated by dots, with the entity acted upon first and the action taken last. In some cases entities are nested, for example, `workflow.job.start`.
- **actor:** The actor who performed this event. In most cases, this will be a CircleCI user. This data is a JSON blob that will always contain `id` and and `type` and will likely contain `name`.
- **target:** The entity instance acted upon for this event, for example, a project, an org, an account, or a build. This data is a JSON blob that will always contain `id` and and `type` and will likely contain `name`.
- **payload:** A JSON blob of action-specific information. The schema of the payload is expected to be consistent for all events with the same `action` and `version`.
- **occurred_at:** When the event occurred in UTC expressed in ISO-8601 format with up to nine digits of fractional precision, for example '2017-12-21T13:50:54.474Z'.
- **metadata:** A set of key/value pairs that can be attached to any event. All keys and values are strings. This can be used to add additional information to certain types of events.
- **id:** A UUID that uniquely identifies this event. This is intended to allow consumers of events to identify duplicate deliveries.
- **version:** Version of the event schema. Currently the value will always be 1. Later versions may have different values to accommodate schema changes.
- **scope:** If the target is owned by an Account in the CircleCI domain model, the account field should be filled in with the Account name and ID. This data is a JSON blob that will always contain `id` and `type` and will likely contain `name`.
- **success:** A flag to indicate if the action was successful.
- **request:** If this event was triggered by an external request, this data will be populated and may be used to connect events that originate from the same external request. The format is a JSON blob containing `id` (the unique ID assigned to this request by CircleCI).

Checklist To Using CircleCI Securely as a Customer

If you are getting started with CircleCI, there are some points you can ask your team to consider for security best practices as *users* of CircleCI:

- Minimize the number of secrets (private keys / environment variables) your build needs and rotate secrets regularly.

    - It is important to rotate secrets regularly in your organization, especially as team members come and go.
    - Rotating secrets regularly means your secrets are only active for a certain amount of time, helping to reduce possible risks if keys are compromised.
    - Ensure the secrets you *do* use are of limited scope, with only enough permissions for the purposes of your build. Consider carefully adjudicating the role and permission systems of other platforms you use outside of CircleCI; for example, when using something such as IAM permissions on AWS, or GitHub's Machine User feature.
- Sometimes user misuse of certain tools might accidentally print secrets to stdout which will appear in your logs. Please be aware of:
    - running `env` or `printenv` which will print all your environment variables to stdout.
    - literally printing secrets in your codebase or in your shell with `echo`.
    - programs or debugging tools that print secrets on error.
- Consult your VCS provider's permissions for your organization (if you are in an organization) and try to follow the Principle of Least Privilege.
- Use Restricted Contexts with teams to share environment variables with a select security group. Read through the contexts document to learn more.
- Ensure you audit who has access to SSH keys in your organization.
- Ensure that your team is using Two-Factor Authentication (2FA) with your VCS (Github 2FA, Bitbucket). If a user's GitHub or Bitbucket account is compromised a nefarious actor could push code or potentially steal secrets.
- If your project is open source and public, please make note of whether you want to share your environment variables. On CircleCI, you can change a project's settings to control whether your environment variables can pass on to *forked versions of your repo*. This is **not enabled** by default. You can read more about these settings and open source security in our Open Source Projects Document.

CircleCI Server v3.x Application Lifecycle

CircleCI is committed to supporting four minor versions of the software. This means a minor version will receive patches for up to 12 months. In order to help identify releases and their impact to your installation, we use semantic versioning.

Semantic Versioning

Given a version number, MAJOR.MINOR.PATCH increment, the:

1. MAJOR version when you make incompatible API changes,
2. MINOR version when you add functionality in a backwards-compatible manner, and
3. PATCH version when you make backwards compatible bug fixes.

Additional labels for pre-release and build metadata are available as extensions to the MAJOR.MINOR.PATCH format.

Release Schedule

We release monthly patch fixes for bugs and security concerns. We will have quarterly new feature releases. All releases will be posted to the change log. To stay up to date with the most recent releases, please subscribe to the change log.

CircleCI Server v3.x Troubleshooting and Support

This document describes an initial set of troubleshooting steps to take if you are experiencing problems with your CircleCI Server v3.x installation. If your issue is not addressed below, you can generate a support bundle or contact your CircleCI account team.

Start Admin Console

To restart the Admin Console, run:

```
kubectl kots admin-console -n <YOUR_CIRCLECI_NAMESPACE>
```

Open your browser and access http://localhost:8800 to see the Admin console.

Generate Support Bundle

A support bundle is used by CircleCI engineers to diagnose and fix any issues you are experiencing. They are typically requested when you open a ticket.

To download a support bundle for CircleCI support, select the **Troubleshoot** tab from the Admin Console menu bar, and then click **Analyze CircleCI Server**.

Managing Pods

Verify Pod Readiness and Status

Note: please check the `READY` column as well as `STATUS`. Even if the `STATUS` is `Running`, pods are not ready to serve user requests. Some pods may take some time to become ready.

```
kubectl get pods -n <namespace>
NAME READY STATUS RESTARTS AGE
api-service-5c8f557548-zjbsj 1/1 Running 0 6d20h
audit-log-service-77c478f9d5-5dfzv 1/1 Running 0 6d20h
builds-service-v1-5f8568c7f5-62h8n 1/1 Running 0 6d20h
circleci-mongodb-0 1/1 Running 0 6d20h
circleci-nomad-0 1/1 Running 6 6d20h
…
```

To show only pods with a status besides `Running`, you can use the `--field-selector` option.

```
kubectl get pods --field-selector status.phase!=Running -n <namespace>
NAME READY STATUS RESTARTS AGE
nomad-server 0/1 Error 0 5d22h
```

Verify Pod Settings and Status

To show detailed settings and status of pods:

```
kubectl describe pods <pod-name> -n <namespace>
```

Get Pod Logs

To show logs of pods:

```
kubectl logs <pod-name> -n <namespace>
```

Restart Pods

To restart specific pods, the easiest way is remove the pod. Kubernetes will automatically recreate the pod.

```
kubectl delete pod <pod-name> -n <name-space> --now
```

Debug Queuing Builds

For troubleshooting information on debugging queued builds, see the Server 2.x troubleshooting Guide.

CircleCI Server v3.x Backup and Restore

Overview

Backup and restore is available for server v3.1.0 and up.

While operating and administering CircleCI server, you will consider how to maintain backups and recover your installation, should there be a need to migrate it to another cluster or recover from a critical event. This document outlines recommendations for how to back up and restore your CircleCI server instance data and state.

CircleCI server is administered via KOTS, which uses Velero for backup and restore. The benefit of this approach is that it not only restores your application's data, but it also restores the state of the Kubernetes cluster and its resources at the time of the backup. In this way, we can also restore admin-console configurations and customizations you made to your cluster.

Backup and restore of the CircleCI services is dependent on Velero. If your cluster is lost, you will not be able to restore CircleCI until you have successfully started Velero in the cluster. From there you can recover the CircleCI services.

The setup

Backups of CircleCI server can be created quite easily through KOTS. However, to enable backup support you will need to install and configure Velero on your cluster. The following sections outline the steps needed to install Velero on your cluster.

Prerequisites

- Download and install the Velero CLI for your environment.

AWS Prerequisites

- AWS CLI is installed.

GCP Prerequisites

- `gcloud` and `gsutil` are installed. You can set them up by installing Google Cloud SDK, which includes both, by referring to the documentation.

For more information, see Velero's supported providers documentation.

Below, you will find instructions for creating a server 3.x backup on AWS and GCP.

S3 Compatible Storage Prerequisites

- minio CLI is installed and configured for your storage provider.

Server 3.x backups on AWS

The following steps will assume AWS as your provider and you have met the prerequisites listed above.

These instructions were sourced from the Velero documentation here.

Step 1 - Create an AWS S3 bucket

```
BUCKET=<YOUR_BUCKET>
REGION=<YOUR_REGION>
aws s3api create-bucket \
    --bucket $BUCKET \
    --region $REGION \
    --create-bucket-configuration LocationConstraint=$REGION
```

`us-east-1` does not support a LocationConstraint. If your region is `us-east-1`, omit the bucket configuration.

Step 2 - Setup permissions for Velero

- Create an IAM user

```
aws iam create-user --user-name velero
```

- Attach policies to give user `velero` the necessary permissions:

```
cat > velero-policy.json <<EOF
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeVolumes",
                "ec2:DescribeSnapshots",
                "ec2:CreateTags",
                "ec2:CreateVolume",
                "ec2:CreateSnapshot",
                "ec2:DeleteSnapshot"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
```

```
                "s3:GetObject",
                "s3:DeleteObject",
                "s3:PutObject",
                "s3:AbortMultipartUpload",
                "s3:ListMultipartUploadParts"
            ],
            "Resource": [
                "arn:aws:s3:::${BUCKET}/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::${BUCKET}"
            ]
        }
    ]
}
EOF
```

```
aws iam put-user-policy \
  --user-name velero \
  --policy-name velero \
  --policy-document file://velero-policy.json
```

- Create an access key for user `velero`:

```
aws iam create-access-key --user-name velero
```

The result should look like this:

```
{
  "AccessKey": {
        "UserName": "velero",
        "Status": "Active",
        "CreateDate": "2017-07-31T22:24:41.576Z",
        "SecretAccessKey": <AWS_SECRET_ACCESS_KEY>,
        "AccessKeyId": <AWS_ACCESS_KEY_ID>
  }
}
```

- Create a Velero-specific credentials file (for example, `./credentials-velero`) in your local directory, with the following contents:

```
[default]
aws_access_key_id=<AWS_ACCESS_KEY_ID>
aws_secret_access_key=<AWS_SECRET_ACCESS_KEY>
```

where the `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` placeholders are values returned from the `create-access-key` request in the previous step.

Step 3 - Install and start Velero

- Run the following `velero install` command. This will create a namespace called `velero` and install all the necessary resources to run Velero. Make sure that you pass the correct file name containing the AWS credentials that you have created in Step 2.

KOTS backups require restic to operate. When installing Velero, ensure that you have the `--use-restic` flag set, as shown below:

```
velero install \
    --provider aws \
    --plugins velero/velero-plugin-for-aws:v1.2.0 \
    --bucket $BUCKET \
    --backup-location-config region=$REGION \
    --snapshot-location-config region=$REGION \
    --secret-file ./credentials-velero \
    --use-restic \
    --wait
```

- Once Velero is installed on your cluster, check the new `velero` namespace. You should have a Velero deployment and a restic daemonset, for example:

```
$ kubectl get pods --namespace velero
NAME                      READY   STATUS    RESTARTS   AGE
restic-5vlww              1/1     Running   0          2m
restic-94ptv              1/1     Running   0          2m
restic-ch6m9              1/1     Running   0          2m
restic-mknws              1/1     Running   0          2m
velero-68788b675c-dm2s7   1/1     Running   0          2m
```

As restic is a daemonset, there should be one pod for each node in your Kubernetes cluster.

Server 3.x backups on GCP

The following steps are specific for Google Cloud Platform and it is assumed you have met the prerequisites.

These instructions were sourced from the documentation for the Velero GCP plugin here.

Step 1 - Create a GCP bucket

To reduce the risk of typos, we will set some of the parameters as shell variables. If you are unable to complete all the steps in the same session, do not forget to reset variables as necessary before proceeding. In the step below, for example, we will define a variable for your bucket name. Replace the `<YOUR_BUCKET>` placeholder with the name of the bucket you want to create for your backups.

```
BUCKET=<YOUR_BUCKET>

gsutil mb gs://$BUCKET/
```

Step 2 - Setup permissions for Velero

If your server installation runs within a GKE cluster, ensure that your current IAM user is a cluster admin for this cluster, as RBAC objects need to be created. More information can be found in the GKE documentation.

1. First, we will set a shell variable for your project ID. To do so, first make sure that your `gcloud` CLI points to the correct project by looking at the current configuration:

   ```
   gcloud config list
   ```

2. If the project is correct, set the variable as follows:

   ```
   PROJECT_ID=$(gcloud config get-value project)
   ```

3. Create a service account:

```
gcloud iam service-accounts create velero \
    --display-name "Velero service account"
```

If you run several clusters with Velero, you might want to consider using a more specific name for the Service Account besides `velero`, as suggested in the example above.

4. You can check if the service account has been created successfully by running:

```
gcloud iam service-accounts list
```

5. Next, store the email address for the Service Account in a variable:

```
SERVICE_ACCOUNT_EMAIL=$(gcloud iam service-accounts list \
  --filter="displayName:Velero service account" \
  --format 'value(email)')
```

   Modify the command as needed to match the display name you have chosen for your Service Account.

6. Grant the necessary permissions to the Service Account:

```
ROLE_PERMISSIONS=(
    compute.disks.get
    compute.disks.create
    compute.disks.createSnapshot
    compute.snapshots.get
    compute.snapshots.create
    compute.snapshots.useReadOnly
    compute.snapshots.delete
    compute.zones.get
)

gcloud iam roles create velero.server \
    --project $PROJECT_ID \
    --title "Velero Server" \
    --permissions "$(IFS=","; echo "${ROLE_PERMISSIONS[*]}")"

gcloud projects add-iam-policy-binding $PROJECT_ID \
    --member serviceAccount:$SERVICE_ACCOUNT_EMAIL \
    --role projects/$PROJECT_ID/roles/velero.server

gsutil iam ch serviceAccount:$SERVICE_ACCOUNT_EMAIL:objectAdmin gs://${BUCKET}
```

Now, you need to ensure that Velero can use this Service Account.

Option 1: JSON key file

You can simply pass a JSON credentials file to Velero to authorize it to perform actions as the Service Account. To do this, we first need to create a key:

```
gcloud iam service-accounts keys create credentials-velero \
    --iam-account $SERVICE_ACCOUNT_EMAIL
```

After running this command, you should see a file named `credentials-velero` in your local working directory.

Option 2: Workload Identities

If you are already using Workload Identities in your cluster, you can bind the GCP Service Account you just created to Velero's Kubernetes service account. In this case, the GCP Service Account will need the `iam.serviceAccounts.signBlob` role in addition to the permissions already specified above.

Step 3 - Install and start Velero

- Run one of the following `velero install` commands, depending on how you authorized the service account. This will create a namespace called `velero` and install all the necessary resources to run Velero.

kots backups require restic to operate. When installing Velero, ensure that you have the `--use-restic` flag set.

If using a JSON key file

```
velero install \
    --provider gcp \
    --plugins velero/velero-plugin-for-gcp:v1.2.0 \
    --bucket $BUCKET \
    --secret-file ./credentials-velero \
    --use-restic \
    --wait
```

If using Workload Identities

```
velero install \
    --provider gcp \
    --plugins velero/velero-plugin-for-gcp:v1.2.0 \
    --bucket $BUCKET \
    --no-secret \
    --sa-annotations iam.gke.io/gcp-service-account=$SERVICE_ACCOUNT_EMAIL \
    --backup-location-config serviceAccount=$SERVICE_ACCOUNT_EMAIL \
    --use-restic \
    --wait
```

For more options on customizing your installation, refer to the Velero documentation.

- Once Velero is installed on your cluster, check the new `velero` namespace. You should have a Velero deployment and a restic daemonset, for example:

```
$ kubectl get pods --namespace velero
NAME                      READY   STATUS    RESTARTS   AGE
restic-5vlww              1/1     Running   0          2m
restic-94ptv              1/1     Running   0          2m
restic-ch6m9              1/1     Running   0          2m
restic-mknws              1/1     Running   0          2m
velero-68788b675c-dm2s7   1/1     Running   0          2m
```

As restic is a daemonset, there should be one pod for each node in your Kubernetes cluster.

Server 3.x backups with S3 Compatible Storage

The following steps will assume you arre using S3-compatible object storage, but not necessarily AWS S3, for your backups. It is also assumed you have met the prerequisites.

These instructions were sourced from the Velero documentation here.

Step 1 - Configure `mc` client

To start, configure `mc` to connect to your storage provider:

```
# Alias can be any name as long as you use the same value in subsequent commands
export ALIAS=my-provider
mc alias set $ALIAS <YOUR_MINIO_ENDPOINT> <YOUR_MINIO_ACCESS_KEY_ID> <YOUR_MINIO_SECRET_ACCESS_KEY>
```

You can verify your client is correctly configured by running `mc ls my-provider` and you should see the buckets in your provider enumerated in the output.

Step 2 - Create a bucket

Create a bucket for your backups. It is important that a new bucket is used, as Velero cannot use an existing bucket that already contains other content.

```
mc mb ${ALIAS}/<YOUR_BUCKET>
```

Set 3 - Create a user and policy

Next, create a user and policy for Velero to access your bucket.

In the following snippet `<YOUR_MINIO_ACCESS_KEY_ID>` and `<YOUR_MINIO_SECRET_ACCESS_KEY>` refer to the credentials used by Velero to access MinIO.

```
# Create user
mc admin user add $ALIAS <YOUR_MINIO_ACCESS_KEY_ID> <YOUR_MINIO_SECRET_ACCESS_KEY>

# Create policy
cat > velero-policy.json << EOF
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:*"
      ],
      "Resource": [
        "arn:aws:s3:::<YOUR_BUCKET>",
        "arn:aws:s3:::<YOUR_BUCKET>/*"
      ]
    }
  ]
}
EOF

mc admin policy add $ALIAS velero-policy velero-policy.json

# Bind user to policy
mc admin policy set $ALIAS velero-policy user=<YOUR_VELERO_ACCESS_KEY_ID>
```

Finally, we add our new user's credentials to a file (`./credentials-velero` in this example) with the following contents:

```
[default]
aws_access_key_id=<YOUR_VELERO_ACCESS_KEY_ID>
aws_secret_access_key=<YOUR_VELERO_SECRET_ACCESS_KEY>
```

Step 4 - Install and start Velero

Run the following `velero install` command. This will create a namespace called `velero` and install all the necessary resources to run Velero.

KOTS backups require restic to operate. When installing Velero, ensure that you have the `--use-restic` flag set, as shown below:

```
velero install --provider aws \
  --plugins velero/velero-plugin-for-aws:v1.2.0 \
  --bucket <YOUR_BUCKET> \
  --secret-file ./credentials-velero \
  --use-volume-snapshots=false \
  --use-restic \
  --backup-location-config region=minio,s3ForcePathStyle="true",s3Url=<YOUR_ENDPOINT> \
  --wait
```

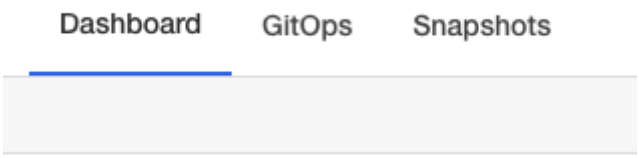
Once Velero is installed on your cluster, check the new `velero` namespace. You should have a Velero deployment and a restic daemonset, for example:

```
$ kubectl get pods --namespace velero
NAME                      READY   STATUS    RESTARTS   AGE
restic-5vlww              1/1     Running   0          2m
restic-94ptv              1/1     Running   0          2m
restic-ch6m9              1/1     Running   0          2m
restic-mknws              1/1     Running   0          2m
velero-68788b675c-dm2s7   1/1     Running   0          2m
```

As restic is a daemonset, there should be one pod for each node in your Kubernetes cluster.

Creating backups

Now that Velero is installed on your cluster, you should see the **Snapshots** option in the navbar of the management console.



If you see this option, you are ready to create your first backup. If you do not see this option, please refer to the troubleshooting section.

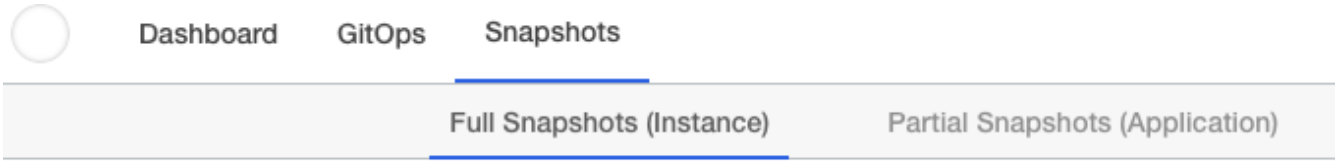
Option 1 - Create a backup with KOTS CLI

To create the backup, run:

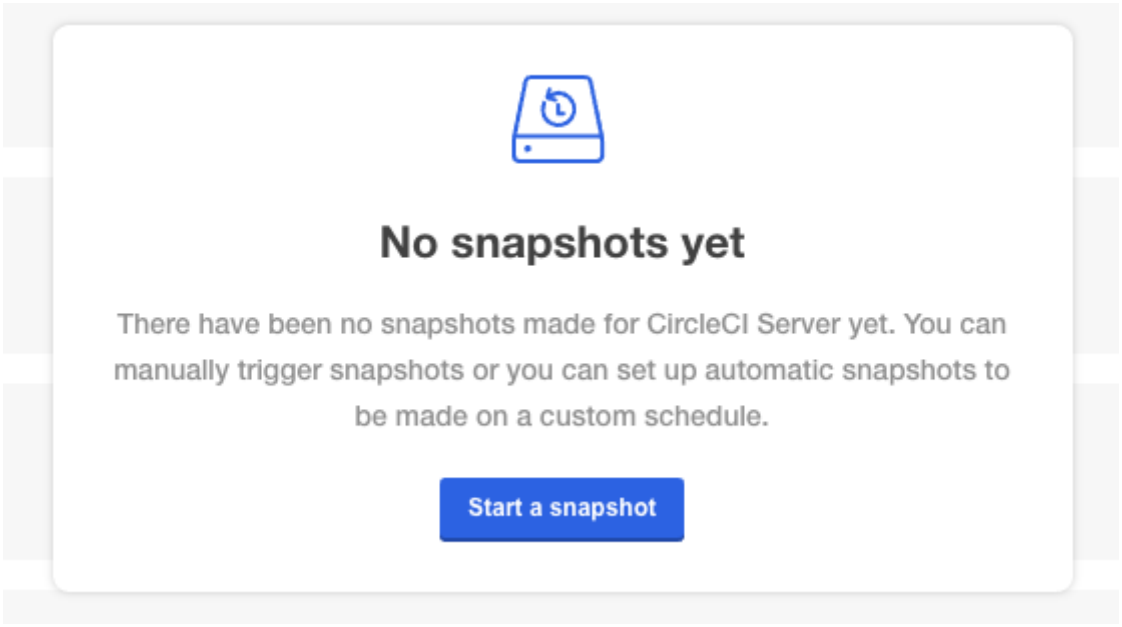
```
kubectl kots backup --namespace <your namespace>
```

Option 2 - Create a backup with KOTS Admin Console

Select **Snapshots** from the navbar. The default selection should be **Full Snapshots**, which is recommended.



Click the **Start a snapshot** button.



Restoring backups

Option 1 - Restore a backup from a snapshot

To restore from a backup stored in your S3 compatible storage, you will need to ensure Velero is installed on your Kubernetes cluster and that Velero has access to the storage bucket containing the backups. When using EKS, restoring CircleCI server requires that an instance of CircleCI server is installed beforehand. When using GKE or other platforms, a cluster with just velero installed may work.

If this is a new cluster or if you need to reinstall Velero, the installation should be done with the same credentials generated above.

Option 2 - Restore a backup using the KOTS CLI

To restore a backup using the KOTS CLI, run the following command to get a list of backups:

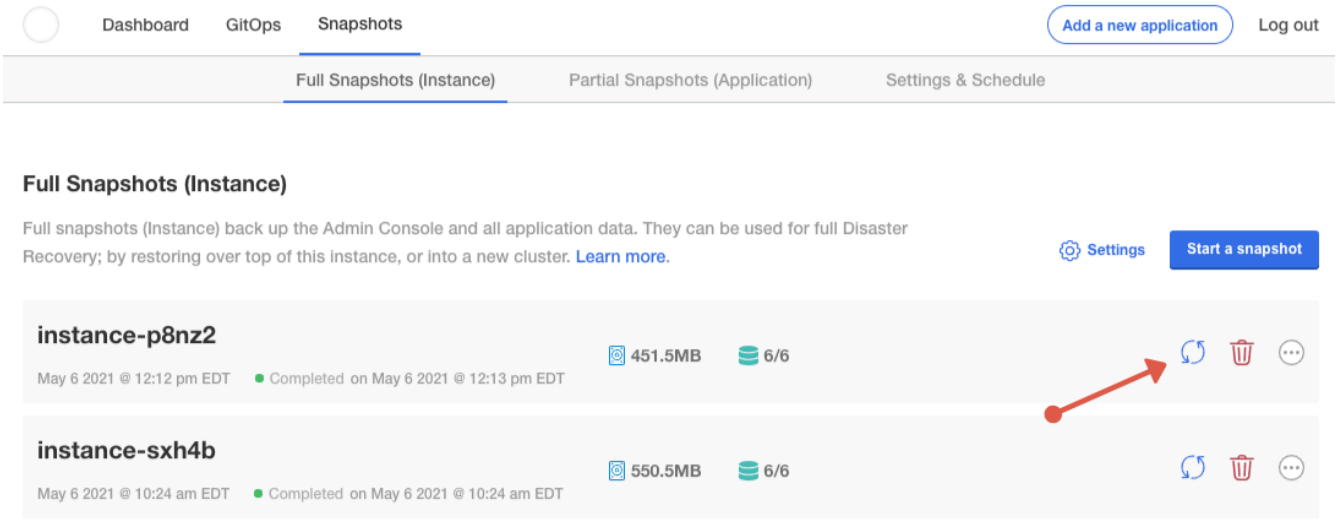
```
kubectl kots get backups
```

Using a backup name from the previous command, run the following to start the restore process:

```
kubectl kots restore --from-backup <backup-instance-id>
```

Option 3 - Restore a backup using the KOTS Admin Console UI

As with backups, navigate to **Snapshots** in the KOTS Admin Console. Now you should see a list of all your backups, each with a restore icon. Choose the backup you wish to use and select restore.



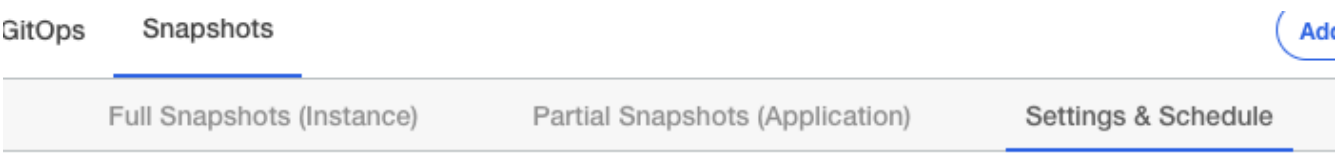
The restore will create new load balancers for CircleCI's services. You will need to either update your DNS records or the hostname configurations in KOTS Admin Console as a result. You may also need to consider updating the `nomad server endpoint` provided to your Nomad clients.

If you are using pre-existing Nomad clients, you will need to restart them before they will connect to the nomad-server cluster.

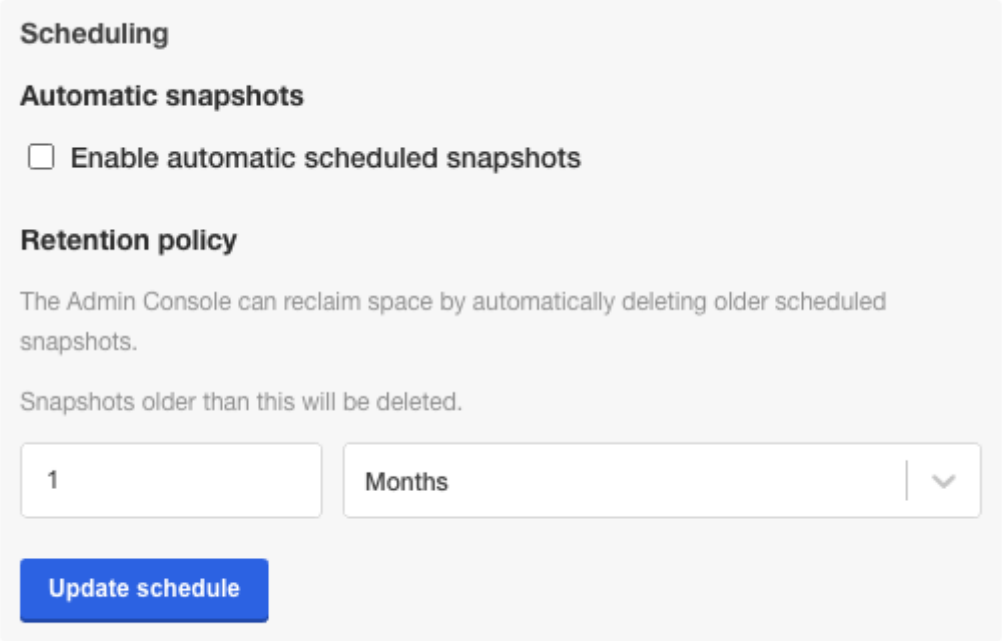
It should take roughly 10-15 mins for CircleCI server to be restored and operational.

Optional - Scheduling backups with kots

To schedule regular backups, select **Snapshots**, and then **Settings & Schedule** from the KOTS Admin Console.



And here you can find configurations related to your snapshots, including scheduling.



Troubleshooting Backups and Restoration

Snapshots are not available in KOTS Admin Console

If your KOTS Admin Console does not display the snapshot option, you may try the following:

- Confirm that your version of KOTS supports snapshots. At this time, we recommend v1.40.0 or above:

```
$ kubectl kots version
Replicated KOTS 1.40.0
```

- Check that Velero is deployed and running correctly. You may check the Velero logs with the command below.

```
$ kubectl logs deployment/velero --namespace velero
```

You may need to reinstall Velero as a result.

- Confirm that snapshots are available on your license. You may reach out to our Customer Support Team for confirmation.

Errors occur during backup or restore process

If you experience an error during backup or restore processes, the first place to look would be the Velero logs. Using the command above, you may find 4XX errors, which would likely be caused by issues with your storage bucket access.

- Confirm that your bucket exists and is in the region you expect.
- Confirm that the credentials provided to Velero can be used to access the bucket.
- You may need to run the command to install Velero again, this time with updated bucket information.

You may also check the status of pods in the `velero` namespace:

```
$ kubectl get pods --namespace velero
NAME                      READY   STATUS    RESTARTS   AGE
restic-5vlww              1/1     Pending   0          10m
restic-94ptv              1/1     Running   0          10m
restic-ch6m9              1/1     Pending   0          10m
restic-mknws              1/1     Running   0          10m
velero-68788b675c-dm2s7   1/1     Running   0          10m
```

In the above example, some restic pods are pending, which means they are waiting for a node to have available CPU or memory resources. In this case, you may need to scale your nodes to accommodate restic.