OPERATIONS GUIDE

A guide for administrators of CircleCI Server v4.0.0 on AWS or GCP

docs@circleci.com

Version 4.0.0, 08/18/2022: FINAL

Operator overview

This guide contains information for CircleCI server operators, or those responsible for ensuring CircleCI server is running properly through maintenance and monitoring. Before reading this operator guide, ensure you have read the CircleCI Server v4.x Overview.

CircleCI server schedules CI/CD jobs using the Nomad scheduler. The Nomad Server (control plane) runs inside the Kubernetes cluster, while 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, and also in dedicated virtual machines. These VM jobs are controlled by the Nomad clients, therefore the Nomad clients must be able to access the VMs 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 need access to object storage.

Execution environment

CircleCI server v4.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.

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, allowing 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 Using Metrics 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 limits 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 for 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 guide.

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.

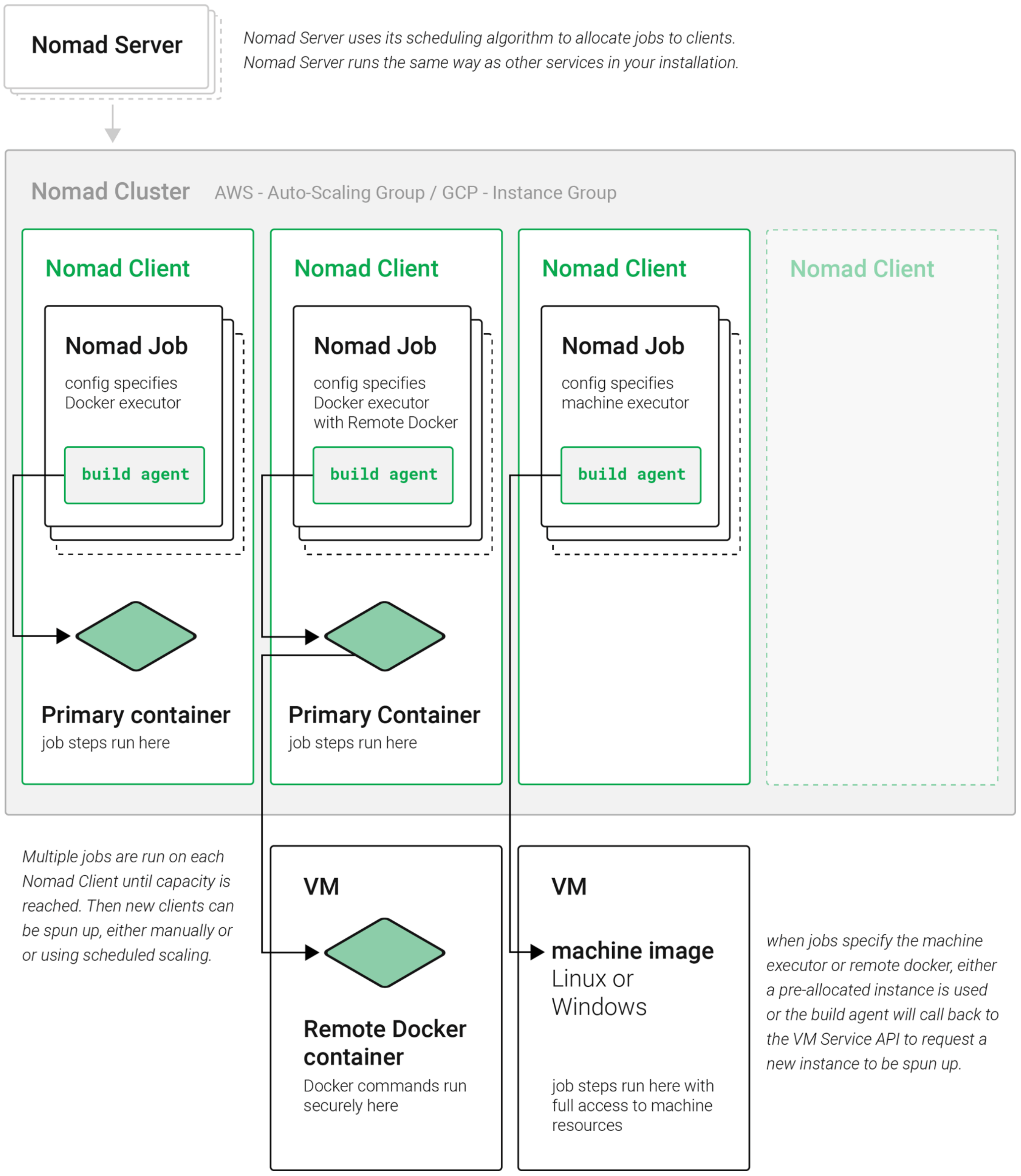
The following table describes the ports used on machines running GitHub to communicate with the services and Nomad client instances.

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

Introduction to Nomad cluster operation

CircleCI uses Nomad as the primary job scheduler. This section provides a basic introduction to Nomad for understanding how to operate the Nomad Cluster in your CircleCI installation.

Basic terminology and architecture



Nomad Server
Nomad Server uses its scheduling algorithm to allocate jobs to clients.
Nomad Server runs the same way as other services in your installation.
Nomad Cluster
AWS - Auto-Scaling Group / GCP - Instance Group
Nomad Client
Nomad Job
config specifies Docker executor
build agent
Primary container
job steps run here
Nomad Client
Nomad Job
config specifies Docker executor with Remote Docker
build agent
Primary Container
job steps run here
Nomad Client
Nomad Job
config specifies machine executor
build agent
Nomad Client
Multiple jobs are run on each Nomad Client until capacity is reached. Then new clients can be spun up, either manually or or using scheduled scaling.
VM
Remote Docker container
Docker commands run securely here
VM
machine image
Linux or Windows
job steps run here with full access to machine resources
when jobs specify the machine executor or remote docker, either a pre-allocated instance is used or the build agent will call back to the VM Service API to request a new instance to be spun up.

Figure 1. Nomad Cluster Management

- **Nomad server:** Nomad servers are the brains of the cluster. They receive and allocate jobs to Nomad clients. In CircleCI server, a Nomad server runs as a service in your Kubernetes cluster.
- **Nomad client:** Nomad clients execute the jobs they are allocated by Nomad servers. Usually a Nomad client runs on a dedicated machine (often a VM) to take full advantage of machine power. You can have multiple Nomad clients to form a cluster and the Nomad server allocates jobs to the cluster with its scheduling algorithm.
- **Nomad jobs:** A Nomad job is a specification, provided by a user, that declares a workload for Nomad. A Nomad job corresponds to an execution of a CircleCI job. If the job uses parallelism, for example `parallelism: 10`, then Nomad runs 10 jobs.
- **Build agent:** Build agent is a Go program written by CircleCI that executes steps in a job and reports the results. Build agent is executed as the main process inside a Nomad job.

Basic operations

The following section is a basic guide to operating a Nomad cluster in your installation.

The `nomad` CLI is installed in the Nomad pod. It is preconfigured to talk to the Nomad cluster, so it is possible to use `kubectl` along with the `nomad` command to run the commands in this section.

Checking the jobs status

The get a list of statuses for all jobs in your cluster, run the following command:

```
kubectl exec -it nomad-server-pod-ID -- nomad status
```

The `Status` is the most important field in the output, with the following status type definitions:

- `running`: Nomad has started executing the job. This typically means your job in CircleCI is started.
- `pending`: There are not enough resources available to execute the job inside the cluster.
- `dead`: Nomad has finished executing the job. The status becomes `dead` regardless of whether the corresponding CircleCI job/build succeeds or fails.

Checking the cluster status

To get a list of your Nomad clients, run the following command:

```
kubectl exec -it nomad-server-pod-ID -- nomad node-status
```

`nomad node-status` reports both Nomad clients that are currently serving (status `active`) and Nomad clients that were taken out of the cluster (status `down`). Therefore, you need to count the number of `active` Nomad clients to know the current capacity of your cluster.

To get more information about a specific client, run the following command from that client:

```
kubectl exec -it nomad-server-pod-ID -- nomad node-status -self
```

This gives information such as how many jobs are running on the client and the resource utilization of the client.

Checking logs

A Nomad job corresponds to an execution of a CircleCI job. Therefore, Nomad job logs can sometimes help to understand the status of a CircleCI job if there is a problem. To get logs for a specific job, run the following command:

```
kubectl exec -it nomad-server-pod-ID -- nomad logs -job -stderr <nomad-job-id>
```

Be sure to specify the `-stderr` flag, as this is where most Build Agent logs appear.

While the `nomad logs -job` command is useful, it is not always accurate because the `-job` flag uses a random allocation of the specified job. The term `allocation` is a smaller unit in Nomad Job, which is beyond the scope of this document. To learn more, please see the official document.

Complete the following steps to get logs from the allocation of the specified job:

1. Get the job ID with `nomad status` command.
2. Get the allocation ID of the job with `nomad status <job-id>` command.
3. Get the logs from the allocation with `nomad logs -stderr <allocation-id>`

Shutting down a Nomad client

When you want to shut down a Nomad client, you must first set the client to `drain` mode. In `drain` mode, the client will finish any jobs that have already been allocated but will not be allocated any new jobs.

1. To drain a client, log in to the client and set the client to drain mode with `node-drain` command as follows:

   ```
   nomad node-drain -self -enable
   ```

2. Then, make sure the client is in drain mode using the `node-status` command:

   ```
   nomad node-status -self
   ```

Alternatively, you can drain a remote node with the following command, substituting the node ID:

```
nomad node-drain -enable -yes <node-id>
```

Scaling down the client cluster

To set up a mechanism for clients to shutdown, first enter `drain` mode, then wait for all jobs to be finished before terminating the client. You can also configure an ASG Lifecycle Hook that triggers a script for scaling down instances.

The script should use the commands in the section above to do the following:

1. Put the instance in drain mode.
2. Monitor running jobs on the instance and wait for them to finish.
3. Terminate the instance.

Using metrics

Metrics such as CPU or memory usage and internal metrics can help you manage and debug your server installation in many ways, including:

- 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 debugging issues with your installation.

Telegraf

Most services running on server 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 supported by Telegraf through 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:

In your custom `values.yaml` for your CircleCI server install you may already have a section for Telegraf as below. If not you can add it now.

```
...
telegraf:
  args:
    - --config
    - /etc/telegraf/telegraf.d/telegraf_custom.conf
  config:
    custom_config_file: |
      [agent]
        collection_jitter = "0s"
        debug = false
        flush_interval = "10s"
        flush_jitter = "0s"
        hostname = "$HOSTNAME"
        interval = "30s"
        logfile = ""
        metric_batch_size = 1000
        metric_buffer_limit = 10000
        omit_hostname = true
        precision = ""
        quiet = false
        round_interval = true

      [[processors.enum]]
        [[processors.enum.mapping]]
          dest = "status_code"
          field = "status"
          [processors.enum.mapping.value_mappings]
              critical = 3
              healthy = 1
              problem = 2

      [[inputs.statsd]]
        datadog_extensions = true
        metric_separator = "."
        percentile_limit = 1000
        percentiles = [
          50,
          95,
          99
        ]
        service_address = ":8125"
      [[inputs.internal]]
        collect_memstats = false

      [[outputs.file]]
        files = ["stdout"]

      [[outputs.prometheus_client]]
        listen = ":9273"
        path = "/metrics"
...
```

In the `telgraf.config.custom_config_file` block of your helm `values.yaml`, add the as seen below to the bottom of the config block, replacing `"my-secret-key"` with your datadog API key.

```
telegraf:
  config:
    custom_config_file: |
      ...
      [[outputs.datadog]]
        ## Replace "my-secret-key" with Datadog API key
        apikey = "my-secret-key"
      ...
```

For more options, see the Telegraf docs.

Finally, you will need to apply the changes via helm update.

```
helm upgrade circleci-server oci://cciserver.azurecr.io/circleci-server -n <namespace> --version
<version> -f <path-to-values.yaml> --username $USERNAME --password $PASSWORD
```

Note: If your Telegraf pod does not restart after your helm upgrade, then the changes will not take effect. You may gracefully restart your Telegraf instance with command below. This will restart Telegraf with zero downtime.

```
kubectl rollout restart deploy telegraf -n <namespace>
```

Managing 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.

Reactivate a new account

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.

Reactivate an account

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.

Managing Orbs

This section describes how to manage orbs for an installation of server v4.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.

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`, 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 <namespace><orb-name>@<orb-version> --host <your-server-installation-
domain> --token <your-api-token>
```

`<orb-name>` and `<orb-version>` are optional. You can choose to only specify a namespace, in which case the most recent versions of all orbs in the namespace will be imported.

Fetch a public orb's updates

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

```
circleci admin import-orb <namespace><orb-name>@<orb-version> --host <your-server-installation-
domain> --token <your-api-token>
```

`<orb-name>` and `<orb-version>` are optional. You can choose to only specify a namespace, in which case the most recent versions of all orbs in the namespace will be updated.

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>
```

Manage virtual machines with VM Service

VM service controls how `machine` executor and Remote Docker jobs are run.

This section describes the available configuration options for VM service. Refer to the default `values.yaml` file for details on how to pre-scale virtual machines.

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 installation guide for AWS and GCP.

If Docker Layer Caching (DLC) is used, VM Service instances need to be spun up on demand. For this to 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.

VM provider

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

AWS

You will need to add a section to your `values.yaml` file to configure VM Service to work with AWS EC2.

Authentication

One of the following options is required:

- Either, select "IAM Keys" and provide:
  - **Access Key ID** (required): Access Key ID for EC2 access.
  - **Secret Key** (required): Secret Key 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.

- Or, select "IAM role" and provide:
  - **Role ARN** (required):

    Role ARN for Service Accounts (Amazon Resource Name) for EC2 access.

```
vm_service:
  providers:
    ec2:
      enabled: true
      region: <region>
      # Subnets must be in the same availability zone
      subnets:
      - <subnet-id>
      securityGroupId: <security-group-id>

      # Authenticate with IAM access keys
      accessKey: <access-key>
      secretKey: <secret-key>
      # or IRSA (IAM roles for service accounts)
      irsaRole: <role-arn>
```

Default AWS AMI list

The default AMIs for server v4.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"
```

GCP

You will need to add a section to your `values.yaml` file to configure VM Service to work with Google Cloud Platform (GCP).

Authentication

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.

Choose one of the following options:

- If you choose to use **GCP IAM Workload Identity**, use the VM service account email address (`service-account-name@project-id`.iam.gserviceaccount.com) which you created here in point 2 and 3 for the `workloadIdentity` field below.
- If you choose to use a **GCP Service Account JSON file**, use the contents of your service account JSON file for `service-account` below.

```
vm_service:
  enabled: true
  replicas: 1
  providers:
    gcp:
      enabled: false
      project_id: <project-id>
      network_tags:
      - circleci-vm
      - <your-network>
      zone: <zone>
      network: <network>
      subnetwork: <subnetwork>

      service_account: <service-account-json>
      # OR
      workloadIdentity: ""  # Leave blank if using JSON keys of service account else service account
email address
```

Configuring External Services

This page describes how to configure external services for use with a CircleCI server v4.x installation.

PostgreSQL

If you are using your own PostgreSQL instance, it needs to be version 12.1 or higher.

If you choose to use an external PostgreSQL instance, add the following to your `values.yaml` file.

```
postgresql:
  internal: false
  postgresqlHost: <domain> # The domain or IP address of your PostgreSQL instance
  postgresqlPort: <port> # The port of your PostgreSQL instance
```

Create the secret and then add the following values to `values.yaml`:

```
kubectl create secret generic postgresql \
  --from-literal=postgres-password=<postgres-password>
```

```
postgresql:
  ...
  auth:
    username: <username>
    existingSecret: postgresql
```

Add the following to the `values.yaml` file. CircleCI will create the secret automatically:

```
postgresql:
  ...
  auth:
    username: <username> # A user with the appropriate privileges to access your PostgreSQL
instance.
    password: <password> # The password of the user account used to access your PostgreSQL instance.
```

Best practices for 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 higher.

If you choose to use an external MongoDB instance, add the following to your `values.yaml` file.

```
mongodb:
  internal: false
  hosts: <hostname:port> # this can be a comma-separated list of multiple hosts for sharded
instances
  ssl: <ssl-enabled>
  # If using an SSL connection with custom CA or self-signed certs, set this
  # to true
  tlsInsecure: false
  # Any other options you'd like to append to the MongoDB connection string.
  # Format as query string (key=value pairs, separated by &, special characters
  # need to be URL encoded)
  options: <additional-options>
  auth:
    database: <authentication-source-database
    mechanism: SCRAM-SHA-1
```

Create the secret and then add the following values to `values.yaml`:

```
kubectl create secret generic mongodb \
--from-literal=mongodb-root-password=<root-password> \
--from-literal=mongodb-password=dontmatter
```

```
mongodb:
  ...
  auth:
    ...
    username: <username>
    existingSecret: mongodb
```

Add the following to the `values.yaml` file. CircleCI will create the secret automatically:

```
mongodb:
  ...
  auth:
    ...
    username: <username>
    rootPassword: <root-password>
    password: <password>
```

Expanding Internal Database Volumes

Overview

If you have chosen to deploy either of the CircleCI databases (MongoDB or PostgreSQL) within the cluster, rather than externally provisioning these databases, 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 PostgreSQL and MongoDB. You will confirm the size of the claims and the disk space made available to your databases before and after this operation.

As a precaution, it is always a good idea 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 via the helm-value file. You can confirm the size of your persistent volume claim capacity using the command: `kubectl get pvc <pvc-name>`.

For PostgreSQL:

```
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
```

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

For PostgreSQL, the directory is `/bitnami/postgresql`. You 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 you can see, your default storage class does not allow volume expansion. However, you 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 you may proceed to expanding your volumes.

Step 2 - Delete the database's stateful set

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

For PostgreSQL:

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

For MongoDB:

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

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

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

For PostgreSQL:

```
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 helm-value file with the new PVC size

Now you need to upgrade the server installation by modifying the PVC size in the helm-value file to persist your changes. In the helm-value file, you will update the values for your PVC size to 10Gi as shown below.

- **PostgreSQL**

```
postgresql:
  persistence:
    size: 10Gi
```

- **MongoDB**

```
mongodb:
  persistence:
    size: 10Gi
```

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

```
helm upgrade <release-name> -n <namespace> -f < helm-value-file> <chart-dictectory>
```

Step 5 - Validate new volume size

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

For PostgreSQL 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 you can see, the size of your 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 PostgreSQL:

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

For MongoDB:

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

Success looks like this:

```
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:

```
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

After following these steps, if you find that 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 minutes while the databases restart. You can use the commands below to restart your databases.

For PostgreSQL:

```
kubectl rollout restart sts postgresql
```

For MongoDB:

```
kubectl rollout restart sts mongodb
```

Managing load balancers

CircleCI server uses a load balancer to manage network traffic entering and leaving the Kubernetes cluster.

The load balancer manages all traffic coming into the application. The 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, depending on your 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 in the CircleCI app UI under **Admin Settings** > **System Settings** > **Override webhook host URL**.

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

In your values.yaml override file, set the following parameter to true. The parameter is false (public) by default.

```
nginx:
    private_load_balancers: false
```

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 requests a new load balancer with the new configuration.

User 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.

Managing 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. Add the following to your `values.yaml` file:

```
serveUnsafeArtifacts: true
```

Usage data collection

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

Current data collected

Currently, Server v4.x 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.

CircleCI server security features

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** CircleCI uses Pusher client libraries for WebSocket communication between the server and the browser. However, for installs CircleCI uses an internal server called Slanger, so Pusher servers have no access to your instance of CircleCI, nor your source control system. This is how CircleCI, for instance, updates the builds list dynamically, or show the output of a build line-by-line as it occurs. CircleCI sends 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.

- **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
- context.secrets.accessed
- 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.

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. We use semantic versioning to help identify releases and their impact on your installation.

Semantic versioning

Given a version number, MAJOR.MINOR.PATCH increment, use 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.

Troubleshooting and support

This document describes an initial set of troubleshooting steps to take if you are experiencing problems with your CircleCI server v4.x installation. If your issue is not addressed below, you can generate a support bundle or contact your CircleCI account team.

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 generate a support bundle, follow the steps below.

Prerequisites

1. First, make sure circleci-server is deployed and you have access to the cluster/namespace through kubectl.

```
# To check if you have access to cluster/namespace
kubectl -n <namespace> get pods
```

1. Next, install krew.
2. Install support-bundle (kubectl plugin) to your local development machine.

```
# To install support-bundle plugin
kubectl krew install support-bundle
```

Generating support bundle

When ready, run the support bundle from the current directory and wait for it to finish.

```
kubectl support-bundle https://raw.githubusercontent.com/CircleCI-Public/server-
scripts/main/support/support-bundle.yaml
```

Managing pods

Verify pod readiness and status

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, use the following command:

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

Get pod logs

To show logs of pods, use the following command:

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

Restart pods

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

Backup and restore

Overview

While operating and administering CircleCI server, you will need to 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. For setup instructions see Phase 4 – Post installation.

CircleCI recommends 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. Setup and install of Velero is covered in the server v4.x installation guide.

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.

Creating backups

```
K8S_NS=$(helm list -o yaml  | yq '.[].namespace')
CHART=$(helm list -o yaml  | yq '.[].chart' )
REV=$(helm list -o yaml  | yq '.[].revision')
RANDOM_STR=$(cat /dev/urandom | env LC_ALL=C tr -dc 'a-z0-9' | head -c 8)

velero backup create "${K8S_NS}-${RANDOM_STR}" --include-namespaces "${K8S_NS}" --labels "chart--
rev=${CHART}--${REV}"
```

Restoring backups

```
# List all existing backups
velero backup get --show-labels

# Restore the specific backup
velero restore create --include-namespaces <circleci-namespace> --from-backup <backup-name>
```

Scheduling backups

See Velero's documentation on creating scheduled backups.

Troubleshooting

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 below, 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.

CircleCI Server v4.x FAQ

Does Server 4.0 have a data retention policy?

We do not currently support a defined data retention policy. Data is stored indefinitely on server.

What control is granted over Nomad certificates?

Full control of the certificates, all the way down to mTLS for Nomad.

Is it possible to change or disable the polling time which checks for health status?

No, this is not customizable.