OPERATIONS GUIDE

A guide for administrators of CircleCI Server v3.0.1 on AWS or GCP

docs@circleci.com

Version 3.0.1, 05/20/2021: 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, via 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 output 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.

Build 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 increase the number of Nomad Client machines as needed to balance the load. For more on tracking metrics see the Metrics and Monitoring section.

Each machine reserves two vCPUs and 4GB of memory for coordinating builds. The remaining processors and memory create the containers for running jobs. Larger machines are able to run more containers and are limited by the number of available cores after two are reserved for coordination.

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 back-end 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

Backup and Restore is not yet available on the current version of Server 3.x; however, it will be available for preview before the next release.

Metrics such as CPU or memory usage, number of executed builds and internal metrics are useful in:

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

Metrics Collection

Grafana

Server ships with Grafana which has become the world's most popular technology used to compose observability dashboards. We have integrated with Prometheus & Loki Logs so you can create dashboards, set up alerts, search for errors and more.

Loki

Loki is a horizontally-scalable, highly-available, log aggregation system inspired by Prometheus. It is designed to be very cost effective and easy to operate. It does not index the contents of the logs, but rather a set of labels for each log stream.

Prometheus

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

Tips and Troubleshooting

Pod Logs

You can check that services/pods are reporting metrics correctly by checking if they are being reported to stdout. To do this, examine the logs of the `circleci-telegraf` pod using `kubectl logs` or a tool like stern.

To view logs for Telegraf, run the following:

- `kubectl get pods` to get a list of services
- `kubectl logs -f circleci-telegraf-<hash>`, substituting the hash for your installation.

While monitoring the current log stream, perform some actions with your server installation (e.g. logging in/out or running a workflow). These activities should be logged, showing that metrics are being reported. Most metrics you see logged will be from the frontend pod. However, when you run workflows, you should also see metrics reported by the dispatcher, `legacy-dispatcher`, `output-processor` and `workflows-conductor`, as well as metrics concerning cpu, memory and disk stats.

You may also check the logs by running `kubectl logs circleci-telegraf-<hash> -n <namespace> -f` to confirm that your output provider (e.g. influx) is listed in the configured outputs.

CircleCI Server v3.x User Accounts

This section provides information to help operators manage user accounts. For an overview of user accounts, view 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. In order 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 so 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 server seats available.

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.

Using the CircleCI CLI

This section covers how to use the CircleCI CLI to list or delete users.

List Users

To view a full list of users for your CircleCI Server installation, first SSH into your Services machine, and run:

```
circleci dev-console
(circle.model.user/where { :$and [{:sign_in_count {:$gte 0}}, {:login {:$ne nil}}]} :only [:login])
```

Delete User

If you need to remove a user from your installation of CircleCI Server, you will need to SSH into the services machine first and then delete using the following command, substituting the user's GitHub username:

```
circleci dev-console
(circle.http.api.admin-commands.user/delete-by-login-vcs-type! "github-username-of-user" :github)
```

CircleCI Server v3.x Orbs

This section describes Orbs and how to manage them. Server installations include their own local orb registry. This registry is private to the server installation. All orbs referenced in configs reference 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 copied prior, and registering your companies private orbs if they exist.

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 https://circleci.com/docs/2.0/managing-api-tokens/. Providing a local repository location using the `--host` option allows you to access your local server orbs vs the 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:

```
circleci orb list
```

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

```
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:

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

For more Orb information, please refer to the Orb docs for the cloud product.

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.

We recommend that you leave these options at their defaults until you have successfully configured and verified your server installation.

AWS EC2

You will need the following fields to configure your VM Service to work with AWS EC2. Note that the Access Key and Secret Key used by VM Service differs from the policy used by object storage in the previous section.

1. **AWS Region** (required): This is the region the application is in.
2. **AWS Windows AMI ID** (optional): If you require Windows builders, you can supply an AMI ID for them here.
3. **Subnet ID** (required): Choose a subnet (public or private) where the VMs should be deployed.
4. **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.

5. **AWS IAM Access Key ID** (required): AWS Access Key ID for EC2 access.
6. **AWS IAM Secret Key** (required): IAM Secret Key for EC2 access.

It is recommended to create a new user with programmatic access for this purpose. You should attach the following IAM policy to the user:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": "ec2:RunInstances",
      "Effect": "Allow",
      "Resource": [
        "arn:aws:ec2:*::image/*",
        "arn:aws:ec2:*::snapshot/*",
        "arn:aws:ec2:*:*:key-pair/*",
        "arn:aws:ec2:*:*:launch-template/*",
        "arn:aws:ec2:*:*:network-interface/*",
        "arn:aws:ec2:*:*:placement-group/*",
        "arn:aws:ec2:*:*:volume/*",
```

```
        "arn:aws:ec2:*:*:subnet/*",
        "arn:aws:ec2:*:*:security-group/${SECURITY_GROUP_ID}"
      ]
    },
    {
      "Action": "ec2:RunInstances",
      "Effect": "Allow",
      "Resource": "arn:aws:ec2:*:*:instance/*",
      "Condition": {
        "StringEquals": {
          "aws:RequestTag/ManagedBy": "circleci-vm-service"
        }
      }
    },
    {
      "Action": [
        "ec2:CreateVolume"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:ec2:*:*:volume/*"
      ],
      "Condition": {
        "StringEquals": {
          "aws:RequestTag/ManagedBy": "circleci-vm-service"
        }
      }
    },
    {
      "Action": [
        "ec2:Describe*"
      ],
      "Effect": "Allow",
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "ec2:CreateTags"
      ],
      "Resource": "arn:aws:ec2:*:*:*/*",
```

```
      "Condition": {
        "StringEquals": {
          "ec2:CreateAction" : "CreateVolume"
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": [
        "ec2:CreateTags"
      ],
      "Resource": "arn:aws:ec2:*:*:*/*",
      "Condition": {
        "StringEquals": {
          "ec2:CreateAction" : "RunInstances"
        }
      }
    },
    {
      "Action": [
        "ec2:CreateTags",
        "ec2:StartInstances",
        "ec2:StopInstances",
        "ec2:TerminateInstances",
        "ec2:AttachVolume",
        "ec2:DetachVolume",
        "ec2:DeleteVolume"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:ec2:*:*:*/*",
      "Condition": {
        "StringEquals": {
          "ec2:ResourceTag/ManagedBy": "circleci-vm-service"
        }
      }
    },
    {
      "Action": [
        "ec2:RunInstances",
        "ec2:StartInstances",
        "ec2:StopInstances",
```

```
        "ec2:TerminateInstances"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:ec2:*:*:subnet/*",
      "Condition": {
        "StringEquals": {
          "ec2:Vpc": "${VPC_ARN}"
        }
      }
    }
  ]
}
```

Google Cloud Platform

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

1. **GCP project ID** (required): Name of the GCP project the cluster resides.
2. **GCP Zone** (required): GCP zone the virtual machines instances should be created in IE `us-east1-b`.
3. **GCP Windows Image** (optional): Name of the image used for Windows builds. Leave this field blank if you do not require them.
4. **GCP VPC Network** (required): Name of the VPC Network.
5. **GCP VPC Subnet** (optional): Name of the VPC Subnet. If using auto-subnetting, leave this field blank.
6. **GCP Service Account JSON file** (required): Copy and paste the contents of your service account JSON file.

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.

Configuring VM Service

1. **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. Note, that 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.
2. **VM Service Custom Configuration**: Custom configuration can fine tune many aspects of your VM service. This is an advanced option and we recommend you reach out to your account manager to learn more.

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 go into effect for CircleCI users in the future. That's why we're 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
```

You can also use images from a private repository like gcr.io or quay.io. Make sure to 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 speeds up 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 come in handy where you have different AWS credentials for different infrastructure. For example, let's say your SaaS app runs the speedier tests and deploys to staging infrastructure on every commit while for Git tag pushes, we run the full-blown 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+$/
```

See also

- Configuration Reference

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 pre-defined artifact types are allowed. 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	Type
Text	html
Text	css
Text	javascript
Text	ecmascript
Application	javascript
Application	ecmascript
Text	xml

CircleCI Server v3.x Usage Data

CircleCI typically collects usage data such as logs and other aggregated data for the purpose of improving our product 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. In short, 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 go around 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 code base 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 doing this a user will have complete access to any files or processes being run inside that build container, so provide access to CircleCI only to those also trusted with your source code.

Integrations

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

- **Web Sockets** We use Pusher client libraries for WebSocket communication between the server and the browser, though 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 (eg: 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.
- **Support Bundles** We use Honeycomb to process and analyze distributed tracing data from Support Bundles that are sent to us. The traces contain metadata about activity in your instance but no secrets, source code, or build output are included. Data is retained for a maximum of 60 days.

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 from the Audit Log page within the Admin section of the application as a CSV file. Audit log fields with nested data contain JSON blobs.

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

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 things you can ask your team to consider for security best practices as *users* of CircleCI:

- Minimise 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 land 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 organizations) 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 or not 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 having 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 --namespace <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 to provide to CircleCI for 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 --field-selector option.

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

Verify Pod Settings and Status

To show detail 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 not yet available on the current version of Server 3.x; however, it will be available for preview before the next release.

While operating and administering CircleCI server, you will undoubtedly ponder 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 our recommendations for how to backup 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.

The setup

Backups of CircleCI server can be created quite easily through Kots. To enable backup support however, you will need to install and configure Velero on your cluster. Below, we will outline the steps needed to install Velero on your cluster.

1. Download and install the Velero CLI for your environment.
2. You will also need an S3 compatible storage bucket created.

The following steps will assume AWS as the provider and that you have aws cli installed, however you may always refer to the linked documentation for instructions on your specific bucket provider.

kots backups require [restic](https://restic.net/) to operate. When installing Velero, ensure that you have the `--use-restic` flag set. Below we discuss installation on AWS for an S3 bucket, however you may refer to these docs for your provider.

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 (eg: `./credentials-velero`) in your local directory:

```
[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 the step before.

```
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
```

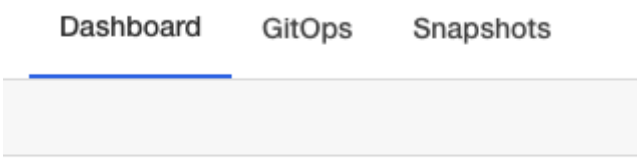
- Once Velero is installed on your cluster, check the new `velero` namespace. You should have a Velero deployment and a restic daemonset. eg:

```
$ 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 your kots admin 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.

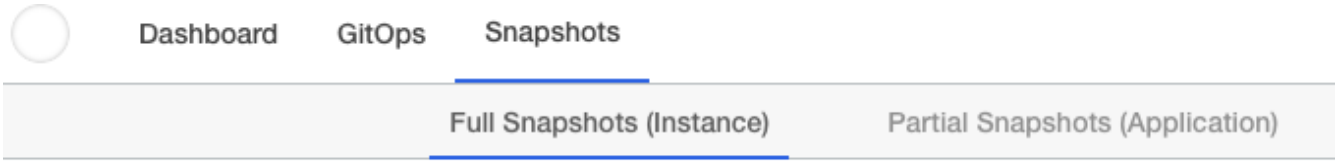
Option 1 - Create a backup with kots CLI

To create the backup, run:

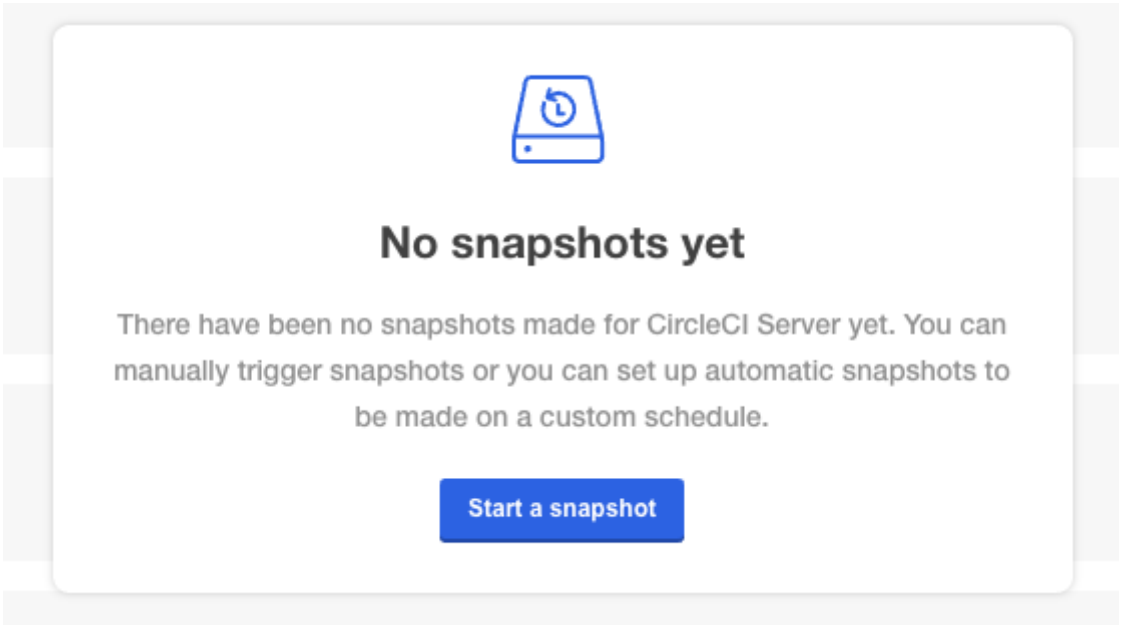
```
kubectl kots backup --namespace <your namespace>
```

Option 2 - Create a backup with kots admin console

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



Select the **Start a snapshot** button.



Restoring backups

Option 1 - Restore a backup from a snapshot

Unlike other restore procedures, which would require you to reinstall server and then restore the data, restoring CircleCI server from a kots backup does not require you to reinstall server yourself before-hand. To restore from a backup stored in your S3 compatible storage, you will need to ensure Velero is installed and configured on your Kubernetes cluster, using the instructions above. Velero must have access to the storage bucket containing the backups.

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

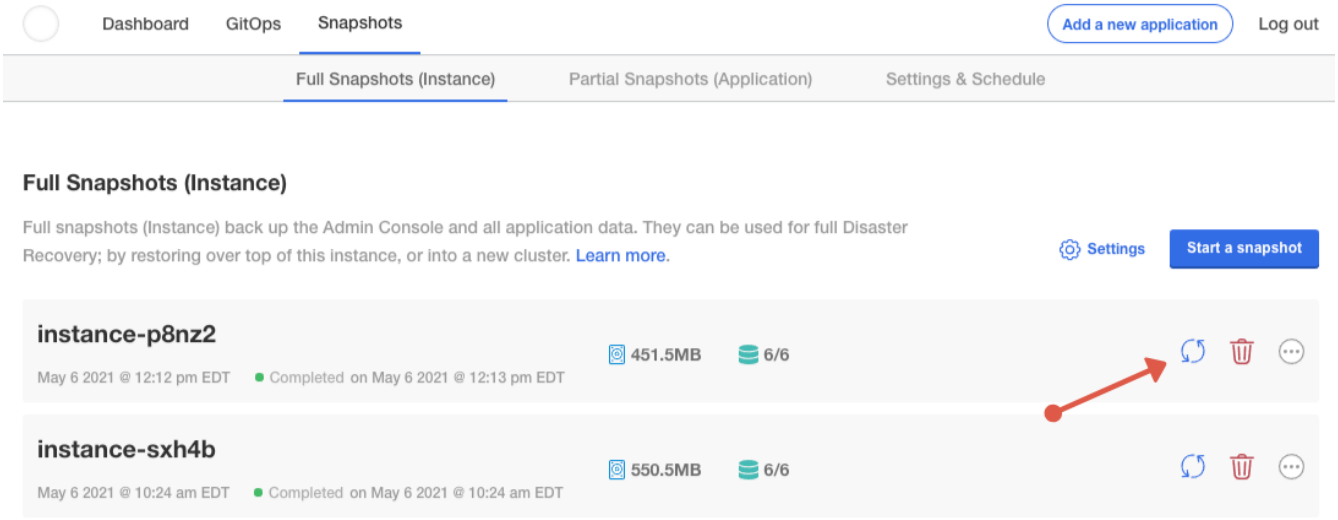
Option 2 - Restore a backup using the kots CLI

To restore a backup using the kots CLI, run the following:

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

Option 3 - Restore a backup using the kots administration console UI

As with backups, navigate to **Snapshots** in kots admin. 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.



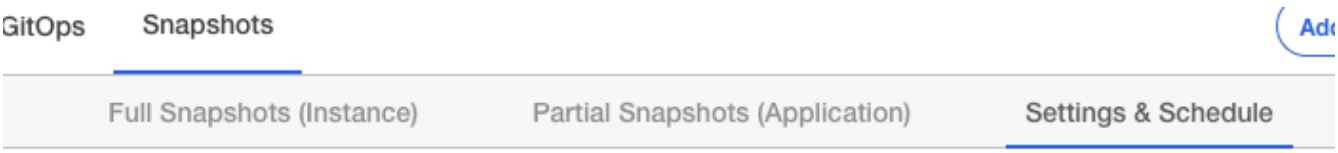
The restore will create new load balancers for CircleCI's services. You will need to either update your DNS records or the hostname configurations in kots admin-console as a result. You may also need to consider updating the `nomad server endpoint` provided to your nomad clients.

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

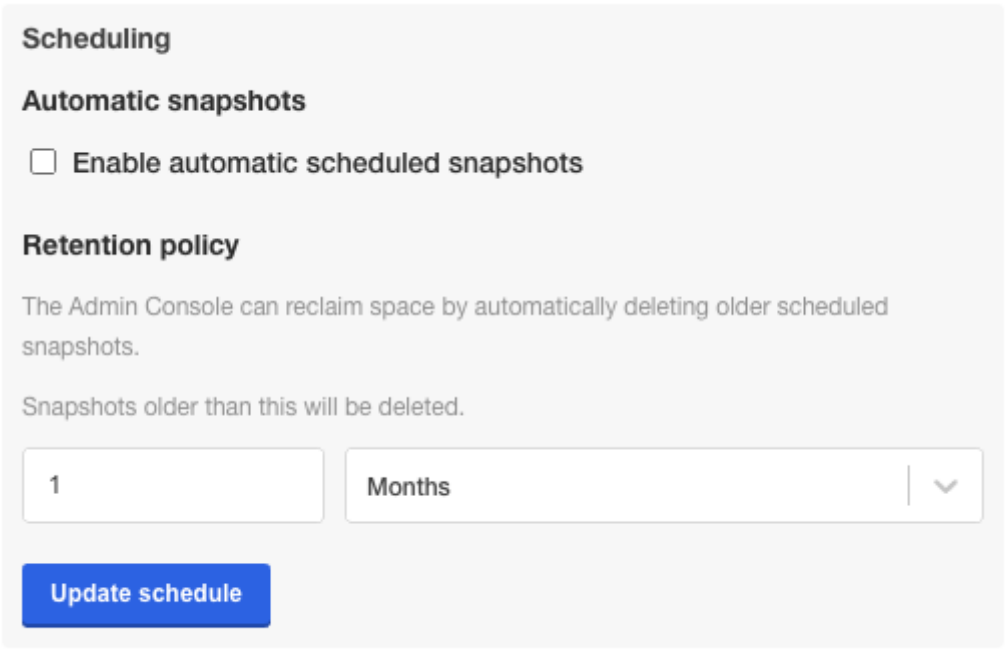
It should take roughly 10-15 mins for CircleCI server to be restored and operational.

Optional - Scheduling backups with kots

To schedule regular backups, select **Snapshots**, and then **Settings & Schedule** from the kots administration 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 to validate this.

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.
- Then 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 info.

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. You may need to scale your nodes to accommodate restic in this case.