OPERATIONS GUIDE

A guide for administrators of CircleCI Server installations on AWS and private infrastructure.

docs@circleci.com

Version 2.19.8, 10/15/2020: FINAL

Overview

CircleCI Server v2.19.8 uses the CircleCI 2.0 architecture.
`version: 2.0` should be used in all `.circleci/config.yml` files.
Currently not supported for Server: orbs, reusable commands, pipelines.

CircleCI Server is a modern continuous integration and continuous delivery (CI/CD) platform installable inside your private cloud or data center. Refer to the Changelog for what's new in this CircleCI Server release.

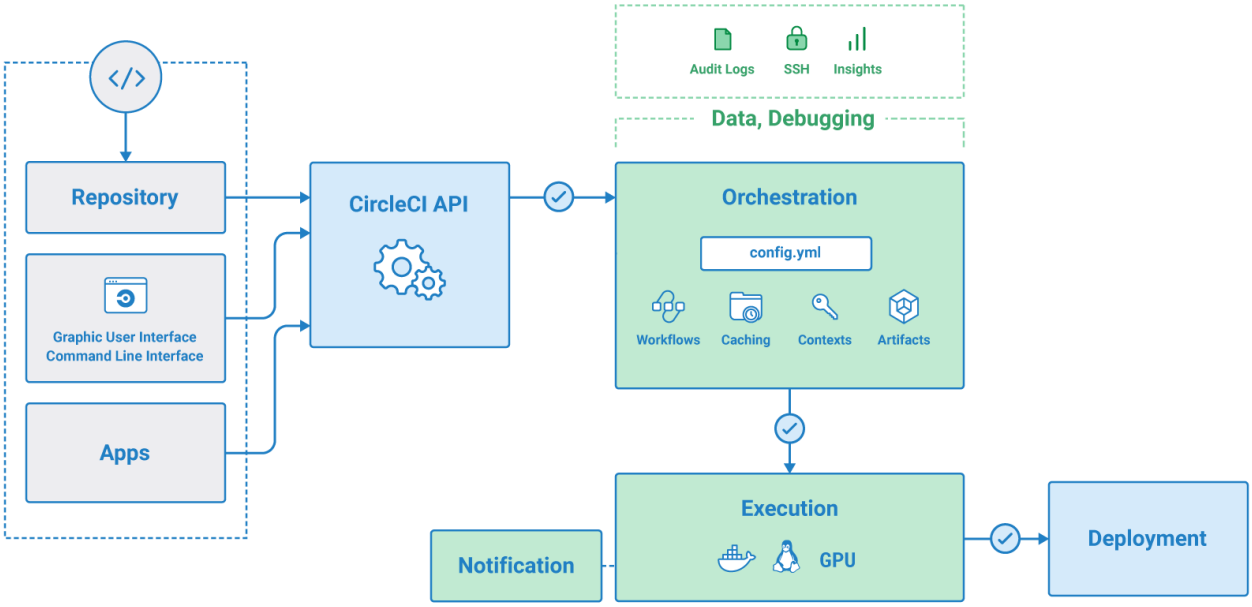


Figure 1. CircleCI Services Architecture

Build Environments

CircleCI 2.0 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 2.0 Nomad clients automatically provision containers according to the image configured for each job in your `.circleci/config.yml` file.

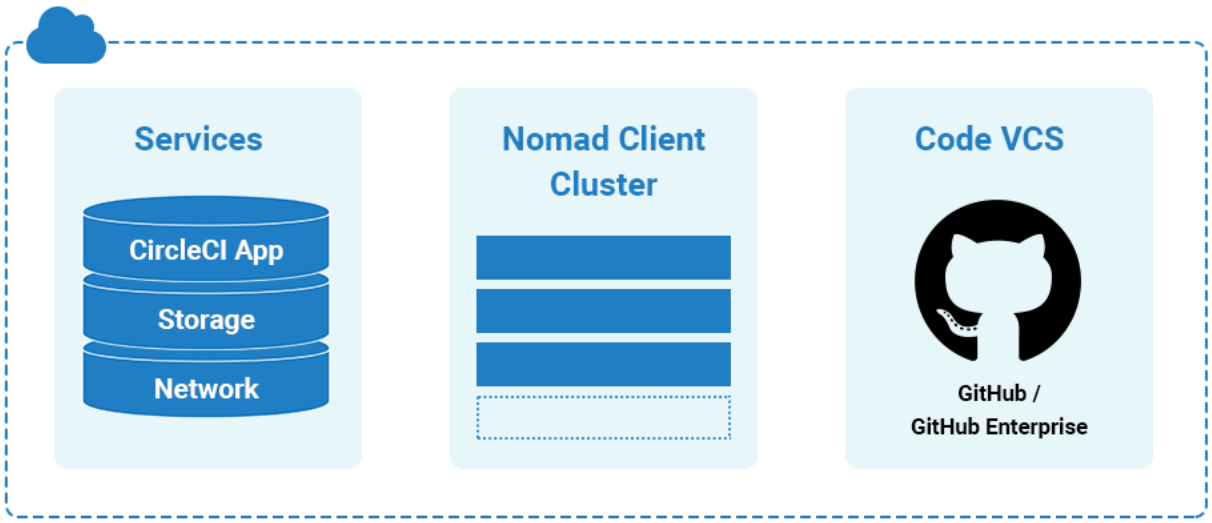
Architecture

Figure 1.1 illustrates CircleCI core components, build orchestration services, and executors. The CircleCI API is a full-featured RESTful API that allows you to access all information and trigger all actions in CircleCI.

Within the CircleCI UI is the Insights page, which acts as a dashboard showing the health of all repositories you are following including:

- median build time
- median queue time
- last build time
- success rate
- parallelism

CircleCI consists of two primary components: Services and Nomad Clients. Any number of Nomad Clients execute your jobs and communicate back to the Services. All components must access GitHub or your hosted instance of GitHub Enterprise on the network, as illustrated below.



Services Machine

The Services machine must not be restarted and may be backed up using VM snapshotting. If you must restart the Services machine, do so only as a last resort, because a restart will result in downtime. Refer to the Backup and Recovery chapter for instructions.

DNS resolution may point to the IP address of the Services machine. It is also possible to point to a load balancer, for example an ELB in AWS. The following table describes the ports used for traffic on the Service machine:

Source	Ports	Use
End Users	80, 443, 4434	HTTP/HTTPS Traffic
Administrators	22	SSH
Administrators	8800	Admin Console
Builder Boxes	all traffic, all ports	Internal Communication
GitHub (Enterprise or .com)	80, 443	Incoming Webhooks

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 Monitoring Your Installation.

Each machine reserves two vCPUs and 4GB of memory for coordinating builds. The remaining processors and memory create the containers. 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.

The following table describes the ports used on Nomad clients:

Source	Ports	Use
End Users	64535-65535	SSH into builds
Administrators	80 or 443	CCI API Access
Administrators	22	SSH
Services Machine	all traffic, all ports	Internal Comms
Nomad Clients (including itself)	all traffic, all ports	Internal Comms

GitHub

CircleCI uses GitHub or GitHub Enterprise credentials for authentication which, in turn, may use LDAP, SAML, or SSH for access. This means CircleCI will inherit the authentication supported by your central SSO infrastructure.

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, 443	API Access
Nomad Client	22	Git Access
Nomad Client	80, 443	API Access

Introduction to Nomad Cluster Operation

CircleCI 2.0 uses Nomad as the primary job scheduler. This chapter provides a basic introduction to Nomad for understanding how to operate the Nomad Cluster in your CircleCI 2.0 installation.

Basic Terminology and Architecture

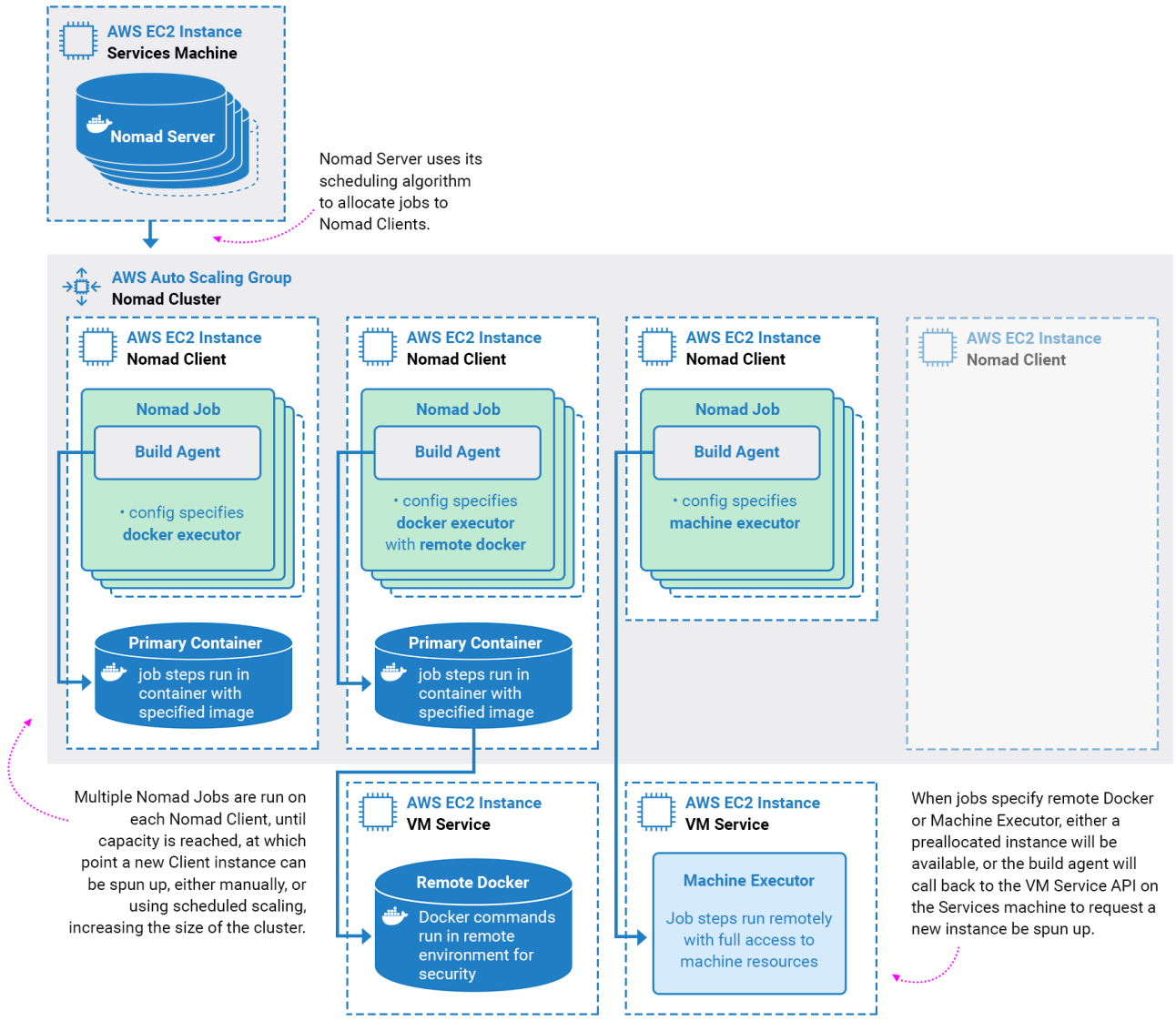


Figure 2. Nomad Cluster Management

- **Nomad Server:** Nomad servers are the brains of the cluster; they receive and allocate jobs to Nomad clients. In CircleCI, a Nomad server runs on your Services machine as a Docker Container.
- **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) in order to fully take the 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. In CircleCI 2.0, a Nomad job corresponds to an execution of a CircleCI job. If the job uses parallelism, say 10 parallelism, then Nomad will run 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 Service instance. It is pre-configured to talk to the Nomad cluster, so it is possible to use the `nomad` command to run the following commands in this section.

Checking the Jobs Status

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

```
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:

```
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 from that client:

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

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

Checking Logs

As noted in the Nomad Jobs section above, 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:

```
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, the command 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 out of scope for 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>`

Scaling the Cluster

By default, your Nomad Client is set up within an Auto Scaling Group (ASG) within AWS. To view settings:

1. Go to your EC2 Dashboard and select Auto Scaling Groups from the left hand menu
2. Select your Nomad Client
3. Select Actions > Edit to set Desired/Minimum/Maximum counts. This defines the number of Nomad Clients to spin up and keep available. Use the Scaling Policy tab to scale up your group automatically at your busiest times, see below for best practices for defining scaling policies. Use nomad job metrics to assist in defining your scaling policies.

Auto Scaling Policy Best Practices

There is a blog post series wherein CircleCI engineering spent time running simulations of cost savings for the purpose of developing a general set of best practices for Auto Scaling. Consider the following best practices when setting up AWS Auto Scaling:

1. In general, size your cluster large enough to avoid queueing builds. That is, less than one second of queuing for most workloads and less than 10 seconds for workloads run on expensive hardware or at highest parallellism. Sizing to reduce queuing to zero is best practice because of the high cost of developer time. It is difficult to create a model in which developer time is cheap enough for under-

provisioning to be cost-effective.

2. Create an Auto Scaling Group with a Step Scaling policy that scales up during the normal working hours of the majority of developers and scales back down at night. Scaling up during the weekday normal working hours and back down at night is the best practice to keep queue times down during peak development, without over provisioning at night when traffic is low. Looking at millions of builds over time, a bell curve during normal working hour emerges for most data sets.

This is in contrast to auto scaling throughout the day based on traffic fluctuations, because modelling revealed that boot times are actually too long to prevent queuing in real time. Use Amazon's Step Policy instructions to set this up along with Cloudwatch Alarms.

Shutting Down a Nomad Client

When you want to shutdown 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

Monitoring Your Installation

This section includes information on metrics for monitoring your CircleCI Server installation.

Metrics Overview

Metrics are technical statistical data collected for monitoring and analytics purposes. The data includes basic information, such as CPU or memory usage, as well as more advanced counters, such as number of executed builds and internal errors. Using metrics you can:

- Quickly detect incidents and abnormal behavior
- Dynamically scale compute resources
- Retroactively understand infrastructure-wide issues

How Metrics Work in CircleCI Server

Telegraf is the main component used for metrics collection in CircleCI Server. Telegraf is server software that brokers metrics data emitted by CircleCI services to data monitoring platforms such as Datadog or AWS CloudWatch.

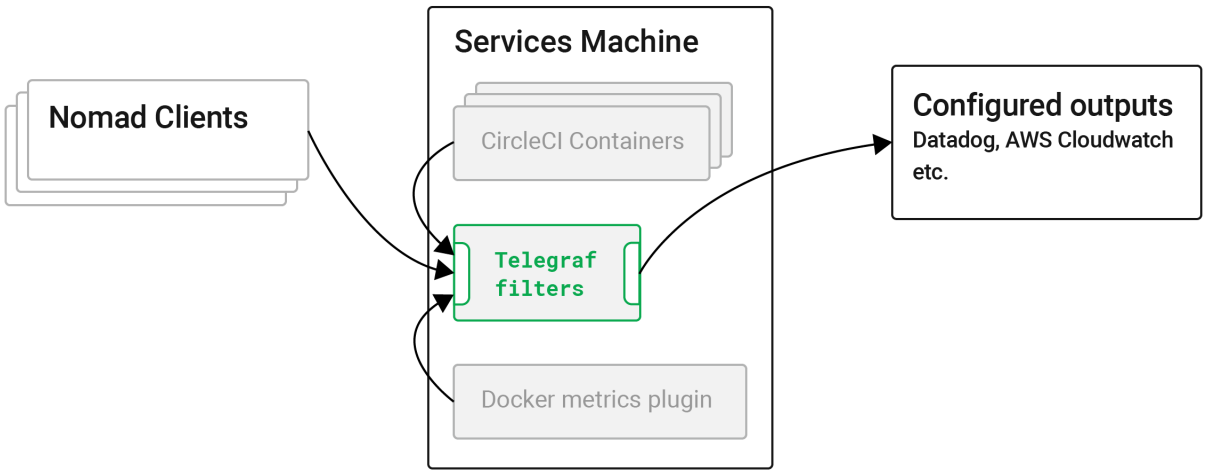


Figure 3. Metrics

Metrics collection in CircleCI Server works as follows:

- Each component of a Server installation sends metrics data to the telegraf container running on the Services machine.
- Telegraf listens on port 8125/UDP and receives data from all components (inputs) and applies configured filters to determine whether data should be kept or dropped.
- For some metric-types, Telegraf keeps metrics data inside and calculates statistical data (such as max, min, mean, stdev, sum) periodically.
- Finally, Telegraf sends out data to configured sinks (outputs), such as stdout (on the Services machine), Datadog and/or AWS CloudWatch.

It is worth noting that Telegraf can accept multiple input and output types at the same time allowing

administrators to configure a single Telegraf instance to collect and forward multiple metrics data sets to both Datadog and CloudWatch.

Standard Metrics Configuration

Review your metrics configuration file using the following command:

```
sudo docker inspect --format='{{range .Mounts}}{{println .Source "->" .Destination}}{{end}}'
telegraf | grep telegraf.conf | awk '{ print $1 }' | xargs cat
```

There are four notable blocks in the file (some blocks might not be there depending on your configuration in the Management Console):

- `[[inputs.statsd]]` – Input configuration to receive metrics data through 8125/UDP (as discussed above)
- `[[outputs.file]]` – Output configuration to emit metrics to stdout. All accepted metrics are configured to be shown in Telegraf docker logs. This is helpful for debugging your metrics configuration.
- `[[outputs.cloudwatch]]` – Output configuration to emit metrics to CloudWatch
- `[[outputs.datadog]]` – Output configuration to emit metrics to Datadog

This configuration file is automatically generated by Replicated (the service used to manage and deploy CircleCI Server) and is fully managed by Replicated. If you wish to customize the standard configuration you will need to configure Replicated to **not** insert the blocks you want to change.

Do not attempt to directly modify the file. Any changes made in this way will be destroyed by Replicated upon certain events, such as service restarts. For example, if a customized `[[inputs.statsd]]` block is added *without* stopping automatic interpolation, you will encounter errors as Telegraf attempts to listen to `8125/UDP` twice, and the second attempt will fail with `EADDRINUSE`.

In a standard configuration with no metrics customization the main config discussed above is all that is required. If you have configured metrics customization by placing files under `/etc/circleconfig/telegraf`, those configurations are appended to the main config – imagine `cat`ing the main config and all of those customization files. For more on customizing metrics see the Custom Metrics section.

System Monitoring Metrics

To enable metrics forwarding to either AWS Cloudwatch or Datadog, follow the steps for the service you wish to use in the Supported Platforms section. The following sections give an overview of available metrics for your installation.

VM Service and Docker Metrics

VM Service and Docker services metrics are forwarded via Telegraf, a plugin-driven server agent for collecting and reporting metrics.

The following metrics are enabled:

- CPU
- Disk
- Memory
- Networking
- Docker

Nomad Job Metrics

Nomad job metrics are enabled and emitted by the Nomad Server agent. Five types of metrics are reported:

Metric	Description
`circle.nomad.server_agent.poll_failure`	Returns 1 if the last poll of the Nomad agent failed, otherwise it returns 0.
`circle.nomad.server_agent.jobs.pending`	Returns the total number of pending jobs across the cluster.
`circle.nomad.server_agent.jobs.running`	Returns the total number of running jobs across the cluster.
`circle.nomad.server_agent.jobs.complete`	Returns the total number of complete jobs across the cluster.
`circle.nomad.server_agent.jobs.dead`	Returns the total number of dead jobs across the cluster.

When the Nomad metrics container is running normally, no output will be written to standard output or standard error. Failures will elicit a message to standard error.

CircleCI Metrics

Introduced in CircleCI Server v2.18

`circle.backend.action.upload-artifact-error`	Tracks how many times an artifact has failed to upload.
`circle.build-queue.runnable.builds`	Tracks how many builds flowing through the system are considered runnable.
`circle.dispatcher.find-containers-failed`	Tracks how many 1.0 builds
`circle.github.api_call`	Tracks how many api calls CircleCI is making to github
`circle.http.request`	Tracks the response codes to CircleCi requests
`circle.nomad.client_agent.*`	Tracks nomad client metrics
`circle.nomad.server_agent.*`	Tracks how many nomad servers there are.
`circle.run-queue.latency`	Tracks how long it takes for a runnable build to be accepted.

`circle.state.container-builder-ratio`	Keeps track of how many containers exist per builder (1.0 only).
`circle.state.lxc-available`	Tracks how many containers are available (1.0 only)
`circle.state.lxc-reserved`	Tracks how many containers are reserved/in use (1.0 only).
`circleci.cron-service.messaging.handle-message`	Provides timing and counts for RabbitMQ messages processed by the `cron-service`
`circleci.grpc-response`	Tracks latency over the system grpc system calls.

Supported Platforms

We have two built-in platforms for metrics and monitoring: AWS CloudWatch and DataDog. The sections below detail enabling and configuring each in turn.

AWS CloudWatch

To enable AWS CloudWatch complete the following:

1. Navigate to the settings page within your Management Console. You can use the following URL, substituting your CircleCI URL: `your-circleci-hostname.com:8800/settings#cloudwatch_metrics`.
2. Check Enabled under AWS CloudWatch Metrics to begin configuration.

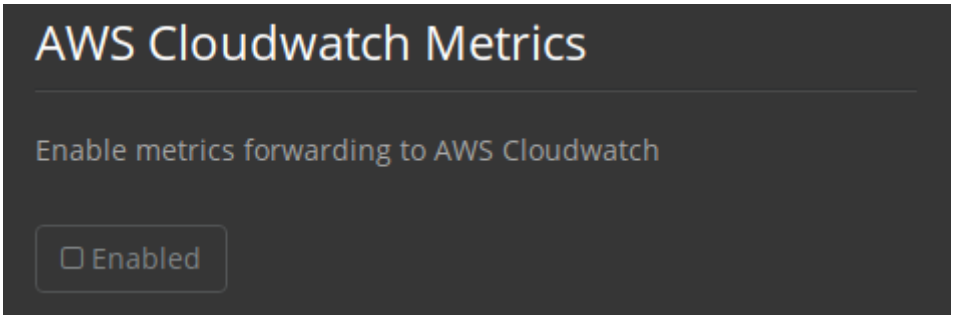


Figure 4. Enable Cloudwatch

AWS CloudWatch Configuration

There are two options for configuration:

- Use the IAM Instance Profile of the services box and configure your custom region and namespace.

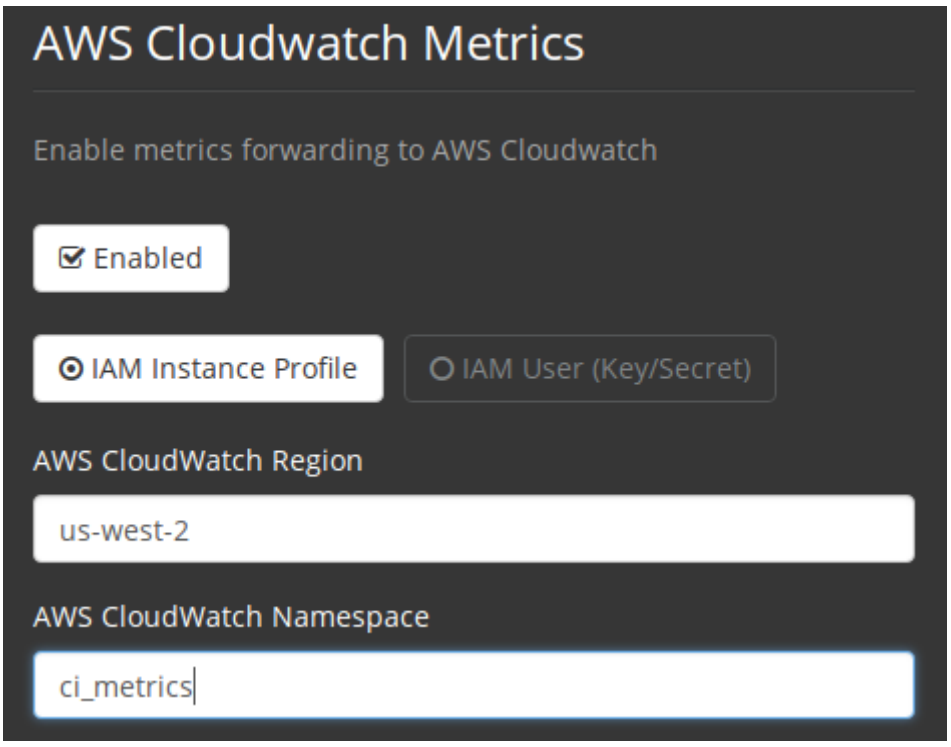


Figure 5. CloudWatch Region and Namespace

- Alternatively, you may use your AWS Access Key and Secret Key along with your custom region and namespace.

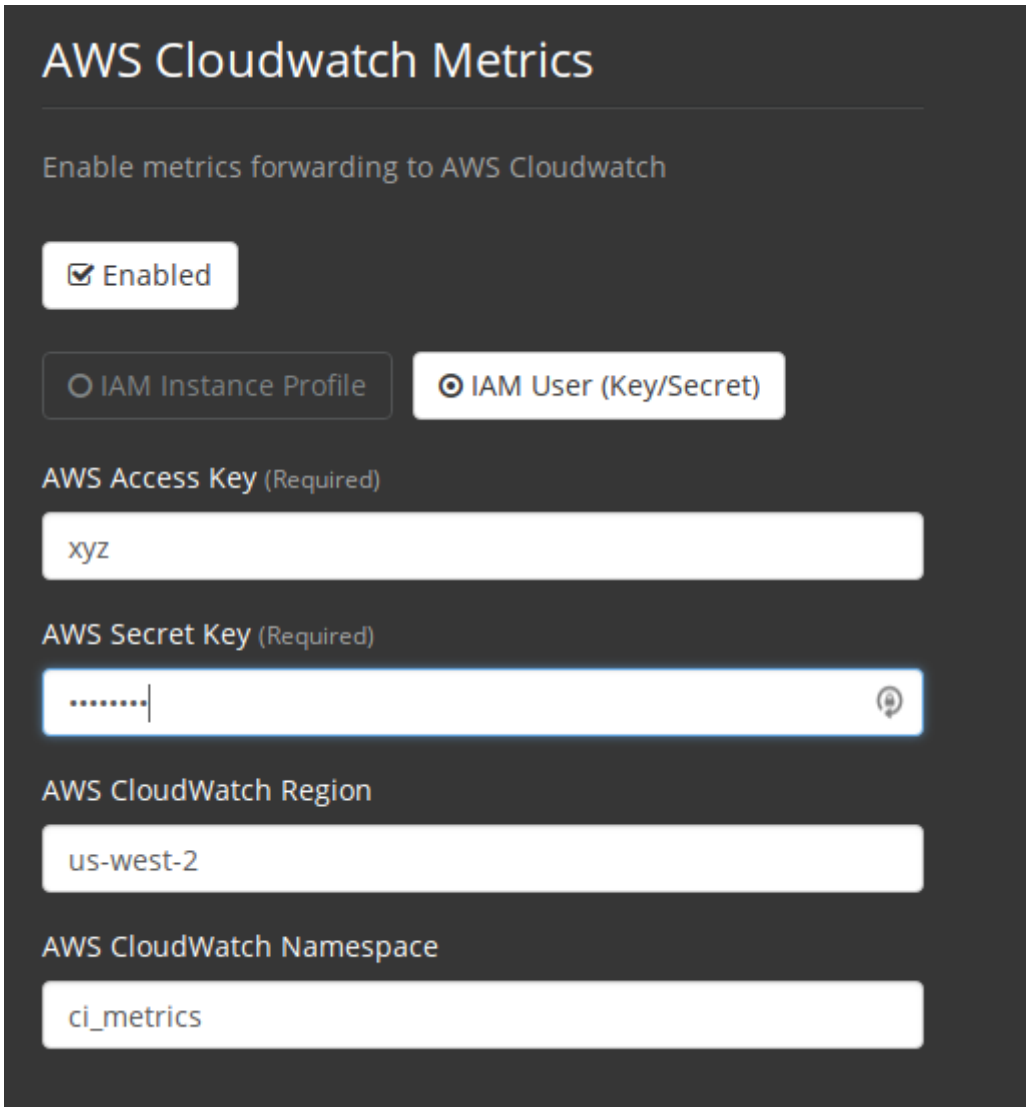


Figure 6. Access Key and Secret Key

After saving you can **verify** that metrics are forwarding by going to your AWS CloudWatch console.

DataDog

To enable Datadog complete the following:

1. Navigate your Management Console Settings. You can use the following URL, substituting your CircleCI hostname: `your-circleci-hostname.com:8800/settings#datadog_metrics`
2. Check Enabled under Datadog Metrics to begin configuration.

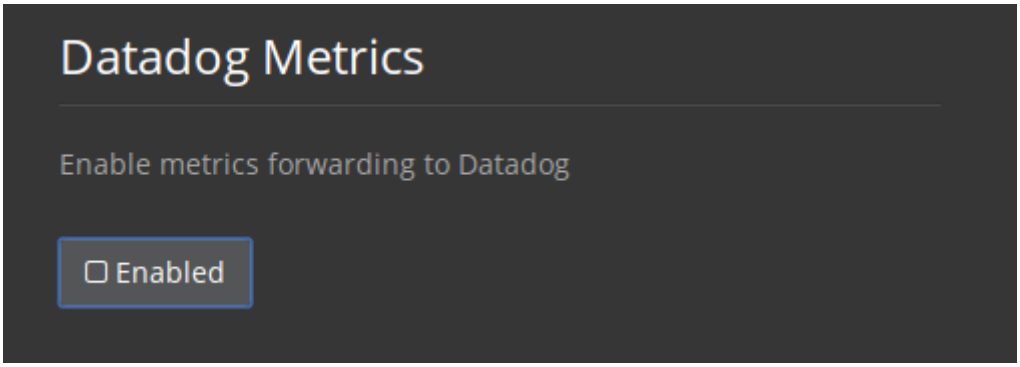


Figure 7. Enable Datadog Metrics

3. Enter your DataDog API Key. You can verify that metrics are forwarding by going to your DataDog metrics summary.

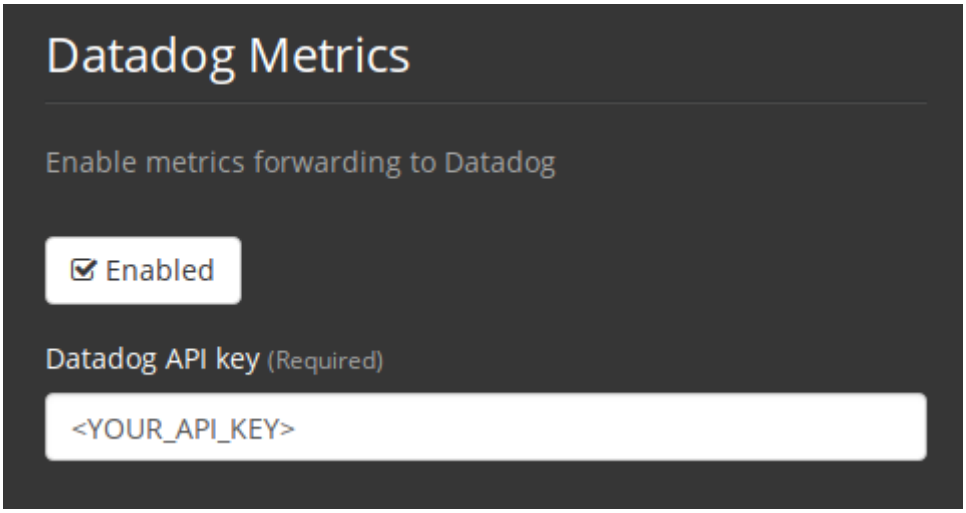


Figure 8. Enter Datadog API key

Custom Metrics

Custom Metrics using a Telegraf configuration file allows for more fine grained control than allowing Replicated to forward standard metrics to Datadog or AWS Cloudwatch.

The basic Server metrics configuration assumes fundamental use cases only. It might be beneficial to customize the way metrics are handled for your installation in the following ways:

- Forward metrics data to your preferred platform (e.g. your own InfluxDB instance)
- Monitor additional metrics in order to detect specific events
- Reduce the number of metrics sent to data analysis platforms (to reduce gross operation costs)

1. Disable Standard Metrics Setup

Disable Replicated's interpolation of the Telegraf configuration to fully customize and outputs:

1. Open the Management Console.
2. On the **Settings** page, go to **Custom Metrics** section and enable the "Use custom telegraf metrics" option.

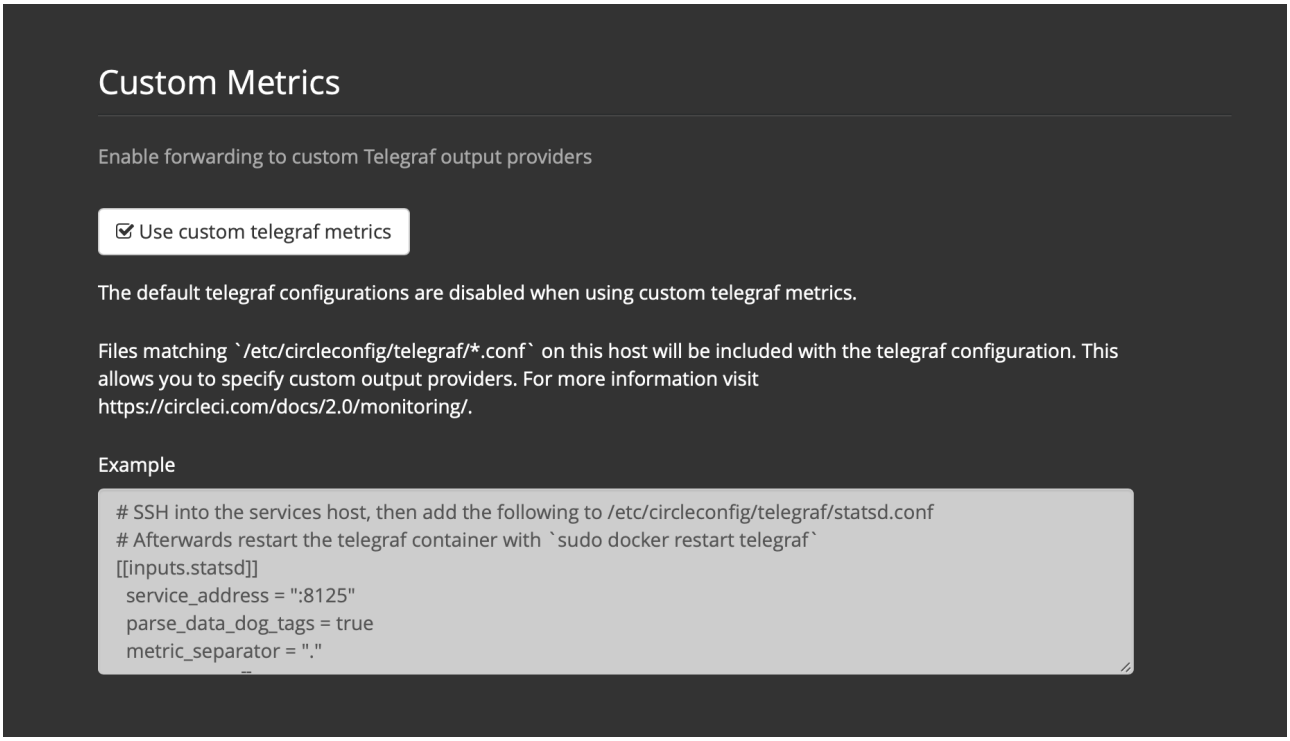


Figure 9. Custom Metrics

3. Scroll down to save the change and restart services.

There will be a downtime along with a service restart. After disabling it you will have to manually configure outputs to Datadog and/or CloudWatch, regardless of configurations on Replicated.

2. Create your Customized Config

Now you are ready to do anything Telegraf supports! All you need to provide is a valid Telegraf config file.

1. SSH into the Services machine
2. Add the following to `/etc/circleconfig/telegraf/statsd.conf`

```
[[inputs.statsd]]
        service_address = ":8125"
        parse_data_dog_tags = true
        metric_separator = "."
        namepass = []
```

3. Under `namepass` add any metrics you wish to receive, the example below shows choosing to configure just the first 4 from the list above. (See below for some additional example configs):

```
[[inputs.statsd]]
        service_address = ":8125"
        parse_data_dog_tags = true
        metric_separator = "."
        namepass = [
            "circle.backend.action.upload-artifact-error",
            "circle.build-queue.runnable.builds",
            "circle.dispatcher.find-containers-failed",
            "circle.github.api_call"
          ]
```

4. Restart the telegraf container by running: `sudo docker restart telegraf`

See the Telegraf README for further config syntax details.

Sample Telegraph Configuration

Scenario 1: Record standard metrics to two InfluxDB instances

The example below records default metrics to two InfluxDB instances: One is your on-premises InfluxDB server (`your-influx-db-instance.example.com`), and the other is InfluxDB Cloud 2.

```
[[inputs.statsd]]
  service_address = ":8125"
  parse_data_dog_tags = true
  metric_separator = "."
  namepass = [
    "circle.backend.action.upload-artifact-error",
    "circle.build-queue.runnable.builds",
    "circle.dispatcher.find-containers-failed",
    "circle.github.api_call",
    "circle.http.request",
    "circle.nomad.client_agent.*",
    "circle.nomad.server_agent.*",
    "circle.run-queue.latency",
    "circle.state.container-builder-ratio",
    "circle.state.lxc-available",
    "circle.state.lxc-reserved",
    "circle.vm-service.vm.assigned-vm",
    "circle.vm-service.vms.delete.status",
    "circle.vm-service.vms.get.status",
    "circle.vm-service.vms.post.status",
    "circleci.cron-service.messaging.handle-message",
    "circleci.grpc-response"
  ]

[[outputs.influxdb]]
  url = "http://your-influx-db-instance.example.com:8086"
  database = "circleci"

[[outputs.influxdb_v2]]
  urls = ["https://us-central1-1.gcp.cloud2.influxdata.com"]
  token = "YOUR_TOKEN_HERE"
  organization = "circle@example.com"
  bucket = "circleci"
```

Scenario 2: Record all metrics to Datadog

The standard configuration handles only selected metrics, and there are many metrics discarded by Telegraf. If you want to receive this discarded, sophisticated data, such as JVM stats and per-container CPU usage, you can keep all received metrics by removing namepass filter. This example also illustrates how to configure metrics emission to Datadog. As discussed above, you need manual configuration for outputs to Datadog regardless of configurations on Replicated.

This scenario leads to very large amounts of data.

```
[[inputs.statsd]]
  service_address = ":8125"
  parse_data_dog_tags = true
  metric_separator = "."

[[outputs.datadog]]
  apikey = 'YOUR_API_KEY_HERE'
```

Scenario 3: Send limited metrics to CloudWatch

AWS charges fees for CloudWatch per series of scalar (i.e. at the level of "mean" or "sum"). Since multiple fields (e.g. mean, max, min and sum) are calculated for each metrics key (e.g. `circle.run-queue.latency`) and some fields can be redundant, you might want to select which fields are sent to CloudWatch. This can be achieved by configuring `[[outputs.cloudwatch]]` with `fieldpass`. You also may declare `[[outputs.cloudwatch]]` multiple times to pick up multiple metrics, as illustrated below.

```
[[inputs.statsd]]
  # Accept all metrics at input level to 1) enable output configurations without thinking of inputs,
and to 2) dump discarded metrics to stdout just in case.
  service_address = ":8125"
  parse_data_dog_tags = true
  metric_separator = "."

[[outputs.cloudwatch]]
    # Fill in these two variables if you need to access CloudWatch with an IAM User, not an IAM Role
attached to your Services box
    # access_key = 'ACCESS'
    # secret_key = 'SECRET'

    # Specify region for CloudWatch
    region = 'ap-northeast-1'
    # Specify namespace for easier monitoring
    namespace = 'my-circleci-server'

    # Name of metrics key to record
    namepass = ['circle.run-queue.latency']
    # Name of metrics field to record; key and field are delimited by an underscore (_)
    fieldpass = ['mean']

[[outputs.cloudwatch]]
    # Outputs can be specified multiple times.

    # Fill in these two variables if you need to access CloudWatch with an IAM User, not an IAM Role
attached to your Services box
    # access_key = 'ACCESS'
    # secret_key = 'SECRET'

    # Specify region for CloudWatch
    region = 'ap-northeast-1'
    # Specify namespace for easier monitoring
    namespace = 'my-circleci-server'

    # Name of metrics key to record
    namepass = ['mem']
    # Name of metrics field to record; key and field are delimited by an underscore (_)
    fieldpass = ['available_percent']
```

Additional Tips

You may check the logs by running `docker logs -f telegraf` to confirm your output provider (e.g. influx) is listed in the configured outputs. Additionally, if you would like to ensure that all metrics in an installation are tagged against an environment you could place the following code in your config:

```
[global_tags]
Env="<staging-circleci>"
```

Please see the InfluxDB documentation for default and advanced installation steps.

Any changes to the config will require a restart of the CircleCI application which will require downtime.

Configuring Nomad Client Metrics

Nomad Metrics is a helper service used to collect metrics data from the Nomad server and clients running on the Services and Nomad instances respectively. Metrics are collected and sent using the DogStatsD protocol and sent to the Services machine.

Nomad Metrics Server

The Nomad Metrics container is run on the services host using the server flag and is installed as part of the CircleCI Server installation process, requiring no additional configuration.

Nomad Metrics Client

The Nomad Metrics client is installed and run on all Nomad client instances. You will need to update your AWS Launch Configuration in order to install and configure it. Additionally, you will need to modify the AWS security group to ensure that UDP port 8125 is open on the Services machine. Steps for both configuration changes are explained below.

Before proceeding, you should be logged into the EC2 Service section of the AWS Console. Make sure that you logged into the region you use to run CircleCI Server.

Updating the Services machine Security Group

1. Select the `Instances` link located under the Instances group in the left sidebar.
2. Select the Services Box Instance. The name tag typically resembles `circleci_services`.
3. In the description box at the bottom, select the users security group link located next to the `Security Groups` section. It typically resembles `*_users_sg`.
4. This will take you straight to the Security Group page highlighting the users security group. In the description box at the bottom, select the `Inbound` tab followed by the `Edit` button.
5. Select the `Add a Rule` button. From the drop-down, select `Custom UDP Rule`. In the Port Range field enter `8125`.
6. The source field gives you a few options. However, this ultimately depends on how you have configured the VPC and subnet. Below are some more common scenarios.
    a. (Suggested) Allow traffic from the nomad client subnet. You can usually match the entries used for ports 4647 or 3001. For example, `10.0.0.0/24`.
    b. Allow all traffic to UDP port 8125 using `0.0.0.0/0`.
7. Press the `Save Button`

Updating the AWS Launch Configuration

Prerequisites

AWS EC2 Launch Configuration ID

1. Select the `Auto Scaling Groups` (ASG) link in the the sidebar on the left.

2. Locate the ASG with a name tag similar to`*_nomad_clients_asg`
3. The Launch Configuration name is next to the ASG name IE `terraform-20180814231555427200000001`

AWS EC2 Services Box Private IP Address

1. Select the `Instances` link located under the Instances group in the left sidebar
2. Select the Services Box Instance. The name tag typically resembles `circleci_services`
3. In the description box at the bottom of the page, make note of the private IP address.

Updating the Launch Configuration

1. Select the `Launch Configurations` link located under `Auto Scaling` in the sidebar to the left. Select the Launch Configuration you retrieved in the previous steps.
2. In the description pane at the bottom, select the `Copy launch configuration` button.
3. Once the configuration page opens, select `3. Configure details` link located at the top of the page.
4. Update the `Name` field to something meaningful IE `nomad-builder-with-metrics-lc-DATE`.
5. Select the `Advanced Details` drop down.
6. Copy and paste the launch configuration script from below in the text field next to `User data`.
7. **IMPORTANT:** Enter the private IP address of the services box at Line 10. For example, `export SERVICES_PRIVATE_IP="192.168.1.2"`.
8. Select the `Skip to review` button and then the `Create launch configuration` button.

```
#!/bin/bash

set -exu

export http_proxy=""
export https_proxy=""
export no_proxy=""
export aws_instance_metadata_url="http://169.254.169.254"
export PUBLIC_IP="$(curl $aws_instance_metadata_url/latest/meta-data/public-ipv4)"
export PRIVATE_IP="$(curl $aws_instance_metadata_url/latest/meta-data/local-ipv4)"
export DEBIAN_FRONTEND=noninteractive
UNAME="$(uname -r)"
export CONTAINER_NAME="nomad_metrics"
export CONTAINER_IMAGE="circleci/nomad-metrics:0.1.198-5f5befe"
export SERVICES_PRIVATE_IP=""
export NOMAD_METRICS_PORT="8125"

echo "------------------------------------------"
echo "     Performing System Updates"
echo "------------------------------------------"
apt-get update && apt-get -y upgrade

echo "--------------------------------------"
echo "        Installing NTP"
echo "--------------------------------------"
```

```
apt-get install -y ntp

# Use AWS NTP config for EC2 instances and default for non-AWS
if [ -f /sys/hypervisor/uuid ] && [ `head -c 3 /sys/hypervisor/uuid` == ec2 ]; then
cat <<EOT > /etc/ntp.conf
driftfile /var/lib/ntp/ntp.drift
disable monitor

restrict default ignore
restrict 127.0.0.1 mask 255.0.0.0
restrict 169.254.169.123 nomodify notrap

server 169.254.169.123 prefer iburst
EOT
else
  echo "USING DEFAULT NTP CONFIGURATION"
fi

service ntp restart

echo "--------------------------------------"
echo "        Installing Docker"
echo "--------------------------------------"
apt-get install -y apt-transport-https ca-certificates curl software-properties-common
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | apt-key add -
add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs)
stable"
apt-get install -y "linux-image-$UNAME"
apt-get update
apt-get -y install docker-ce=5:18.09.9~3-0~ubuntu-xenial

# force docker to use userns-remap to mitigate CVE 2019-5736
apt-get -y install jq
mkdir -p /etc/docker
[ -f /etc/docker/daemon.json ] || echo '{}' > /etc/docker/daemon.json
tmp=$(mktemp)
cp /etc/docker/daemon.json /etc/docker/daemon.json.orig
jq '.["userns-remap"]="default"' /etc/docker/daemon.json > "$tmp" && mv "$tmp"
/etc/docker/daemon.json

sudo echo 'export http_proxy="${http_proxy}"' >> /etc/default/docker
sudo echo 'export https_proxy="${https_proxy}"' >> /etc/default/docker
sudo echo 'export no_proxy="${no_proxy}"' >> /etc/default/docker
sudo service docker restart
sleep 5

echo "--------------------------------------"
echo " Populating /etc/circleci/public-ipv4"
echo "--------------------------------------"
if ! (echo $PUBLIC_IP | grep -qP "^[\d.]+$")
then
  echo "Setting the IPv4 address below in /etc/circleci/public-ipv4."
  echo "This address will be used in builds with \"Rebuild with SSH\"."
  mkdir -p /etc/circleci
```

```
  echo $PRIVATE_IP | tee /etc/circleci/public-ipv4
fi

echo "---------------------------------------"
echo "         Installing nomad"
echo "---------------------------------------"
apt-get install -y zip
curl -o nomad.zip https://releases.hashicorp.com/nomad/0.9.3/nomad_0.9.3_linux_amd64.zip
unzip nomad.zip
mv nomad /usr/bin

echo "---------------------------------------"
echo "      Creating config.hcl"
echo "---------------------------------------"
export INSTANCE_ID="$(curl $aws_instance_metadata_url/latest/meta-data/instance-id)"
mkdir -p /etc/nomad
cat <<EOT > /etc/nomad/config.hcl
log_level = "DEBUG"
name = "$INSTANCE_ID"
data_dir = "/opt/nomad"
datacenter = "default"
advertise {
    http = "$PRIVATE_IP"
    rpc = "$PRIVATE_IP"
    serf = "$PRIVATE_IP"
}
client {
    enabled = true
    # Expecting to have DNS record for nomad server(s)
    servers = ["$SERVICES_PRIVATE_IP:4647"]
    node_class = "linux-64bit"
    options = {"driver.raw_exec.enable" = "1"}
}

telemetry {
    publish_node_metrics = true
    statsd_address = "$SERVICES_PRIVATE_IP:8125"
}
EOT

echo "---------------------------------------"
echo "      Creating nomad.conf"
echo "---------------------------------------"
cat <<EOT > /etc/systemd/system/nomad.service
[Unit]
Description="nomad"
[Service]
Restart=always
RestartSec=30
TimeoutStartSec=1m
ExecStart=/usr/bin/nomad agent -config /etc/nomad/config.hcl
[Install]
WantedBy=multi-user.target
EOT
```

```
echo "------------------------------------"
echo "   Creating ci-privileged network"
echo "------------------------------------"
docker network create --driver=bridge --opt com.docker.network.bridge.name=ci-privileged ci-
privileged

echo "------------------------------------"
echo "      Starting Nomad service"
echo "------------------------------------"
service nomad restart

echo "------------------------------------"
echo "      Setting up Nomad metrics"
echo "------------------------------------"
docker pull $CONTAINER_IMAGE
docker rm -f $CONTAINER_NAME || true

docker run -d --name $CONTAINER_NAME \
    --rm \
    --net=host \
    --userns=host \
    $CONTAINER_IMAGE \
    start --nomad-uri=http://localhost:4646 --statsd-host=$SERVICES_PRIVATE_IP --statsd-port
=$NOMAD_METRICS_PORT --client
```

Updating the Auto Scaling Group

1. Select the `Auto Scaling Groups` (ASG) link in the the sidebar on the left.
2. Select the ASG with a name tag similar to `*_nomad_clients_asg`.
3. In the description box at the bottom, select the `Edit` button.
4. Select the newly created Launch Configuration from the drop-down.
5. Press the `Save` button.
6. At this point, the older Nomad client instances will begin shutting down. They will be replaced with newer Nomad clients running Nomad Metrics.

StatsD Metrics

Metrics sent via StatsD will be updated every 10s.

--server

The number of jobs in a terminal state (`complete` and `dead`) will typically increase until Nomad garbage-collects the jobs from its state.

Name	Type	Description
`circle.nomad.server_agent.poll_failure`	Gauge	1 if the last poll of the Nomad agent failed; 0 otherwise. This gauge is set independent of `circle.nomad.client_agent.poll_failure` when nomad-metrics is operating in `--client` and `--server` modes simultaneously.
`circle.nomad.server_agent.jobs.pending`	Gauge	Total number of pending jobs across the cluster.
`circle.nomad.server_agent.jobs.running`	Gauge	Total number of running jobs across the cluster.
`circle.nomad.server_agent.jobs.complete`	Gauge	Total number of complete jobs across the cluster.
`circle.nomad.server_agent.jobs.dead`	Gauge	Total number of dead jobs across the cluster.

--client

Name	Type	Description
`circle.nomad.client_agent.poll_failure`	Gauge	1 if the last poll of the Nomad agent failed; 0 otherwise.
`circle.nomad.client_agent.resources.total.cpu`	Gauge	(See below)
`circle.nomad.client_agent.resources.used.cpu`	Gauge	(See below)
`circle.nomad.client_agent.resources.available.cpu`	Gauge	(See below)
`circle.nomad.client_agent.resources.total.memory`	Gauge	(See below)
`circle.nomad.client_agent.resources.used.memory`	Gauge	(See below)
`circle.nomad.client_agent.resources.available.memory`	Gauge	(See below)
`circle.nomad.client_agent.resources.total.disk`	Gauge	(See below)
`circle.nomad.client_agent.resources.used.disk`	Gauge	(See below)
`circle.nomad.client_agent.resources.available.disk`	Gauge	(See below)

Name	Type	Description
`circle.nomad.client_agent.resources.total.iops`	Gauge	(See below)
`circle.nomad.client_agent.resources.used.iops`	Gauge	(See below)
`circle.nomad.client_agent.resources.available.iops`	Gauge	(See below)

- CPU resources are reported in units of MHz. Memory resources are reported in units of MB. Disk (capacity) resources are reported in units of MB.
- Resource metrics are scoped to the Nomad node that nomad-metrics has been configured to poll. Figures from a single nomad-metrics job operating in `--client` mode are *not* representative of the entire cluster (Though these timeseries may be aggregated by an external mechanism to arrive at a cluster-wide view.)
- All metrics in the `circle.nomad.client_agent.resources` namespace will be accompanied with the following tags when writing to DogStatsD:
  - `drain`: `true` if the Nomad node has been marked as drained; `false` otherwise.
  - `status`: One of `initializing`, `ready`, or `down`.

Setting Up HTTP Proxies

This section describes how to configure CircleCI to use an HTTP proxy.

Overview

If you are setting up your proxy through Amazon, read this before proceeding:

Using an HTTP Proxy - AWS Command Line Interface

Avoid proxying internal requests, especially for the Services machine. To add these to the `NO_PROXY` rules, run:

```
export NO_PROXY=<services_box_ip>
```

In an ideal case, traffic to S3 will not be proxied, and will instead be bypassed by adding `s3.amazonaws.com,*.s3.amazonaws.com` to the `NO_PROXY` rule.

These instructions assume an unauthenticated HTTP proxy at `10.0.0.33:3128`, a Services machine at `10.0.1.238` and use of `ghe.example.com` as the GitHub Enterprise host.

The following proxy instructions must be completed **before** installing CircleCI on fresh VMs or instances. You must also configure JVM OPTs again as described below.

Service Machine Proxy Configuration

The Service machine has many components that need to make network calls, as follows:

- **External Network Calls** - Replicated is a vendor service that we use for the Management Console of CircleCI. CircleCI requires Replicated to make an outside call to validate the license, check for updates, and download upgrades. Replicated also downloads docker, installs it on the local machine, and uses a Docker container to create and configure S3 buckets. GitHub Enterprise may or may not be behind the proxy, but github.com will need to go through the proxy.
- **Internal Network Calls**
    - If S3 traffic requires going through an HTTP proxy, CircleCI must pass proxy settings into the container.
    - The CircleCI instance on the Services machine runs in a Docker container, so it must to pass the proxy settings to the container to maintain full functionality.

Set up Service Machine Proxy Support

For a static installation, not on AWS, SSH into the Services machine and run the following code snippet with your proxy address:

```
echo '{"HttpProxy": "http://<proxy-ip:port>"}' |
sudo tee  /etc/replicated.conf
(cat <<'EOF'
HTTP_PROXY=<proxy-ip:port>
HTTPS_PROXY=<proxy-ip:port>

EOF
| sudo tee -a /etc/circle-installation-customizations
 sudo service replicated-ui stop; sudo service replicated stop;
 sudo service replicated-operator stop; sudo service replicated-ui start;
  sudo service replicated-operator start; sudo service replicated start
```

If you run in Amazon's EC2 service then you'll need to include `169.254.169.254` EC2 services as shown below:

```
echo '{"HttpProxy": "http://<proxy-ip:port>"}' |
sudo tee  /etc/replicated.conf
(cat <<'EOF'
HTTP_PROXY=<proxy-ip:port>
HTTPS_PROXY=<proxy-ip:port>
NO_PROXY=169.254.169.254,<circleci-service-ip>,
127.0.0.1,localhost,ghe.example.com
JVM_OPTS="-Dhttp.proxyHost=<ip> -Dhttp.proxyPort=<port>
-Dhttps.proxyHost=<proxy-ip> -Dhttps.proxyPort=<port) -Dhttp.nonProxyHosts=169.254.169.254|<circleci
-service-ip>|
127.0.0.1|localhost|ghe.example.com"

EOF
| sudo tee -a /etc/circle-installation-customizations
 sudo service replicated-ui stop; sudo service replicated stop;
 sudo service replicated-operator stop; sudo service replicated-ui start;
  sudo service replicated-operator start; sudo service replicated start
```

The above is not handled by by our enterprise-setup script and will need to be added to the user data for the Services Machine startup or done manually.

Corporate Proxies

When our instructions ask if you use a proxy, you will also be prompted to input the address. It is **very important** that you input the proxy in the following format: `<protocol>://<ip>:<port>`. If you miss any part, then `apt-get` won't work correctly and the packages won't download.

Nomad Client Configuration

External Network Calls

CircleCI uses `curl` and `awscli` scripts to download initialization scripts, along with jars from Amazon S3. Both `curl` and `awscli` respect environment settings, but if you have allowed traffic from Amazon S3 you should not have any problems.

Internal Network Calls

- CircleCI JVM:
  - Any connections to other Nomad Clients or the Services machine should be excluded from HTTP proxy
  - Connections to GitHub Enterprise should be excluded from HTTP proxy
- The following contains parts that may be impacted due to a proxy configuration:
  - Amazon EC2 metadata. This **should not** be proxied. If it is, then the machine will be misconfigured.
  - Amazon S3 traffic — note S3 discussion above
  - Amazon EC2 API - EC2 API traffic may need to be proxied. You would note lots of failures (timeout failures) in logs if the proxy setting is misconfigured, but it will not block CircleCI from functioning.

Nomad Client Proxy Setup

- If you are installing CircleCI Server on AWS using Terraform, you should add the below to your Nomad client launch configuration – these instructions should be added to `/etc/environment`.
- If you are using Docker refer to the Docker HTTP Proxy Instructions documentation.
- If you are running a static installation, add the following to the server before installation:

```
#!/bin/bash

(cat <<'EOF'
HTTP_PROXY=<proxy-ip:port>
HTTPS_PROXY=<proxy-ip:port>
NO_PROXY=169.254.169.254,<circleci-service-ip>,
127.0.0.1,localhost,ghe.example.com
JVM_OPTS="-Dhttp.proxyHost=<ip> -Dhttp.proxyPort=<port>
-Dhttps.proxyHost=<proxy-ip> -Dhttps.proxyPort=3128
-Dhttp.nonProxyHosts=169.254.169.254|<circleci-service-ip>|
127.0.0.1|localhost|ghe.example.com"
EOF
) | sudo tee -a /etc/environment

set -a
. /etc/environment
```

If your containers need to use a proxy server you will need to set the following schedulerer environment variables: `DOCKER_HTTP_PROXY`, `DOCKER_HTTPS_PROXY`, `NO_PROXY`, corresponding to those listed in the Docker instructions. This will ensure your containers have outbound/proxy access. For more information on creating configuration overrides, see the Customizations Guide.

Troubleshooting

Can't access the Management Console

If you cannot access the CircleCI Management Console, but the Services machine seems to be running, try to SSH tunnel into the machine by running the following, substituting your proxy address and the IP address of your Services machine:

```
ssh -L 8800:<address you want to proxy through>:8800 ubuntu@<ip_of_services_machine>
```

REPL time out

If you experience a timeout when connecting to the REPL, you will need to allow access, through your corporate proxy, to the domains of any Clojure library repositories that are required to download dependencies for running the REPL.

```
sudo su
docker exec -it frontend /bin/bash
lein repl :connect 6005
```

Refer to the error output for guidance on which repositories need to be granted access. The list will be different for each corporate proxy, but following is an example list:

- repo1.maven.org

- build.clojure.org
- clojars.org
- repo.clojars.org

Data Persistence

Contact CircleCI Support to discuss externalizing services for data persistence.

Authentication

This document describes the various ways users of your CircleCI Server installation can get access and authenticate their accounts. CircleCI currently supports OAuth and LDAP as authentication methods.

OAuth with GitHub/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 application - for example `<your-circleci-hostname>.com` – and it will prompt them to set up an account by running through the GitHub/GitHub Enterprise OAuth flow. The application will then redirect them to the CircleCI login screen.

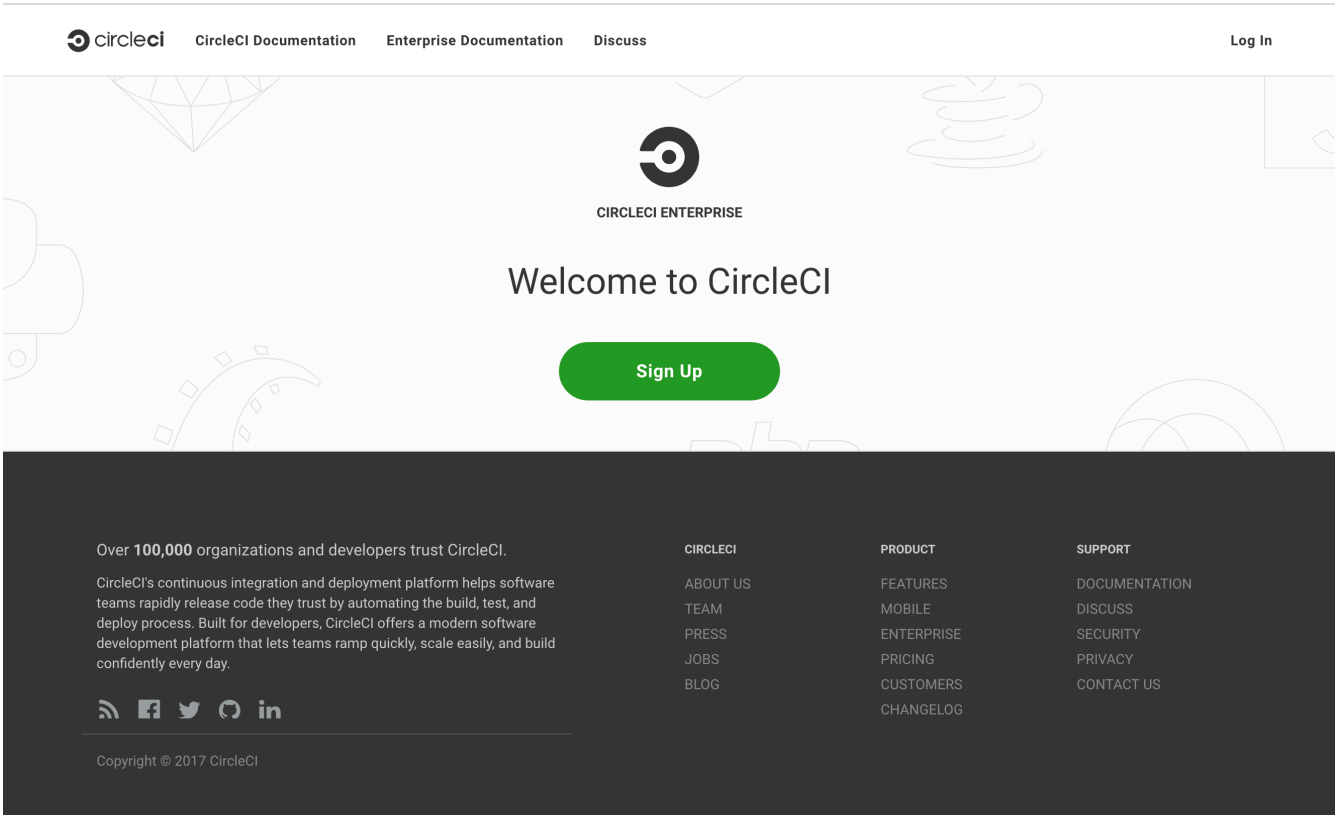


Figure 10. CircleCI Server Login Screen

LDAP

As an alternative to the OAuth/GitHub option, you can choose LDAP authentication. Many organizations use a LDAP server to centralize all their identity information in a single place. This section describes how to enable, configure, and test CircleCI to authenticate users with OpenLDAP or Active Directory credentials.

Enabling LDAP will **disable** other authentication methods. CircleCI **does not recommmend to turn on LDAP Authentication for existing installations that previously had users authenticating with GitHub**. Consider contacting your account team if you need to switch to LDAP for an existing installation.

Prerequisites

- Install and configure your OpenLDAP server or Active Directory.
- Configure GitHub Enterprise or GitHub.com to be the source of organizations and projects to which users have access.
- Install a new instance of CircleCI Server with no existing users.

Configure LDAP Authentication

This section provides information to configure LDAP. Below is an example configuration to give an idea of the information types required. The example shows OpenLDAP in use but the settings for Active Directory are comparable:

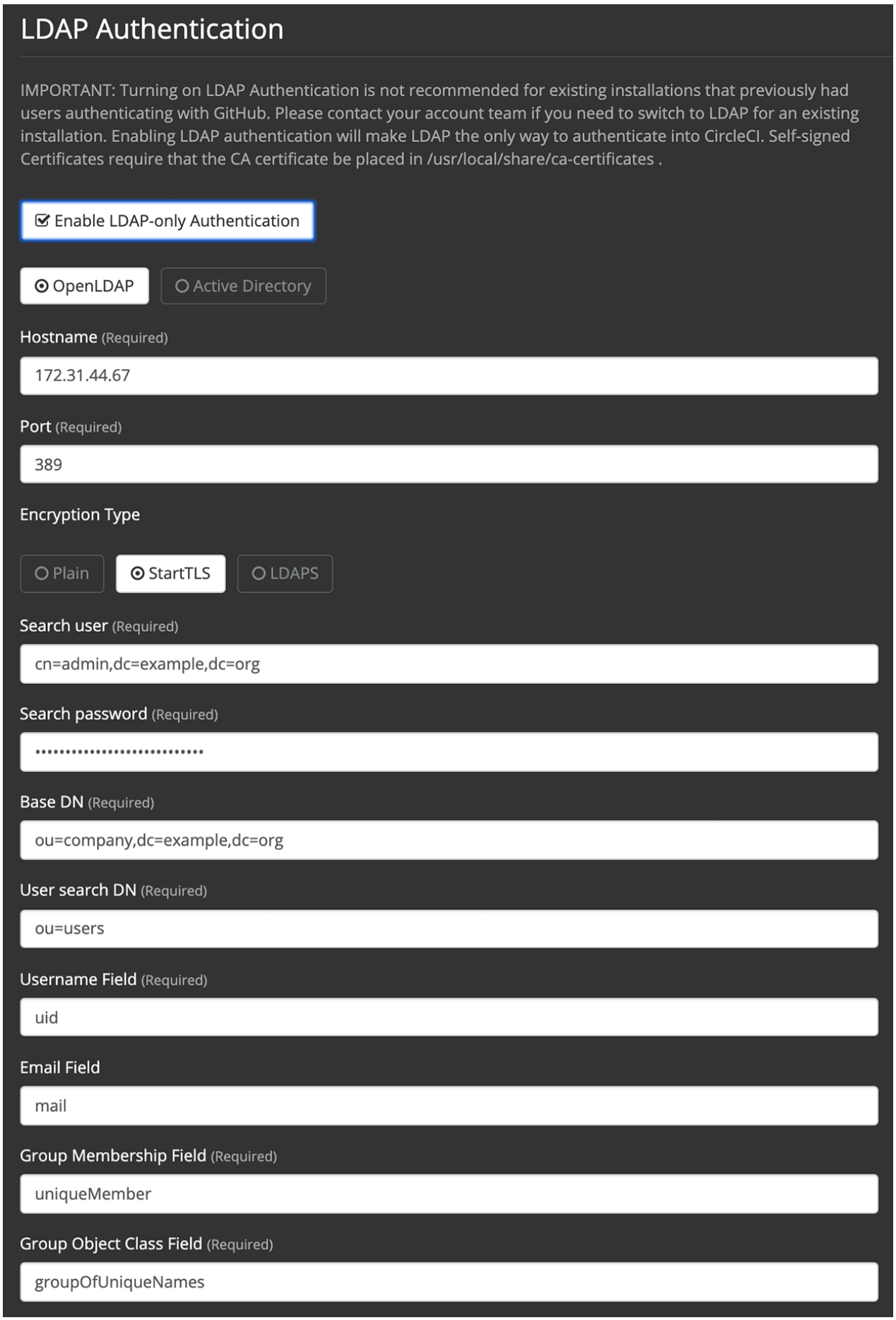

LDAP Authentication
IMPORTANT: Turning on LDAP Authentication is not recommended for existing installations that previously had users authenticating with GitHub. Please contact your account team if you need to switch to LDAP for an existing installation. Enabling LDAP authentication will make LDAP the only way to authenticate into CircleCI. Self-signed Certificates require that the CA certificate be placed in /usr/local/share/ca-certificates .
Enable LDAP-only Authentication
OpenLDAP
Active Directory
Hostname (Required)
172.31.44.67
Port (Required)
389
Encryption Type
Plain
StartTLS
LDAPS
Search user (Required)
cn=admin,dc=example,dc=org
Search password (Required)
Base DN (Required)
ou=company,dc=example,dc=org
User search DN (Required)
ou=users
Username Field (Required)
uid
Email Field
mail
Group Membership Field (Required)
uniqueMember
Group Object Class Field (Required)
groupOfUniqueNames

Figure 11. LDAP Config Example

These are the steps to configure LDAP in the CircleCI Server Management Console:

1. Verify access over the LDAP/AD ports to your LDAP/AD servers.
2. Log in as administrator to the Management Console for your newly installed CircleCI instance.
3. Navigate to the Settings page (for example `<your-circleci-hostname>.com:8800`) and scroll down to check the Enable LDAP-only Authentication button. Select either OpenLDAP or Active Directory.
4. Fill in your LDAP instance Hostname and port number.
5. Select the encryption type (plain text is insecure and thus not recommended).
6. Fill in the Search user field with the Fully Distinguished Name for a user you authorize to perform search queries over a LDAP database. Example: `cn=<admin>,dc=<example>,dc=<org>`.
7. Fill in the Search password field with the LDAP password for a user from the previous step.
8. Fill in the Base DN field with a Distinguished Name for a point in the directory from where CircleCI will be looking for users/groups. Example: `ou=company,dc=example,dc=org`
9. Fill in the User search DN field with a Relative Distinguished Name for a point in a directory where CircleCI will find users. Should be relative to the Base DN provided above. Example: `ou=users`.
10. Fill in the Username field with a name of an attribute that is the source of usernames for Logging In. Example: `uid` (in this case users will have to use their UID for logging in) or `mail` (in this case users will be using emails for logging in).
11. Fill in the Email field with the name of an attribute that is the source of a user email. Example: `mail`
12. Fill in the Group Membership field with a name of an attribute that is the user membership in a particular group. Example: `uniqueMember`.
13. Fill in the Group Object Class field with a name of an Object Class that will identify DN as a group. Example: `groupOfUniqueNames`
14. (Optional) Fill in the Test username and Test password fields with a test email and password for an LDAP user you want to test. This is a 3rd party infrastructure and this test option is not always reliable.
15. Save the settings.

Known Issue: Grant Admin Access to User

If you are using LDAP authentication, use one of the following methods to grant admin privileges to a user. Admin privileges should be automatically granted to the first user accessing a project, but currently there is a known issue preventing this from happening when LDAP authentication is in use.

Only one of the following methods should be necessary to grant admin access. If you are unfamiliar with using the REPL, consider contacting customer support before running these steps.

For a given user by LDAP user name (prior to GitHub account connection, or if they have no GitHub account):

```
(-> (circle.repl.mongo/fetch :users :domain-model :where {:login "the-ldap-username"
:first_vcs_authorized_client_id nil} :limit 1)
    (first)
    (circle.model.user/set-fields! {:admin "all"})
    (:analytics-id)
    (circle.services.domain/delete-user-cache))
```

By GitHub user name (after GitHub account connection, which replaces prior :login value):

```
(-> (circle.repl.mongo/fetch :users :domain-model :where {:login "the-github-username"} :limit 1)
    (first)
    (circle.model.user/set-fields! {:admin "all"})
    (:analytics-id)
    (circle.services.domain/delete-user-cache))
```

By analytics ID

```
(-> (circle.model.user/find-one-by-analytics-id "3b35037c-6eb3-4e41-88e2-3913b2f43d96")
    (circle.model.user/set-fields! {:admin "all"})
    (:analytics-id)
    (circle.services.domain/delete-user-cache))
```

User Interaction

After setting up LDAP, a user who logs in to CircleCI will be redirected to the Accounts page with a Connect button that they must use to connect their GitHub account. After they click Connect, an LDAP section with their user information (for example, their email) on the page will appear and they will be directed to authenticate their GitHub account. After authenticating their GitHub account users are directed to the **Job page** to use CircleCI.

A user who has authenticated with LDAP and is then removed from LDAP/AD will be able to access CircleCI as long as they stay logged in (because of cookies). As soon as the user logs out or the cookie expires, they will not be able to log back in. GitHub permissions define users' ability to see projects or to run builds. Therefore, if GitHub permissions are synced with LDAP/AD permissions, a removed LDAP/AD user will automatically lose authorization to view or access CircleCI as well.

Troubleshooting

Troubleshoot LDAP server settings with LDAP search as follows:

```
ldapsearch -x LLL -h <ldap_address_server>
```

VM Service

This section outlines how to set up and customize VM service for your CircleCI installation. VM Service controls how `machine` executor (Linux and Windows images) and remote Docker jobs are run.

The information on this page is only applicable to installations on AWS. Please contact your CircleCI account representative to request VM service guidance for a static installation.

Overview

VM service enables users of CircleCI Server, installed on AWS, to run jobs using the Remote Docker Environment and the `machine` executor.

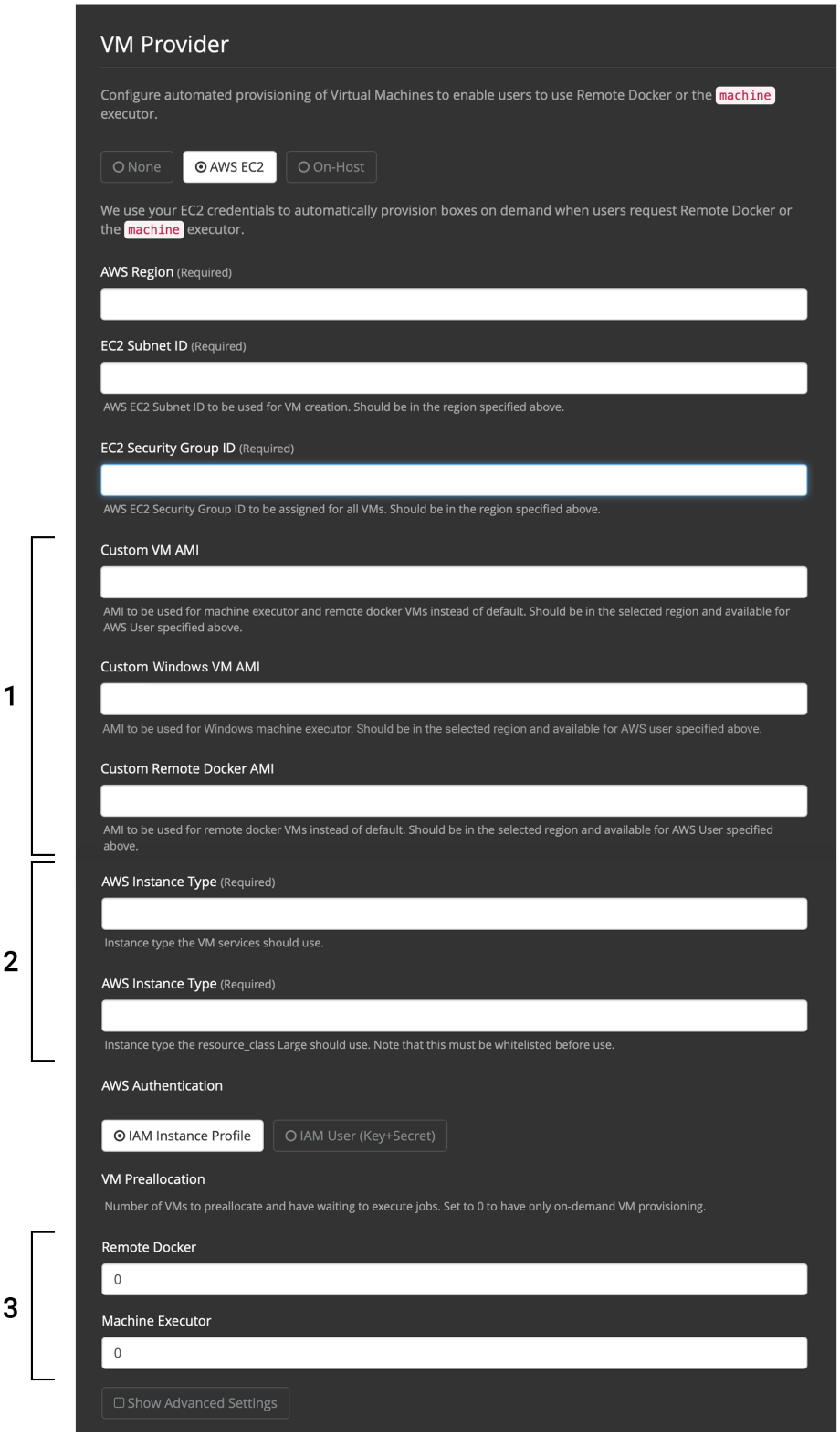


Figure 12. VM Service Settings

Any changes to management console settings require downtime while the CircleCI application restarts.

The following sections will run through the settings and options displayed in the VM Service screenshot

show above.

To configure VM service, it is best practice to select the AWS EC2 option in the Management Console Settings. This will allow CircleCI to run remote Docker and `machine` executor jobs using dedicated EC2 instances.

1. Supply AMIs

You can provide custom Amazon Machine Image (AMIs) for VM service, as described in the sections below. If you do not provide any custom images, all `machine` executor and remote Docker jobs will be run on instances built with one of our default AMIs (listed below), which have Ubuntu 16.04, Docker version 18.06.3 and a selection of common languages, tools, and frameworks. See the `picard-vm-image` branch of our image-builder repository for details. To run Windows jobs you must supply a Windows AMI, without this Windows jobs will fail to run.

Default VM service Linux AMIs

- Ap-northeast-1:ami-0e49af0659db9fc5d
- Ap-northeast-2:ami-03e485694bc2da249
- Ap-south-1:ami-050370e57dfc6574a
- Ap-southeast-1:ami-0a75ff7b28897268c
- Ap-southeast-2:ami-072b1b45245549586
- Ca-central-1:ami-0e44086f0f518ad2d
- Eu-central-1:ami-09cbcfe446101b4ea
- Eu-west-1:ami-0d1cbc2cc3075510a
- Eu-west-2:ami-0bd22dcdc30fa260b
- Sa-east-1:ami-038596d5a4fc9893b
- Us-east-1:ami-0843ca047684abe87
- Us-east-2:ami-03d60a35576647f63
- Us-west-1:ami-06f6efb13d9ccf93d
- Us-west-2:ami-0b5b8ad02f405a909

Customizing and Creating VM Service Images

Customizing the VM service images for your installation will allow you to specify versions of Docker and Docker Compose, as well as install any additional dependencies that may be part of your CI/CD pipeline. You can create separate AMIs for jobs that use remote Docker or the `machine` executor, and for `machine` you can specify separate AMIs for Linux and Windows. It's worth noting that if you choose not to customize the base Linux image, developers will likely need to configure jobs to run additional install and update steps on every commit as part of each project's `config.yml` file.

From Server v2.18, you can either provide a single custom Linux AMI to use for both `machine` and remote Docker jobs using just the field marked '1' below, or, by providing a second custom AMI in the field marked '2', you can use different settings for each.

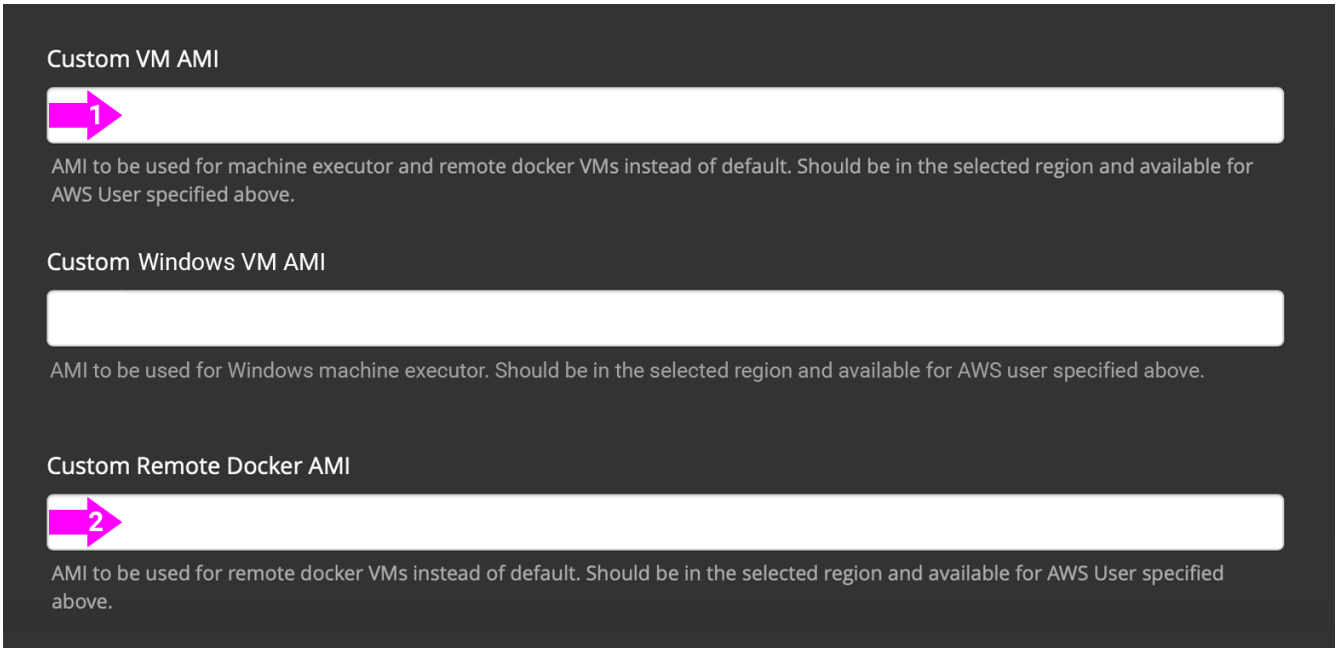


Figure 13. Custom VM Service Images

Custom Linux AMI

Prerequisites

- Packer (https://packer.io/intro/getting-started/install.html)
- AWS Access Key ID and Secret Access Key

Creating a Custom Linux AMI

1. Clone our image builder repo: https://github.com/circleci/image-builder/tree/picard-vm-image
2. Open `aws-vm.json` in your editor. This provides a baseline template for building an AMI with Packer. An AWS access key ID and secret access key are required to upload. You can find more information about managing AWS authentication with Packer here. If the baseline template is too limited, you can find additional AWS configuration options here.
3. (Optional) Consider restricting the `ami_groups` to only within your organization. See the [Packer documentation](https://packer.io/docs/builders/amazon-ebs.html#ami_groups) for more information on AMI groups.
4. We provide a list of [pre-configured dependencies](https://github.com/circleci/image-builder/blob/picard-vm-image/provision.sh). You can customize the provision.sh script to meet the needs of your environment.
5. Run `packer build aws-vm.json`

Once your AMI(s) have been created, copy the AMI ID(s) into the relevant field shown in the screenshot above.

Creating a Windows AMI

Introduced in CircleCI Server v2.18.3

Creating a Windows image and specifying it under the VM Service settings lets your users run jobs on dedicated Windows VMs. To create your Windows image run through the steps listed in our image builder repo, then copy the generated AMI ID and paste into the Custom Windows VM AMI field in your Management Console settings, under VM Provider (for example, `<your-hostname.com:8800/settings>`).

Windows images are built on CircleCI, so we suggest you run through this process once your installation is up and running. Alternatively you can use any other CircleCI account – including on our managed Cloud service – to generate the image.

2. Define Instance Types

There are two fields for defining the AWS instance types you wish to use. The first is for the default instance type, and the second is to set the instance type to use when a Job specifies the `large` resource class.

3. On Demand and Preallocated Instances

Remote Docker and `machine` executor instances are spun up on demand. It is also possible to preallocate instances to remain up and running, ready for remote Docker and `machine` jobs to be run (see the last two fields in figure 1).

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

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

Job and Instance Management

Jobs run using the remote Docker Environment, or the `machine` executor are scheduled and dispatched by the Nomad server to your Nomad clients and passed on to remote Docker or `machine` from there. This means jobs run on remote Docker and the `machine` executor can be monitored in the usual way, using the Nomad CLI. See our Introduction to Nomad Cluster Operation for more about Nomad commands and terminology.

A cron job is scheduled to cycle all default and preallocated instances at least once per day to ensure instances do not end up in an unworkable state.

Accessing Remote Docker and `machine` instances

By default, private IP addresses are used to communicate with VM service instances. If you need to grant wider access, for example, to allow developers SSH access, this can be set using the checkbox in the VM Provider Advanced Settings.

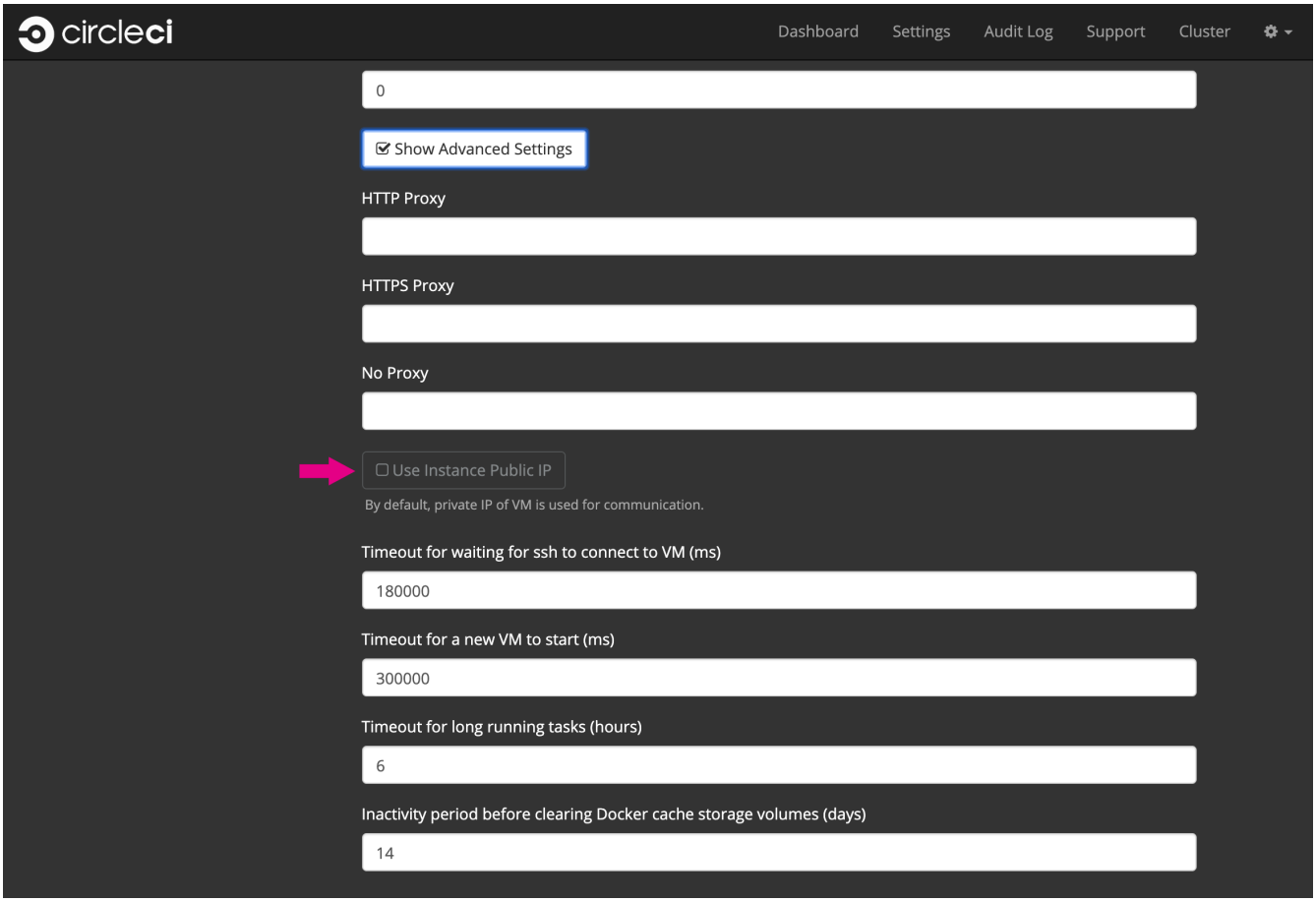


Figure 14. Allowing Access to VM Service Instances

Running GPU Executors

This document outlines how to run GPU (graphics processing unit) machine executors using CircleCI Server.

Prerequisites

Configure the `vm-service` in the Replicated management console to start a GPU-enabled instance.

Overview

Run the following commands on any Nvidia GPU-enabled instance. The following example uses CUDA 8.0, but you can use any CUDA runtime version supported by your GPU instance.

```
wget https://developer.nvidia.com/compute/cuda/8.0/prod/local_installers/cuda-repo-ubuntu1404-8-0-local_8.0.44-1_amd64-deb
sudo apt-get update
export OS_RELEASE=$(uname -r)
sudo apt-get install -y linux-image-extra-$OS_RELEASE linux-headers-$OS_RELEASE linux-image-$OS_RELEASE
sudo dpkg -i cuda-repo-ubuntu1404-8-0-local_8.0.44-1_amd64-deb
sudo apt-get update
sudo apt-get --yes --force-yes install cuda
nvidia-smi
```

`nvidia-smi` is only required for testing purposes. After you install the CUDA driver in Step 7 you should be good to go!

Adding GPU Steps to an AMI

To avoid start up time associated with the above steps, they may be included in an AMI by following the instructions in the Configuring VM Service documentation.

Setting Up Certificates

This document provides a script for using a custom Root Certificate Authority and the process for using an Elastic Load Balancing (ELB) certificate.

Using a Custom Root CA

Any valid certificates added to the following path will be trusted by CircleCI services: `usr/local/share/ca-certificates/`

The following example `openssl` command is one way of placing the certificate. It is also possible to pull a certificate from a vault/PKI solution within your company.

Some installation environments use internal Root Certificate Authorities (RCAs) for encrypting and establishing trust between servers. If you are using a customer Root certificate, you will need to import and mark it as a trusted certificate at CircleCI GitHub Enterprise instances. CircleCI will respect such trust when communicating with GitHub and webhook API calls.

CA Certificates must be in a format understood by Java Keystore, and include the entire chain.

The following script provides the necessary steps:

```
GHE_DOMAIN=github.example.com

# Grab the CA chain from your GitHub Enterprise deployment.
openssl s_client -connect ${GHE_DOMAIN}:443 -showcerts < /dev/null | sed -ne '/-BEGIN CERTIFICATE-/,/-END CERTIFICATE-/p' > /usr/local/share/ca-certificates/ghe.crt
```

Then, navigate to the system console at port 8800 and change the protocol to upgraded. You can change the protocol to HTTPS (TLS/SSLEnabled) setting and restart the services. When trying Test GitHub Authentication you should get Success now rather than x509 related error.

Setting up ELB Certificates

CircleCI requires the following steps to get ELB certificates working as your primary certs. The steps to accomplish this are below. You will need certificates for the ELB and CircleCI Server as described in the following sections.

Opening the port for HTTP requests will allow CircleCI to return a HTTPS redirect.

From Server v2.19.02 and up, Classic Load Balancer (CLB) is no longer supported.

1. Open the following ports on your ELB:

Load Balancer Protocol	Load Balancer Port	Instance Protocol	Instance Port	Cipher	SSL Certificate
HTTP	80	HTTP	80	N/A	N/A
SSL	443	SSL	443	Change	your-cert
SSL	3000	SSL	3000	Change	your-cert
HTTPS	8800	HTTPS	8800	Change	your-cert
SSL	8081	SSL	8081	Change	your-cert
SSL	8082	SSL	8082	Change	your-cert

2. Add the following security group on your ELB:

The sources below are left open so that anybody can access the instance over these port ranges. If that is not what you want, then feel free to restrict them. Users will experience reduced functionality if your stakeholders are using IP addresses outside of the Source Range.

Type	Protocol	Port Range	Source
SSH	TCP	22	0.0.0.0
HTTPS	TCP	443	0.0.0.0
Custom TCP Rule	TCP	8800	0.0.0.0
Custom TCP Rule	TCP	64535-65535	0.0.0.0

3. Next, in the management console for CircleCI, upload a valid certificate and key file to the `Privacy` Section. These don't need to be externally signed or even current certs as the actual cert management is done at the ELB. But, to use HTTPS requests, CircleCI requires a certificate and key in which the "Common Name (FQDN)" matches the hostname configured in the admin console.
4. It is now possible to set your GitHub Authorization Callback to `https` rather than `http`.

Using Self-Signed Certificates

Because the ELB does not require a *current* certificate, you may choose to generate a self-signed certificate with an arbitrary duration.

1. Generate the certificate and key using openssl command `openssl req -newkey rsa:2048 -nodes -keyout key.pem -x509 -days 1 -out certificate.pem`
2. Provide the appropriate information to the prompts.

The Common Name provided must match the host configured in CircleCI.

3. Save the certificate.pem and key.pem file locally.

Setting up TLS/HTTPS on CircleCI Server

You may use various solutions to generate valid SSL certificate and key file. Two solutions are provided below.

Using Certbot

This section describes setting up TLS/HTTPS on your Server install using Certbot by manually adding a DNS record set to the Services machine. Certbot generally relies on verifying the DNS record via either port 80 or 443, however this is not supported on CircleCI Server installations as of 2.2.0 because of port conflicts.

1. Stop the Service CircleCI Server Management Console (`<circleci-hostname>.com:8800`).
2. SSH into the Services machine.
3. Install Certbot and generate certificates using the following commands:

```
sudo apt-get update
sudo apt-get install software-properties-common
sudo add-apt-repository ppa:certbot/certbot
sudo apt-get update
sudo apt-get install certbot
certbot certonly --manual --preferred-challenges dns
```

4. You will be instructed to add a DNS TXT record.
5. After the record is successfully generated, save `fullchain.pem` and `privkey.pem` locally.

If you are using Route 53 for your DNS records, adding a TXT record is straightforward. When you're creating a new record set, be sure to select type → TXT and provide the appropriate value enclosed in quotes.

Adding the certificate to CircleCI Server

Once you have a valid certificate and key file in `.pem` format, you must upload it to CircleCI Server.

1. To do so, navigate to `hostname:8800/console/settings`
2. Under "Privacy" section, check the box for "SSL only (Recommened)"
3. Upload your newly generated certificate and key
4. Click "Verify TLS Settings" to ensure everything is working
5. Click "Save" at the bottom of the settings page and restart when prompted

More information is available here.

Ensure the hostname is properly configured from the Management Console (`<circleci-hostname>.com:8800`) **and** that the hostname used matches the DNS records associated with the TLS certificates.

Make sure the Auth Callback URL in GitHub/GHE matches the domain name pointing to the Services

machine, including the protocol used, for example `https://info-tech.io/`.

Managing User Accounts in Server Installations

This section provides information to help system administrators of self-hosted CircleCI Server installations manage accounts for their users. 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. This overview provides the active user count and the total number of licensed users.

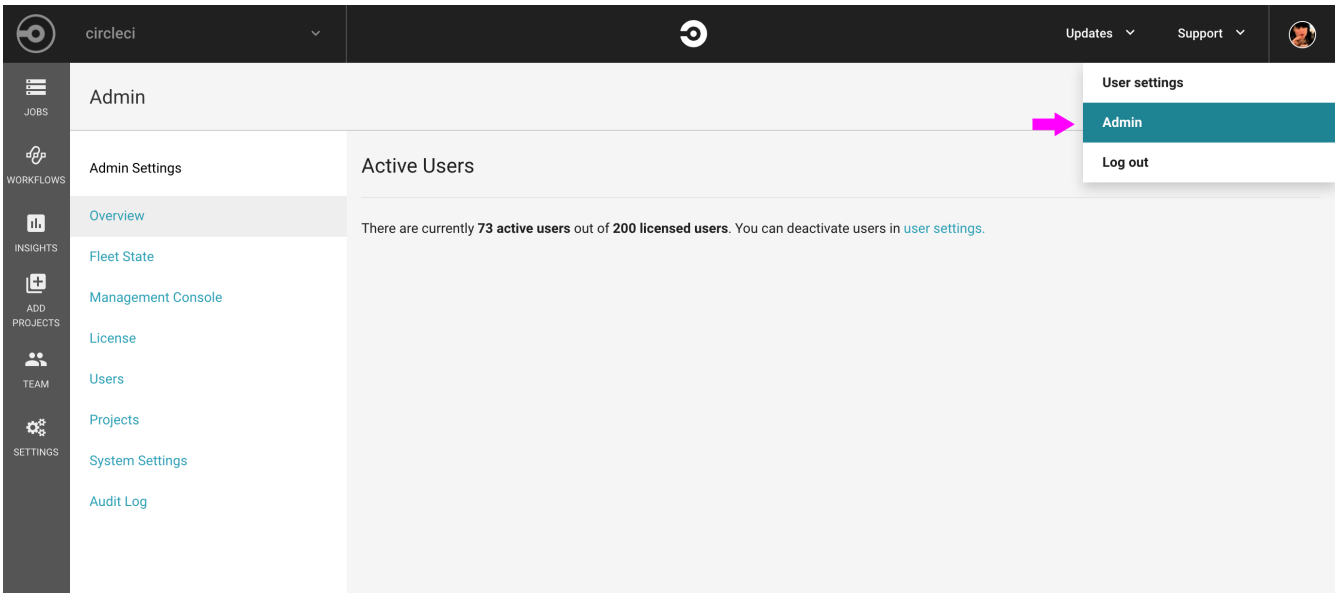


Figure 15. Admin Settings – Account Overview

Suspending 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

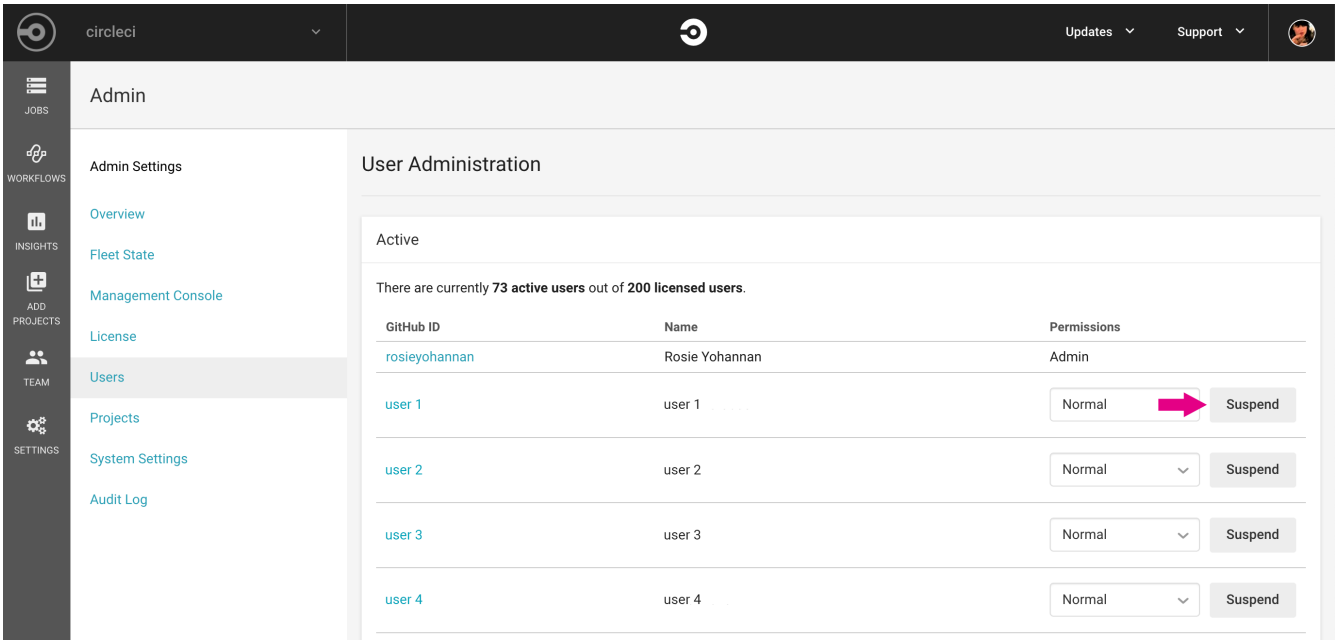


Figure 16. Suspending an Account

Reactivating a Suspended User 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

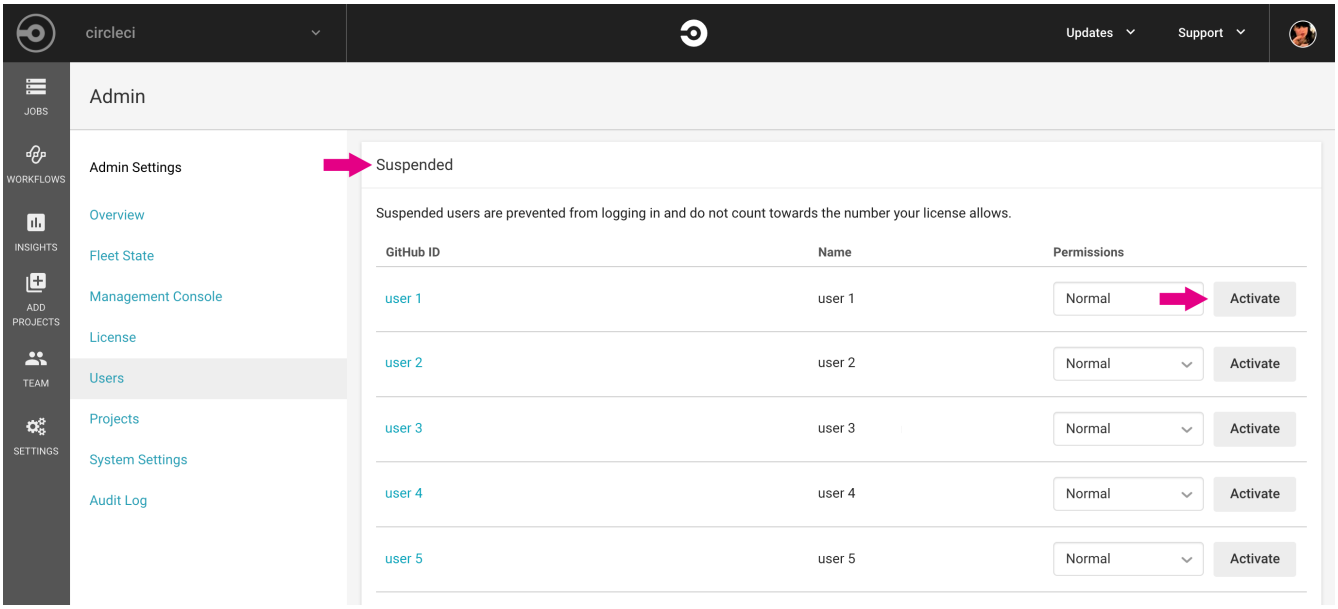


Figure 17. Rectivate Existing Users

Controlling Account Access

Any user associated with your GitHub.com or GitHub Enterprise organization can create a user account for your CircleCI installation. In order to control who has access, you can 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`

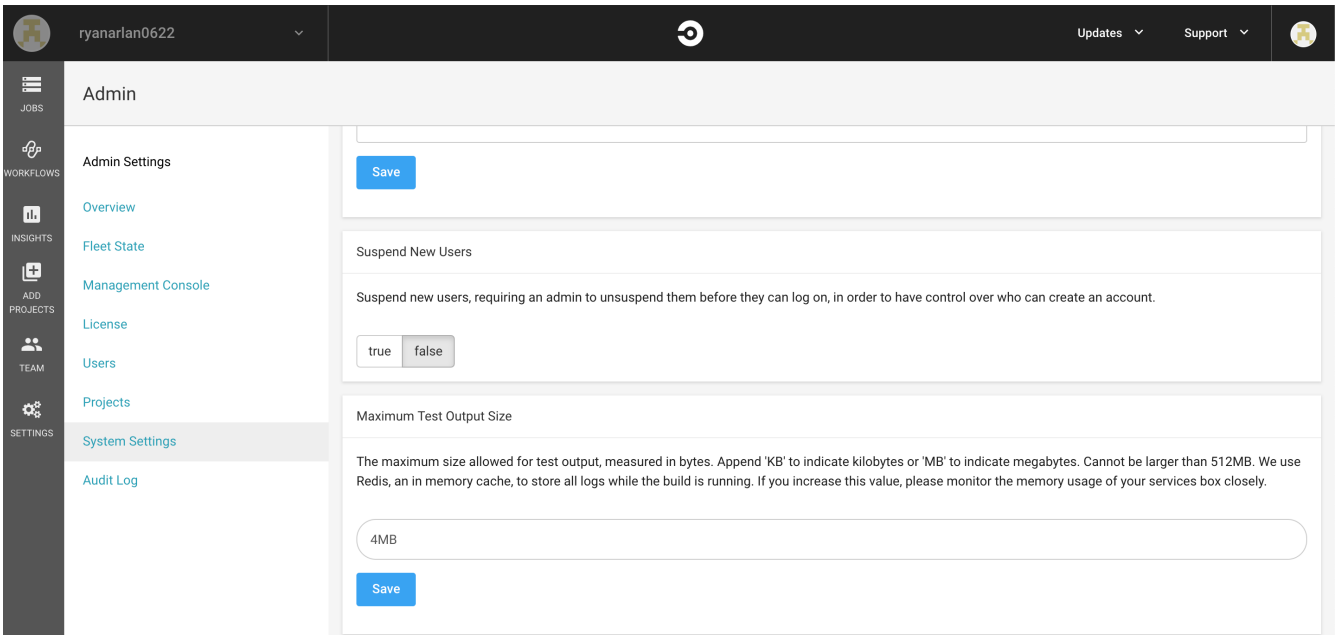


Figure 18. Auto Suspend New Users

Activating a Suspended New User Account

To activate an **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

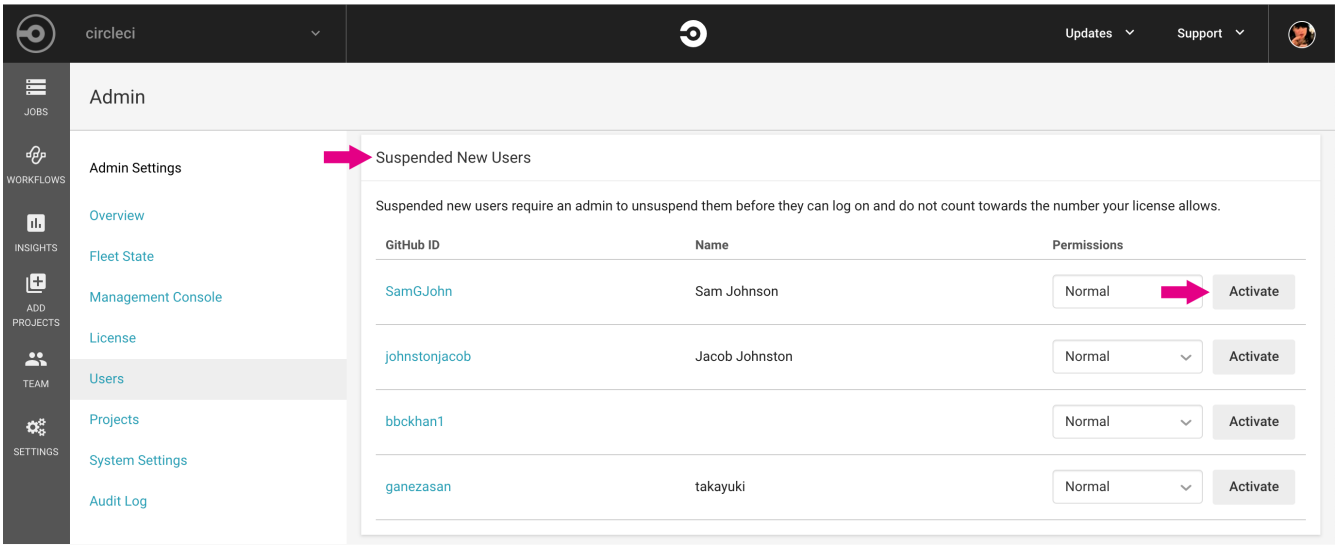


Figure 19. Activate a Suspended New User

Limit User Registrations 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

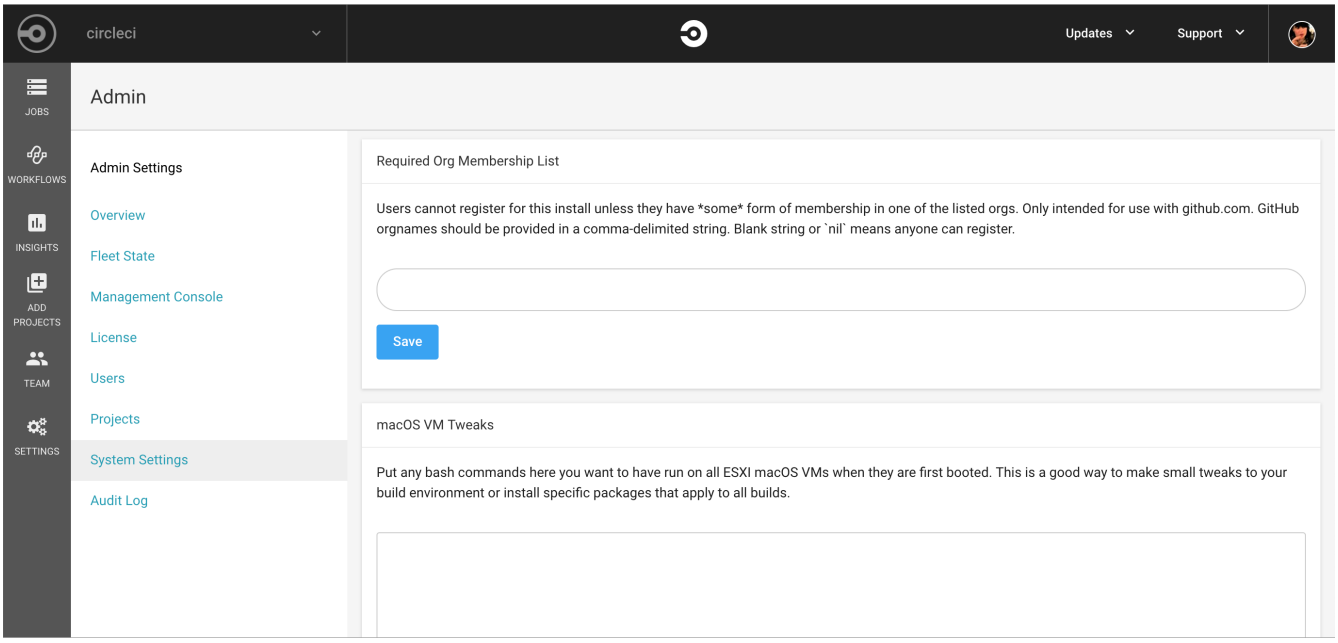


Figure 20. Organization Membership

Any form of organization membership is within the scope of this approval feature, and it does not stop users from running builds associated with other organizations they may belong to.

Full User List

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

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

Deleting a 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)
```

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

Allow Unsafe Content types

If you would like to allow content types that are not included in the list above, follow these steps:

1. Navigate to the CircleCI Management Console (for example, `<your-circleci-hostname>:8800/settings`) and select Settings from the menu bar.
2. Scroll down to find the Artifacts section.
3. Select Serve Artifacts with Unsafe Content-Types.

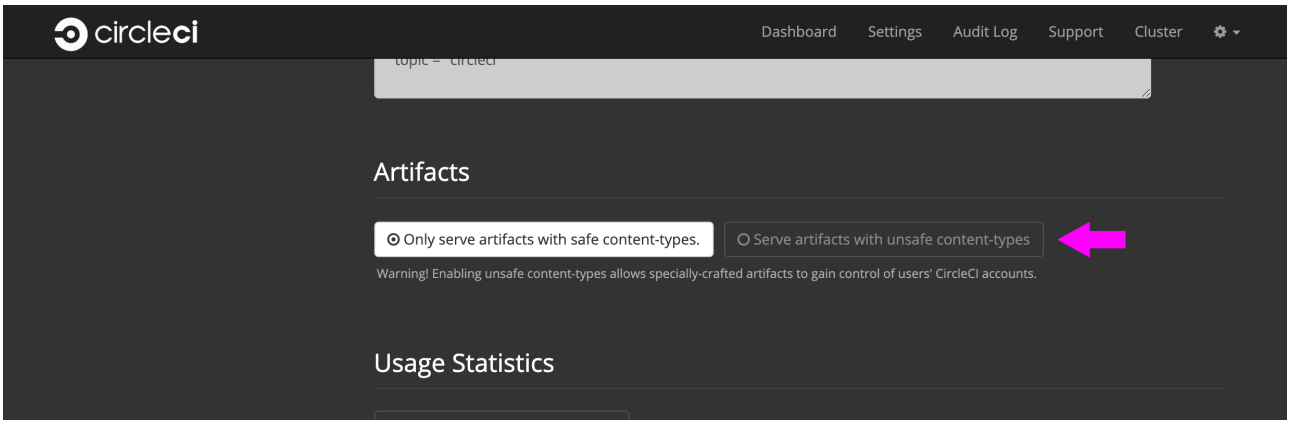


Figure 21. Allow Unsafe Content Types

4. Click Save at the bottom of the page and Restart Now in the pop-up to save your changes and restart the console.

Any change to the settings within the Management Console will incur downtime as the console will need to be restarted.

Enabling Usage Statistics

This chapter is for System Administrators who want to automatically send some aggregate usage statistics to CircleCI. Usage statistics data enhances visibility into CircleCI installations and is used to better support you and ensure a smooth transition from CircleCI 1.0 to CircleCI 2.0.

To opt-in to this feature, navigate to your Management Console settings (e.g. circleci-hostname.com:8800/settings) and scroll down to Usage Statistics. Enable the radio button labeled Automatically send some usage statistics to CircleCI, as shown below.

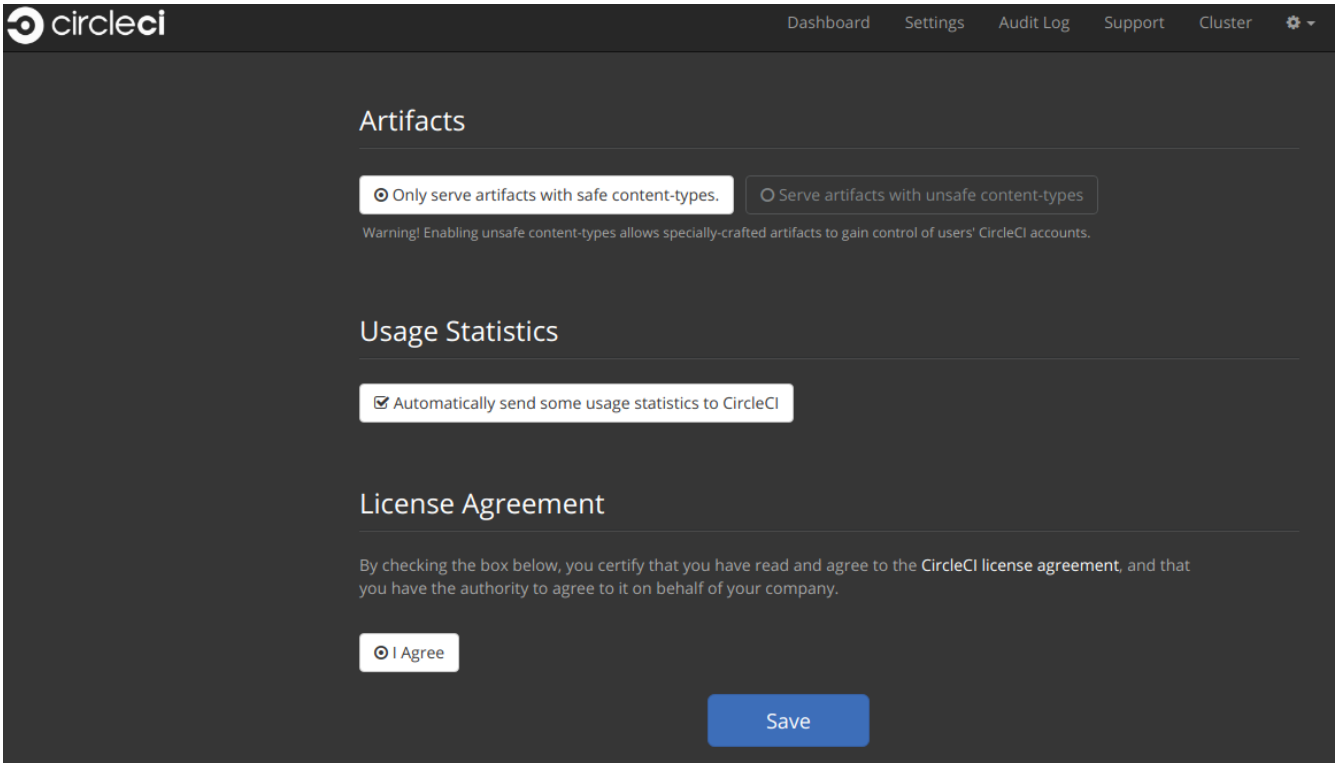


Figure 22. Usage Statistics Settings

Detailed Usage Statistics

The following sections provide information about the usage statistics CircleCI will gather when this setting is enabled.

Weekly Account Usage

Name	Type	Purpose
account_id	UUID	*Uniquely identifies each vcs account*
usage_current_macos	minutes	*For each account, track weekly builds performed in minutes.*
usage_legacy_macos	minutes	
usage_current_linux	minutes	
usage_legacy_linux	minutes	

Weekly Job Activity

Name	Type	Purpose
utc_week	date	*Identifies which week the data below applies to*
usage_oss_macos_legacy	minutes	*Track builds performed by week*
usage_oss_macos_current	minutes	
usage_oss_linux_legacy	minutes	
usage_oss_linux_current	minutes	
usage_private_macos_legacy	minutes	
usage_private_macos_current	minutes	
usage_private_linux_legacy	minutes	
usage_private_linux_current	minutes	
new_projects_oss_macos_legacy	sum	*Captures new Builds performed on 1.0. Observe if users are starting new projects on 1.0.*
new_projects_oss_macos_current	sum	
new_projects_oss_linux_legacy	sum	
new_projects_oss_linux_current	sum	
new_projects_private_macos_legacy	sum	
new_projects_private_macos_current	sum	
new_projects_private_linux_legacy	sum	
new_projects_private_linux_current	sum	
projects_oss_macos_legacy	sum	*Captures Builds performed on 1.0 and 2.0. Observe if users are moving towards 2.0 or staying with 1.0.*
projects_oss_macos_current	sum	
projects_oss_linux_legacy	sum	
projects_oss_linux_current	sum	
projects_private_macos_legacy	sum	
projects_private_macos_current	sum	
projects_private_linux_legacy	sum	

Name	Type	Purpose
projects_private_linux_current	sum	

Accessing Usage Data

If you would like programatic access to this data in order to better understand your users you may run this command from the Services VM.

```
docker exec usage-stats /src/builds/extract
```

Security and Privacy

Please reference exhibit C within your terms of contract and our standard license agreement for our complete security and privacy disclosures.

Configuring the JVM Heap Size

The JVM heap size is configurable for the following containers: `frontend`, `test-results`, `output-processing` and `contexts-service`. You might want to consider increasing the heap size if you see "out of memory" errors, such as: `Terminating due to java.lang.OutOfMemoryError: Java heap space`.

Setting up

To be able to configure the JVM_HEAP_SIZE value for each container, you will first need to create customizations files on your services machine.

1. Create customizations files:

```
/etc/circleconfig/frontend/customizations
/etc/circleconfig/test-results/customizations
/etc/circleconfig/output-processor/customizations
/etc/circleconfig/contexts-service/customizations
```

2. In each customization file add the line below to export your desired JVM heap size:

```
export JVM_HEAP_SIZE=2g
```

3. Stop and restart CircleCI application from the Management Console dashboard (for example, your-circleci-hostname.com:8800)

Verify customization is applied

Once your installation has successfully restarted, you can confirm the configured value was applied correctly by running the following REPL commands per container:

- `frontend`

```
sudo docker exec -it frontend lein repl :connect 6005
```

- `test-results`

```
sudo docker exec -it test-results lein repl :connect 2719
```

- `output-processing`

```
sudo docker exec -it picard-output-processor lein repl :connect 6007
```

And following are the outputs you should see:

```
(System/getenv "JVM_HEAP_SIZE") ;; should return what you have set above
```

```
(-> (java.lang.Runtime/getRuntime) (.maxMemory)) ;; return value should match with JVM_HEAP_SIZE
```

SSH Rerun Architecture in Server

This section describes how SSH reruns work within installations of CircleCI Server. This information is designed to help system administrators have an overview to help with considerations when customizations are made to an installation. This guide is also here to help with debugging in the event of a problem with SSH reruns in an installation.

Rerunning a Job with SSH

For an overview of the purpose of SSH reruns in CircleCI, see the Debugging with SSH guide. The image below describes the IP addresses and ports used to restart jobs within VMs and containers when a rerun with SSH is requested.

The default situation for a Server installation with no customizations is for the `outer` agent to use AWS metadata to select the instance's public IP address.

If any networking customizations are made from the default that either block the metadata endpoint or make the public IP unsuitable, then creating a file at `/etc/circleci/public-ipv4` can be used as an override. This file should be created on each Nomad Client, usually via a boot script, and be populated with the IP address that should be used to connect. As an example, the following script uses the instance private IP from AWS metadata:

```
#!/bin/sh
PRIVATE_IP="$(curl http://169.254.169.254/latest/meta-data/local-ipv4)"
mkdir -p /etc/circleci
echo $PRIVATE_IP | tee /etc/circleci/public-ipv4
```

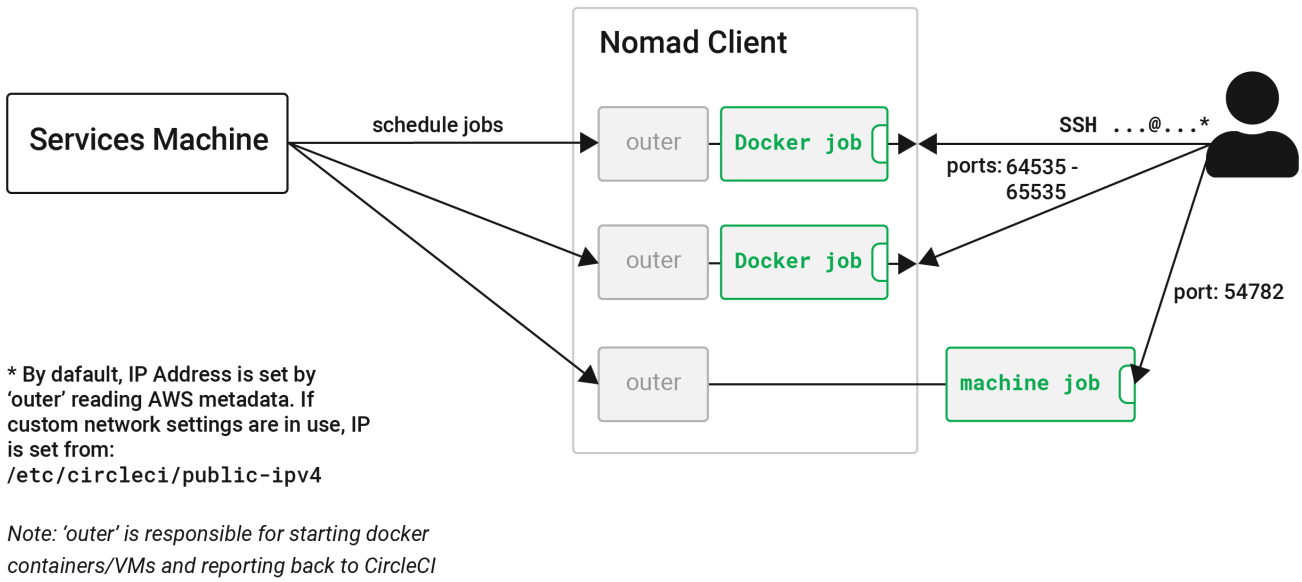


Figure 23. SSH Rerun Architecture

Maintenance

This chapter describes system checks and the basics of user management.

System Checks

When are executor instances created and destroyed?

Answer: CircleCI creates a new instance for each job. The instance will be destroyed at the end of the job. However, given that cloud instance creation may take significant time (~1 to 3 minutes), CircleCI offers a pre-scale option, where a set number of instances will be created in anticipation of demand. These will be killed at the end of the job. The number of pre-scaled instances is configured in the settings section of the Management Console. At any given time, CircleCI expects to have a base of pre-scaled instances and the required instances to service current job load.

When are executor instances reused?

Answer: Machine executor VMs never get reused for multiple jobs. EBS Volumes are reused for multiple jobs, but only get shared among jobs within the same project.

How are EBS volumes managed?

Answer: Since docker layers can be large (GBs), CircleCI prefers caching by using attached EBS volumes to using an object storage (for example, S3). Volumes are created when a job is configured to use docker layer caching (for example, set `docker_layer_caching: true` in config). **Note:** For docker layer caching to work, you **cannot** use preallocated instances. You must set the remote docker and/or machine executor (depending on which one you want to use DLC, or both) to 0 in the replicated settings for "on-demand" instances. Otherwise, DLC will not work.

CircleCI reuses any existing available volume for that job project. If there is none (or all existing volumes are busy), CircleCI creates a new volume for the project. Volumes are associated with a project. No two project jobs can share an EBS volume for security reasons. CircleCI deletes EBS volumes in few circumstances (for example, when there is a risk of running out of disk space).

Can the amount of EBS volumes and EC2 instances be bounded?

Answer: Not at this time. You may utilize the metrics provided to alert when reaching a specific threshold.

How do you prevent executors from existing indefinitely?

Answer: A process runs that periodically detects and stops any leaked VMs (for example, a task completed but it's VM is running for over N hours). You may also manually inspect instances that have been running for over 24 hours (CircleCI currently does this as well). You may also utilize the metrics provided to alert when stale VMs are detected.

Where can I find the audit log(s)?

Answer: The Audit logs are found at the root of your object storage installation under `/audit-logs/audit_log/v1`. Audit Log Service (as of CircleCI v2.13) handles the storage of audit log events. Services running within a cluster may fire audit events that are then captured by this service and persisted to the

provisioned Storage mechanism for AWS S3 and On-Host.

What do the audit log files contain?

Answer: A JSON representation of event(s) for the period of time since the last file created (each file starts with a timestamp and is generally an hourly period). For example;


```
{
  "id":"27aa77e3-0255-4464-93ad-f8236533ab53",
  "version":1,
  "action":"workflow.job.finish",
  "success":true,
  "payload":{
    "job":{
      "id":"e8cef7c4-60d4-429b-8c94-09c05f309408",
      "contexts":[ ],
      "job_name":"remote_docker",
      "job_status":"success"
    },
    "workflow":{
      "id":"c022ca3c-5f6f-41ba-a6ca-05977f6a336a",
      "vcs_branch":"master"
    }
  },
  "target":{
    "id":"3c4886e1-b810-4765-a1a2-d588e6e4b9cb",
    "type":"project"
  },
  "request":{
    "id":""
  },
  "actor":{
    "id":"27075c88-9ba4-47d7-8523-fa576e839bfd",
    "type":"user"
  },
  "scope":{
    "id":"3c4886e1-b810-4765-a1a2-d588e6e4b9cb",
    "type":"project"
  }
}
```

What action types are there?

Answer:

```
context.create
context.delete
context.env_var.delete
context.env_var.store
project.add
project.follow
project.settings.update
project.stop_building
project.unfollow
user.create
user.logged_in
user.logged_out
user.suspended
workflow.error
workflow.job.context.request
workflow.job.finish
workflow.job.scheduled
workflow.job.start
workflow.retry
workflow.start
```

How can I access the files and do something with them?

Answer:

1. Set up the `awscli` and `jq` or another JSON processor for your OS.
2. In this example, grep for all `workflow.job.start` events.

```
#!/bin/bashBUCKET=YOUR-BUCKET-NAME
for key in `aws s3api list-objects --bucket BUCKET --prefix audit-logs/audit_log/v1/ --output
json | jq -r '.Contents[].Key'`;
do
echo $key;
aws s3 cp --quiet s3://BUCKET/$key - | grep  workflow.job.start;
done
```

How do I ensure proper injection of Internal CA Certificate?

Answer: If using an internal CA, or self-signed certificate, you must ensure the signing certificate is trusted by the domain service to properly connect to GitHub Enterprise.

1. The Domain Service uses a Java Truststore, loaded with Keytool. Must match the formats supported by that tool.
2. You need the full CA chain, not just `root/intermediate` certificates.
3. The CA certificate chain should be saved in `/usr/local/share/ca-certificates/`

Security and Access Control

CircleCI conducts ongoing security checks, for example, CircleCI containers are scanned by TwistLock prior to being published. CircleCI does **not** conduct ongoing security checks of your environment.

What kind of security is in place for passwords and Personally Identifiable Information (PII)? Are the passwords hashed with a strong hash function and salted?

Answer: Passwords are hashed with a 10-character salt and SHA265, refer to the Security chapter for more details.

How will the Host and Nomad clients be monitored for security issues?

Answer: Your internal security teams are responsible for monitoring the Host and Nomad clients installed in your private datacenter or cloud. CircleCI containers are scanned by TwistLock prior to being published.

System Configuration

How is configuration managed for the system?

Answer: Replicated Management Console handles all of the post-installation configuration. Installation-specific configuration is managed by Terraform or Shell scripts.

How are configuration secrets managed?

Answer: Configuration secrets are stored in plain-text on the host.

Metrics

What significant metrics will be generated?

Answer: Refer to the Monitoring section for details about monitoring and metrics.

How do I find out how many builds per day are running?

Answer:

```
use <database>
var coll = db.builds
var items = coll.find({
    "start_time": {
        $gte: ISODate("2018-03-15T00:00:00.000Z"),
        $lt: ISODate("2018-03-16T00:00:00.000Z")
    }
})
items.count()
```

Usage Statistics

How do I find the usage statistics?

Answer:

```
docker exec server-usage-stats /src/builds/extract
```

Health Checks

How is the health of dependencies (components and systems) assessed? How does the system report its own health?

Answer: Ready Agent can be used to determine the health of the system. Replicated looks to the server-ready-agent API for a 200 response. `server-ready-agent` waits to receive a 200 from all listed services, reporting a 5XX until all services come online and then it reports a 200. You can tail the logs to determine current and final state as follows:

```
docker logs -f ready-agent
```

Health of Service

Each documented service provides `/health-check`, `/healthcheck`, `/status` HTTP endpoint: 200 indicates basic health, 500 indicates bad configuration. To determine the health of individual services you must ssh into your Services VM (where all the containers are running) and make the request. The current list of services that expose a check are listed below:

- Frontend localhost:80/health-check
- API Service localhost:8082/status
- Workflows Conductor localhost:9999/healthcheck
- Federations Service localhost:8090/status

- Permissions Service localhost:3013/status
- Context Service localhost:3011/status
- Domain Service localhost:3014/status
- Cron Service localhost:4261/status
- VM Service* localhost:3001/status

* if enabled

As an example, following is how you would determine if the frontend is healthy:

```
curl -s -o /dev/null -I -w "%{http_code}\n"  0.0.0.0:80/health-check
```

Health of Dependencies

Use `/health` HTTP endpoint for internal components that expose it. Other systems and external endpoints: typically use HTTP 200 except some synthetic checks for some services.

Operational Tasks

How is the software deployed? How does rollback happen?

Answer: CircleCI uses Enterprise-Setup Terraform or Static bash scripts for deployments, Replicated is installed and orchestrates pulling all containers into your VPC. Rollbacks can only occur by reloading a previous backup and are not possible through Replicated.

What kind of scaling events take place?

Answer: Vertically scaling Service and Nomad clients is possible with downtime, Horizontally scaling Nomad Clients is possible without downtime. Refer to the Monitoring section of the Configuration chapter for details.

What kind of checks need to happen on a regular basis?

Answer: All `/health` endpoints should be checked every 60 seconds including the Replicated endpoint.

Troubleshooting

How should troubleshooting happen? What tools are available?

Answer:

It is worth noting two things. First is that the REPL is a extremely powerful tool that can cause irreparable damage to your system when used improperly. We cannot guarantee that any of the `repl` commands outside of this guide are safe to run, and do not support custom `repl` being run in our shell. The second thing is that in order to run any of our commands you'll need to run the following commands below:

1. ssh into services box

2. run `circleci dev-console`

If the above does not bring you into a REPL that mentions it is the CircleCI Dev-Console you can run the alternative command.

1. ssh into the services box
2. Run `sudo docker exec -it frontend bash`
3. Run `lein repl :connect 6005`

Once you are in the repl, you can copy and paste any of the commands below, and making the necessary substitutions in order to make the command work.

How do I view all users?

Answer:

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

How do I delete a user?

Answer:

```
(circle.http.api.admin-commands.user/delete-by-login-vcs-type! "Sirparthington" :github)
```

How do I make a user an admin?

Answer:

```
(circle.model.user/set-fields! (circle.model.user/find-one-by-github-login "your-github-username-
here") {:admin "all"})
```

How do I get user statistics?

Answer: If a if you need some basic statistics (name, email, sign in history) for your users, run the following REPL commands:

- **All Time**

```
circleci dev-console
(circle.model.user/where {} :only [:name :login :emails :admin :dev_admin :activated :sign_in_count
:current_sign_in_at :current_sign_in_ip :last_sign_in_at :last_sign_in_ip])
```

- **Last Month**

```
(circle.model.user/where
  {:last_sign_in_at {:$gt (clj-time.core/minus (clj-time.core/now) (clj-time.core/months 1))}}
  :only
  [:name :login :emails :admin :dev_admin :activated :sign_in_count :current_sign_in_at
:current_sign_in_ip :last_sign_in_at :last_sign_in_ip])
```

How do I create a new admin?

Answer: By default, the first user to access the CircleCI Server installation after it is started becomes the admin.

Options for designating additional admin users are found under the Users page in the Admin section at `https://[domain-to-your-installation]/admin/users`.

In the event the admin is unknown, or has left the company without creating a new admin, you can promote a user in the following way:

1. SSH into the services box
2. Open the CircleCI dev console with the command `circleci dev-console`
3. Run this command (replacing \<username\> with the GitHub username of the person you want to promote:

```
(-> (circle.model.user/find-one-by-login "<username>") (circle.model.user/set-fields!  {:admin
"write-settings"}))
```

How do I reset the Management Console password?

Answer: https://www.replicated.com/docs/kb/supporting-your-customers/resetting-console-password/

1. SSH into the services box
2. Use the following command: `replicated auth reset` to remove the password
3. Visit `<server>:8800/create-password` to create a new password or connect LDAP.

How do I resolve the case of VM spin-up / spin-down issues?

Answer: Make sure no builds are running that require the remote Docker environment or the machine executor, and make sure to terminate any running preallocated/remote VM EC2 instances first. Then, complete the following:

1. SSH into the services box
2. Log into the VM service database in the Postgres container: `sudo docker exec -it postgres psql -U circle vms`
3. Delete these records: `delete from vms.tasks; delete from vms.volumes; delete from vms.vms;`
4. Configure the settings in the management console to on-demand instancing (for example, set to 0 to prevent preallocated instances from being used)

5. Terminate all existing vm ec2 instances that are currently running.
6. Run `circleci dev-console` to REPL in. You should now be able to run the below commands to check queues.
7. After checking queues with the commands below, change the setting back to their original values.

Queues

Queues may become an issues for you if you are running version 2.10 or earlier. As 1.0 builds pile up and block any builds from running, run the commands below to get a feeling for how long the queues are. Then, you can promote builds from the usage-queue to the run-queue or just cancel them from the run queue.

Checking Usage Queue

```
(in-ns 'circle.backend.build.usage-queue)
(->> (all-builds) count) # Will give you the count for how many builds are in the queue

(->> (all-builds) (take 3) (map deref) (map circle.http.paths/build-url)) # If you want to check the
top three builds at the top of the queue.

(->> (all-builds) reverse (take 3) (map circle.http.paths/build-url)) # If you want to check the
builds at the end of the queue.

# If you want to promote builds from the usage queue to the run queue you can do the following:

(let [builds (->> (all-builds)
                  (take 3)
                  (map circle.http.paths/build-url)
                  (map circle.model.build/find-one-by-circle-url))]
  (doseq [b builds]
    (circle.backend.build.usage-queue/forward-build b)))

Its safe to do this by the 100's, but do not put the entire queue in.
```

Checking Run Queue

```
(circle.backend.build.run-queue/queue-depths) # returns how many are in the queue
(->> (circle.backend.build.run-queue/all-builds) (take 3) (map circle.http.paths/build-url)) # Check
the top three builds in the run-queue

# In case builds are jammed run the following. You can cancel in batches of 100.
(->> (circle.backend.build.run-queue/all-builds) (take 100) (map
circle.backend.build.cancel/cancel!))
```

Remember to set values back to original in your settings after checking queues.

Daylight-saving time changes

Is the software affected by daylight-saving time changes (both client and server)?

Answer: No. All date/time data converted to UTC with offset before processing.

Data cleardown

Which data needs to be cleared down? How often? Which tools or scripts control cleardown?

Answer: If using On-Host storage and Static, all storage should be mounted.

Log rotation

Is log rotation needed? How is it controlled?

Answer: Docker automatically rotates logs.

Replicated Failover and Recovery procedures

What needs to happen when parts of the system are failed over to standby systems? What needs to happen during recovery?

Answer: Refer to the Backup and Troubleshooting sections of this document for details.

User Management

How do I provision admin users?

Answer: The first user who logs in to the CircleCI application will automatically be designated an admin user. Options for designating additional admin users are found under the Users page in the Admin section at `https://[domain-to-your-installation]/admin/users`.

Backup and Recovery

This chapter describes failover or replacement of the services machine. Refer to the Backup section below for information about possible backup strategies and procedures for implementing a regular backup image or snapshot of the services machine.

Disaster Recovery

Specify a spare machine, in an alternate location, with the same specs for disaster recovery of the services machine. Having a hot spare regularly imaged with the backup snapshot in a failure scenario is best practice.

At the very least, provide systems administrators of the CircleCI installation with the hostname and location (even if co-located) of an equivalent server on which to install a replacement server with the latest snapshot of the services machine configuration. To complete recovery, use the Installation procedure, replacing the image from that procedure with your backup image.

Backing up CircleCI Data

This document describes how to back up your CircleCI application so that you can recover from accidental or unexpected loss of CircleCI data attached to the Services machine:

If you are running CircleCI in an HA configuration, you must use standard backup mechanisms for the external datastores. Contact support@circleci.com for more information document for more information.

Backing up the Database

If you have **not** configured CircleCI for external services, the best practice for backing up your CircleCI data is to use VM snapshots of the virtual disk acting as the root volume for the Services machine. Backups may be performed without downtime as long the underlying virtual disk supports such an operation as is true with AWS EBS. There is a small risk, that varies by filesystem and distribution, that snapshots taken without a reboot may have some data corruption, but this is rare in practice.

"Snapshots Disabled" refers to Replicated's built-in snapshot feature that is turned off by default.

Backing up Object Storage

Build artifacts, output, and caches are generally stored in object storage services like AWS S3. These services are considered highly redundant and are unlikely to require separate backup. An exception is if your instance is setup to store large objects locally on the Services machine, either directly on-disk or on an NFS volume. In this case, you must separately back these files up and ensure they are mounted back to the same location on restore.

Snapshotting on AWS EBS

There are a few features of AWS EBS snapshots that make the backup process quite easy:

1. To take a manual backup, choose the instance in the EC2 console and select Actions > Image > Create Image.
2. Select the No reboot option if you want to avoid downtime. An AMI that can be readily launched as a new EC2 instance for restore purposes is created.

It is also possible to automate this process with the AWS API. Subsequent AMIs/snapshots are only as large as the difference (changed blocks) since the last snapshot, such that storage costs are not necessarily larger for more frequent snapshots, see Amazon's EBS snapshot billing document for details.

Restoring From Backup

When restoring test backups or performing a restore in production, you may need to make a couple of changes on the newly launched instance if its public or private IP addresses have changed:

1. Launch a fresh EC2 instance using the newly generated AMI from the previous steps
2. Stop the app in the Management Console (at port 8800) if it is already running
3. Ensure that the hostname configured in the Management Console at port 8800 reflects the correct address. If this hostname has changed, you will also need to change it in the corresponding GitHub OAuth application settings or create a new OAuth app to test the recovery and log in to the application.
4. Update any references to the backed-up instance's public and private IP addresses in `/etc/default/replicated` and `/etc/default/replicated-operator` on Debian/Ubuntu or `/etc/sysconfig/*` in RHEL/CentOS to the new IP addresses.
5. From the root directory of the Services box, run `sudo rm -rf /opt/nomad`. State is saved in the `/opt/nomad` folder that can interfere with builds running when an installation is restored from a backup. The folder and its contents will be regenerated by Nomad when it starts.
6. Restart the app in the Management Console at port 8800.

Cleaning up Build Records

While filesystem-level data integrity issues are rare and preventable, there will likely be some data anomalies in a point-in-time backup taken while builds are running on the system. For example, a build that is only half-way finished at backup time may result in missing the latter half of its command output, and it may permanently show that it is in Running state in the application.

If you want to clean up any abnormal build records in your database after a recovery, you can delete them by running the following commands on the Services machine replacing the example build URL with an actual URL from your CircleCI application:

```
circleci dev-console
# Wait for console to load
user=> (admin/delete-build "https://my-circleci-hostname.com/gh/my-org/my-project/1234")
```

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 request ID assigned to this request by CircleCI), `ip_address` (the original IP address in IPV4 dotted notation from which the request was made, eg. 127.0.0.1), and `client_trace_id` (the client trace ID header, if present, from the 'X-Client-Trace-Id' HTTP header of the original request).

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.

Troubleshooting Server Installations

This document describes an initial set of troubleshooting steps to take if you are having problems with your CircleCI installation on your private server. If your issue is not addressed below, you can generate a support bundle and contact CircleCI Support Engineers by opening a support ticket.

Generating a Support Bundle

To download a support bundle, select **Support** from the Management Console menu bar, and then select **Download Support Bundle**. CircleCI support engineers will often request support bundles to help diagnose/fix the problem you are experiencing.

Debug Queuing Builds

If your Services component is fine, but builds are not running, or all builds are queueing, follow the steps below.

1. Check Dispatcher Logs for Errors

Run `sudo docker logs dispatcher`, if you see log output that is free of errors you may continue on the next step.

If the logs dispatcher container does not exist or is down, start it by running the `sudo docker start <container_name>` command and monitor the progress. The following output indicates that the logs dispatcher is up and running correctly:

```
Jan 4 22:38:38.589:+0000 INFO circle.backend.build.run-queue dispatcher mode is on - no need for

 run-queue
Jan 4 22:38:38.589:+0000 INFO circle.backend.build.usage-queue 5a4ea0047d560d00011682dc:

 GERey/realitycheck/37 -> forwarded to run-queue
Jan 4 22:38:38.589:+0000 INFO circle.backend.build.usage-queue 5a4ea0047d560d00011682dc: publishing

 :usage-changed (:recur) event

Jan 4 22:38:39.069:+0000 INFO circle.backend.build.usage-queue got usage-queue event for

 5a4ea0047d560d00011682dc (finished-build)
```

If you see errors or do not see the above output, investigate the stack traces because they indicate that there is an issue with routing builds from 1.0 to 2.0. If there are errors in the output, then you may have a problem with routing builds to 1.0 or 2.0 builds.

If you can run 1.0 builds, but not 2.0 builds, or if you can only run 2.0 builds and the log dispatcher is up and running, continue on to the next steps.

2. Check Picard-Dispatcher Logs for Errors

Run the `sudo docker logs picard-dispatcher` command. A healthy `picard-dispatcher` should output the following:

```
Jan 9 19:32:33 INFO picard-dispatcher.init Still running...
Jan 9 19:34:33 INFO picard-dispatcher.init Still running...
Jan 9 19:34:44 INFO picard-dispatcher.core taking build=GERey/realitycheck/38
Jan 9 19:34:45 INFO circle.http.builds project GERey/realitycheck at revision

2c6179654541ee3d succcessfully fetched and parsed .circleci/config.yml

picard-dispatcher.tasks build GERey/realitycheck/38 is using resource

class {:cpu 2.0, :ram 4096, :class :medium}
picard-dispatcher.tasks Computed tasks for build=GERey/realitycheck/38,

 stage=:write_artifacts, parallel=1
Jan 9 19:34:45 INFO picard-dispatcher.tasks build has matching jobs:

 build=GERey/realitycheck/38 parsed=:write_artifacts passed=:write_artifacts
```

The output should be filled with the above messages. If it is a slow day and builds are not happening very often, the output will appear as follows:

```
Jan 9 19:32:33.629:+0000 INFO picard-dispatcher.init Still running...
```

As soon as you run a build, you should see the above message to indicate that it has been dispatched to the scheduler. If you do not see the above output or you have a stack trace in the picard-dispatcher container, contact support@circleci.com.

If you run a 2.0 build and do not see a message in the picard-dispatcher log output, it often indicates that a job is getting lost between the dispatcher and the picard dispatcher.

Stop and restart the CircleCI app in the Management Console at port 8800 to re-establish the connection between the two containers.

3. Check Picard-Scheduler Logs for Errors

Run `sudo docker logs picard-scheduler` . The `picard-scheduler` schedules jobs and sends them to nomad through a direct connection. It does not actually handle queuing of the jobs in CircleCI.

4. Check Nomad Node Status

Check to see if there are any nomad nodes by running the `nomad node-status -allocs` command and viewing the following output:

```
ID        DC         Name              Class        Drain  Status  Running Allocs
ec2727c5  us-east-1  ip-127-0-0-1      linux-64bit  false  ready   0
```

If you do not see any nomad clients listed, please consult our <nomad#,Introduction to Nomad Cluster Operation> for more detailed information on managing and troubleshooting the nomad server.

DC in the output stands for datacenter and will always print us-east-1 and should be left as such. It doesn't affect or break anything. The things that are the most important are the Drain, Status, and Allocs columns.

- **Drain** - If `Drain` is `true` then CircleCI will **not** route jobs to that nomad client. It is possible to change this value by running the following command `nomad node-drain [options] <node>`. If you set Drain to `true`, it will finish the jobs that were currently running and then stop accepting builds. After the number of allocations reaches 0, it is safe to terminate instance. If `Drain` is set to `false` it means the node is accepting connections and should be getting builds.
- **Status** - If Status is `ready` then it is ready to accept builds and should be wired up correctly. If it is not wired up correctly it will not show `ready` and it should be investigated because a node that is not showing `ready` in the Status will not accept builds.
- **Allocs** - Allocs is a term used to refer to builds. So, the number of Running Allocs is the number of builds running on a single node. This number indicates whether builds are routing. If all of the Builders have Running Allocs, but your job is still queued, that means you do not have enough capacity and you need to add more Builders to your fleet.

If you see output like the above, but your builds are still queued, then continue to the next step.

5. Check Job Processing Status

Run the `sudo docker exec -it nomad nomad status` command to view the jobs that are currently being processed. It should list the status of each job as well as the ID of the job, as follows:

```
ID                                                  Type   Priority  Status
5a4ea06b7d560d000116830f-0-build-GERey-realitycheck-1   batch  50        dead
5a4ea0c9fa4f8c0001b6401b-0-build-GERey-realitycheck-2   batch  50        dead
5a4ea0cafa4f8c0001b6401c-0-build-GERey-realitycheck-3   batch  50        dead
```

After a job has completed, the Status shows `dead`. This is a regular state for jobs. If the status shows `running`, the job is currently running. This should appear in the CircleCI app builds dashboard. If it is not appearing in the app, there may be a problem with the output-processor. Run the `docker logs picard-output-processor` command and check the logs for any obvious stack traces.

- If the job is in a constant `pending` state with no allocations being made, run the `sudo docker exec -it nomad nomad status JOB_ID` command to see where Nomad is stuck and then refer to standard Nomad Cluster error documentation for information.
- If the job is running/dead but the CircleCI app shows nothing:
  - Check the Nomad job logs by running the `sudo docker exec -it nomad nomad logs --stderr --job`

`JOB_ID` command.

- Run the `picard-output-processor` command to check those logs for specific errors.

The use of `--stderr` is to print the specific error if one exists.

Why do my Jobs stay in `queued` status until they fail and never successfully run?

If the nomad client logs contain the following error message typw, check port 8585:

```
{"error":"rpc error: code = Unavailable desc = grpc: the connection is
unavailable","level":"warning","msg":"error fetching config, retrying","time":"2018-04-
17T18:47:01Z"}
```

Why is the cache failing to unpack?

If a `restore_cache` step is failing for one of your jobs, it is worth checking the size of the cache - you can view the cache size from the CircleCI Jobs page within the `restore_cache` step. We recommend keeping cache sizes under 500MB – this is our upper limit for corruption checks because above this limit check times would be excessively long. Larger cache sizes are allowed but may cause problems due to a higher chance of decompression issues and corruption during download. To keep cache sizes down, consider splitting into multiple distinct caches.

How do I get around the API service being impacted by a high thread count?

Disable cache warming by completing the following steps:

1. Add the export `DOMAIN_SERVICE_REFRESH_USERS=false` flag to the `` `/etc/circleconfig/api-service/customizations` `` file on the Services machine. For more information on configuration overrides, see the guide to Service Configuration Overrides.
2. Restart CircleCI:
    1. Navigate to the Management Console
    2. Click Stop Now and wait for it to stop
    3. Click Start

Frequently Asked Questions

This document is intended for system administrators of self-hosted installations of CircleCI Server.

This chapter answers frequently asked questions and provides installation troubleshooting tips.

Can I move or change my GitHub Enterprise URL without downtime?

No, because of the nature of CircleCI integration with GitHub authentication, you should not change the domain of your GHE instance after CircleCI is in production. Redeploying GitHub without will result in a corrupted CircleCI instance. Contact support if you plan to move your GitHub instance.

Can I monitor available build containers?

Yes, refer to the Introduction to Nomad Cluster Operation section for details. Refer to the Monitoring Your Installation section for how to enable additional container monitoring for AWS.

How do I provision admin users?

The first user who logs in to the CircleCI application will automatically be designated an admin user. Options for designating additional admin users are found under the Users page in the Admin section at `https://[domain-to-your-installation]/admin/users`.

How can I gracefully shutdown Nomad Clients?

Refer to the Introduction to Nomad Cluster Operation chapter for details.

Why is Test GitHub Authentication failing?

This means that the GitHub Enterprise server is not returning the intermediate SSL certificates. Check your GitHub Enterprise instance with https://www.ssllabs.com/ssltest/analyze.html – it may report some missing intermediate certs. You can use commands like `openssl` to get the full certificate chain for your server.

In some cases authentication fails when returning to the configuration page after it was successfully set up once. This is because the secret is encrypted, so when returning checking it will fail.

How can I use HTTPS to access CircleCI?

While CircleCI creates a self-signed cert when starting up, that certificate only applies to the management console and not the CircleCI product itself. If you want to use HTTPS, you'll have to provide certificates to use under the `Privacy` section of the settings in the management console.

Why doesn't terraform destroy every resource?

CircleCI sets the services box to have termination protection in AWS and also writes to an s3 bucket. If you want terraform to destroy every resource, you'll have to either manually delete the instance, or turn off termination protection in the `circleci.tf` file. You'll also need to empty the s3 bucket that was created as part of the terraform install.

Do the Nomad Clients store any state?

They can be torn down without worry as they don't persist any data.

How do I verify TLS settings are failing?

Make sure that your keys are in unencrypted PEM format, and that the certificate includes the entire chain of trust as follows:

```
-----BEGIN CERTIFICATE-----
your_domain_name.crt
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
intermediate 1
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
intermediate 2
-----END CERTIFICATE-----
...
```

How do I debug the Management Console (Replicated)?

The CircleCI management console is powered by Replicated. If you are experiencing any issues with the Management Console, here are a few ways to debug it:

1. Check you have Replicated installed

First, make sure you have the CLI tool for Replicated installed by running the following:

```
replicated -version
```

2. Restart Replicated and the CircleCI app

Try restarting Replicated services. You can do this by running the following commands on the service box, for Ubuntu 14.04:

```
sudo service replicated-ui restart
sudo service replicated restart
sudo service replicated-operator restart
```

For Ubuntu 16.04, run the following commands:

```
sudo systemctl restart replicated-ui
sudo systemctl restart replicated
sudo systemctl restart replicated-operator
```

Then try restarting the CircleCi app: go to your services box admin (for example, `<your-circleci-hostname>.com:8800`) and try restarting with "Stop Now" and "Start Now".

3. Try to log into Replicated

Try logging in to Replicated. You can do this by running the following command on the service box. You will be asked to enter your password - the same one used to unlock the Management Console (i.e. `<your-circleci-hostname>.com:8800`).

```
replicated login
```

If you could login, then run the following command and send the output to us at support@circleci.com so we can help diagnose what is causing the problem you are experiencing.

```
sudo replicated apps
```

If you were seeing the following error: `request returned Unauthorized for API route` this could be because you are not logged into Replicated, so please check if you are still getting the error after a successful login.

4. Check Replicated logs

You can find Replicated logs on the Services machine under `/var/log/replicated`.

5. Check what Docker containers are currently running

Replicated starts many Docker containers to run CircleCI Server, so it can be useful to check what containers are running.

To check what containers are currently running, run `sudo docker ps` and you should see something similar to this output:

```
$ sudo docker ps
CONTAINER ID        IMAGE                                                                     COMMAND
CREATED             STATUS              PORTS
NAMES
eb2970306859        172.31.72.162:9874/circleci-api-service:0.1.6910-8b54ef9              "circleci-
service-run"   26 hours
ago         Up 26 hours         0.0.0.0:32872->80/tcp, 0.0.0.0:32871->443/tcp, 0.0.0.0:8082-
>3000/tcp,
0.0.0.0:32870->6010/tcp, 0.0.0.0:32869->8585/tcp                                  api-service

01d26714f5f5        172.31.72.162:9874/circleci-workflows-conductor:0.1.38931-1a904bc8
```

```
"/service/docker-ent…"   26 hours
ago        Up 26 hours         0.0.0.0:9998->9998/tcp, 0.0.0.0:32868->80/tcp, 0.0.0.0:32867-
>443/tcp,
0.0.0.0:9999->3000/tcp, 0.0.0.0:32866->8585/tcp                                  workflows-
conductor

0cc6e4248cfb        172.31.72.162:9874/circleci-permissions-service:0.1.1195-b617002
"/service/docker-ent…"   26 hours
ago        Up 26 hours         0.0.0.0:3013->3000/tcp
permissions-service

9e6efc98b7d6        172.31.72.162:9874/circleci-cron-service:0.1.680-1fcd8d2             "circleci-
service-run"   26 hours
ago        Up 26 hours         0.0.0.0:4261->4261/tcp
cron-service
8c40bd1cecf6        172.31.72.162:9874/circleci-federations-service:0.1.1134-72edcbc
"/service/docker-ent…"   26 hours
ago        Up 26 hours         0.0.0.0:3145->3145/tcp, 0.0.0.0:8010->8010/tcp, 0.0.0.0:8090-
>8090/tcp
federations-service
71c71941684f        172.31.72.162:9874/circleci-contexts-service:0.1.6073-5275cd5        "./docker-
entrypoint…"   26 hours
ago        Up 26 hours         0.0.0.0:2718->2718/tcp, 0.0.0.0:3011->3011/tcp, 0.0.0.0:8091-
>8091/tcp
contexts-service
71ffeb230a90        172.31.72.162:9874/circleci-domain-service:0.1.4040-eb63b67
"/service/docker-ent…"   26 hours
ago        Up 26 hours         0.0.0.0:3014->3000/tcp
domain-service
eb22d3c10dd8        172.31.72.162:9874/circleci-audit-log-service:0.1.587-fa47042        "circleci-
service-run"   26 hours
ago        Up 26 hours
audit-log-service
243d9082e35c        172.31.72.162:9874/circleci-frontend:0.1.203321-501fada              "/docker-
entrypoint.…"   26 hours
ago        Up 26 hours         0.0.0.0:80->80/tcp, 0.0.0.0:443->443/tcp, 0.0.0.0:4434->4434/tcp
frontend
af34ca3346a7        172.31.72.162:9874/circleci-picard-dispatcher:0.1.10401-aa50e85      "circleci-
service-run"   26 hours
ago        Up 26 hours
picard-dispatcher
fb0ee1b02d48        172.31.72.162:9874/circleci-vm-service:0.1.1370-ad05648              "vm-
service-service-…"   26 hours ago        Up 26 hours         0.0.0.0:3001->3000/tcp
vm-service
3708dc80c63e        172.31.72.162:9874/circleci-vm-scaler:0.1.1370-ad05648               "/scaler-
entrypoint.…"   26 hours
ago        Up 26 hours         0.0.0.0:32865->5432/tcp
vm-scaler
77bc9d0b4ac9        172.31.72.162:9874/circleci-vm-gc:0.1.1370-ad05648                   "docker-
entrypoint.s…"   26 hours
ago        Up 26 hours         0.0.0.0:32864->5432/tcp
vm-gc
4b02f202a05d        172.31.72.162:9874/circleci-output-processing:0.1.10386-741e1d1      "output-
processor-se…"   26 hours
```

```
ago        Up 26 hours         0.0.0.0:8585->8585/tcp, 0.0.0.0:32863->80/tcp, 0.0.0.0:32862->443/tcp
picard-output-processor
b8f982d32989        172.31.72.162:9874/circleci-frontend:0.1.203321-501fada              "/docker-
entrypoint.…"   26 hours ago        Up 26 hours         0.0.0.0:32861->80/tcp, 0.0.0.0:32860-
>443/tcp, 0.0.0.0:32859->4434/tcp
dispatcher
601c363a0c38        172.31.72.162:9874/circleci-frontend:0.1.203321-501fada              "/docker-
entrypoint.…"   26 hours
ago        Up 26 hours         0.0.0.0:32858->80/tcp, 0.0.0.0:32857->443/tcp, 0.0.0.0:32856-
>4434/tcp                                                                                 legacy-
notifier
f2190c5f3aa9        172.31.72.162:9874/mongo:3.6.6-jessie
"/entrypoint.sh"         26 hours
ago        Up 26 hours         0.0.0.0:27017->27017/tcp
mongo
3cbbd959f42e        172.31.72.162:9874/telegraf:1.6.4
"/telegraf-entrypoin…"   26 hours
ago        Up 26 hours         0.0.0.0:8125->8125/udp, 0.0.0.0:32771->8092/udp, 0.0.0.0:32855-
>8094/tcp                                                                                 telegraf
15b090e8cc02        172.31.72.162:9874/circleci-schedulerer:0.1.10388-741e1d1            "circleci-
service-run"   26 hours
ago        Up 26 hours
picard-scheduler
fb967bd3bca0        172.31.72.162:9874/circleci-server-nomad:0.5.6-5.1                   "/nomad-
entrypoint.sh"   26 hours
ago        Up 26 hours         0.0.0.0:4646-4648->4646-4648/tcp
nomad
7e0743ee2bfc        172.31.72.162:9874/circleci-test-results:0.1.1136-b4d94f6            "circleci-
service-run"   26 hours
ago        Up 26 hours         0.0.0.0:2719->2719/tcp, 0.0.0.0:3012->3012/tcp
test-results
0a95802c87dc        172.31.72.162:9874/circleci-slanger:0.4.117-42f7e6c                  "/docker-
entrypoint.…"   26 hours
ago        Up 26 hours         0.0.0.0:4567->4567/tcp, 0.0.0.0:8081->8080/tcp
slanger
ca445870a057        172.31.72.162:9874/circleci-postgres-script-enhance:0.1.9-38edabf    "docker-
entrypoint.s…"   26 hours
ago        Up 26 hours         0.0.0.0:5432->5432/tcp
postgres
a563a228a93a        172.31.72.162:9874/circleci-server-ready-agent:0.1.105-0193c73       "/server-
ready-agent"    26 hours
ago        Up 26 hours         0.0.0.0:8099->8000/tcp
ready-agent
d6f9aaae5cf2        172.31.72.162:9874/circleci-server-usage-stats:0.1.122-70f28aa       "bash -c
/src/entryp…"   26 hours
ago        Up 26 hours
usage-stats
086a53d9a1a5        registry.replicated.com/library/statsd-graphite:0.3.7
"/usr/bin/supervisor…"   26 hours
ago        Up 26 hours         0.0.0.0:32851->2443/tcp, 0.0.0.0:32770->8125/udp
replicated-statsd
cc5e062844be        172.31.72.162:9874/circleci-shutdown-hook-poller:0.1.32-9c553b4
"/usr/local/bin/pyth…"   26 hours
ago        Up 26 hours
```

```
musing_volhard
9609f04c2203        172.31.72.162:9874/circleci-rabbitmq-delayed:3.6.6-management-12      "docker-
entrypoint.s…"   26 hours
ago        Up 26 hours         0.0.0.0:5672->5672/tcp, 0.0.0.0:15672->15672/tcp, 0.0.0.0:32850-
>4369/tcp, 0.0.0.0:32849->5671/tcp, 0.0.0.0:32848->15671/tcp, 0.0.0.0:32847->25672/tcp   rabbitmq
2bc0cfe43639        172.31.72.162:9874/tutum-logrotate:latest                             "crond -f"
26 hours
ago        Up 26 hours
hardcore_cray
79aa857e23b4        172.31.72.162:9874/circleci-vault-cci:0.3.8-e2823f6                   "./docker-
entrypoint…"   26 hours
ago        Up 26 hours         0.0.0.0:8200-8201->8200-8201/tcp
vault-cci
b3e317c9d62f        172.31.72.162:9874/redis:4.0.10                                       "docker-
entrypoint.s…"   26 hours
ago        Up 26 hours         0.0.0.0:6379->6379/tcp
redis
f2d3f77891f0        172.31.72.162:9874/circleci-nomad-metrics:0.1.90-1448fa7
"/usr/local/bin/dock…"   26 hours
ago        Up 26 hours
nomad-metrics
1947a7038f24        172.31.72.162:9874/redis:4.0.10                                       "docker-
entrypoint.s…"   26 hours
ago        Up 26 hours         0.0.0.0:32846->6379/tcp
slanger-redis
3899237a5782        172.31.72.162:9874/circleci-exim:0.2.54-697cd08                       "/docker-
entrypoint.…"   26 hours
ago        Up 26 hours         0.0.0.0:2525->25/tcp
exim
97ebdb831a7e        registry.replicated.com/library/retraced:1.2.2
"/src/replicated-aud…"   26 hours
ago        Up 26 hours         3000/tcp
retraced-processor
a0b806f3fad2        registry.replicated.com/library/retraced:1.2.2
"/src/replicated-aud…"   26 hours
ago        Up 26 hours         172.17.0.1:32771->3000/tcp
retraced-api
19dec5045f6e        registry.replicated.com/library/retraced:1.2.2                        "/bin/sh
-c '/usr/lo…"   26 hours
ago        Up 26 hours         3000/tcp
retraced-cron
7b83a3a193da        registry.replicated.com/library/retraced-postgres:10.5-20181009       "docker-
entrypoint.s…"   26 hours
ago        Up 26 hours         5432/tcp
retraced-postgres
029e8f454890        registry.replicated.com/library/retraced-nsq:v1.0.0-compat-20180619   "/bin/sh
-c nsqd"        26 hours
ago        Up 26 hours         4150-4151/tcp, 4160-4161/tcp, 4170-4171/tcp
retraced-nsqd
500619f53e80        quay.io/replicated/replicated-operator:current
"/usr/bin/replicated…"   26 hours
ago        Up 26 hours
replicated-operator
e1c752b4bd6c        quay.io/replicated/replicated:current
```

```
"entrypoint.sh -d"       26 hours
ago        Up 26 hours         0.0.0.0:9874-9879->9874-9879/tcp
replicated
1668846c1c7a        quay.io/replicated/replicated-ui:current
"/usr/bin/replicated…"   26 hours
ago        Up 26 hours         0.0.0.0:8800->8800/tcp
replicated-ui
f958cf3e8762        registry.replicated.com/library/premkit:1.2.0
"/usr/bin/premkit da…"   3 weeks
ago         Up 26 hours         80/tcp, 443/tcp, 2080/tcp, 0.0.0.0:9880->2443/tcp
replicated-premkit
```

Providing support@circleci.com with the output of `sudo docker ps` from the Services machine will help us diagnose the cause of your problem.

Customization and Configuration

The following sections summarize the key files and variables that impact CircleCI Server behavior, and configuration options for your Server installation.

Notable Files & Folders

Need	Path	More info
General Config	`/etc/circle-installation-customizations`	See table below for values
JVM Heap Sizes	`/etc/circleconfig/XXXX/customizations` Supports: frontend, test_results	Adjust heap size for individual containers with `JVM_HEAP_SIZE`
Custom CA Certs	`/usr/local/share/ca-certificates/`	
Container Customizations	`/etc/circleconfig/XXX/customizations`	Used lots of places in replicated
`/etc/hosts`	`/etc/hosts`	Respected by several containers including frontend, copied to container's /etc/hosts
`/etc/environment`	`/etc/environment`	Respected by all containers

Properties of `/etc/circle-installation-customizations`

Every property should be in the format `export ENV_VAR="value"`

Property	Impact	More info
CIRCLE_URL	Override the scheme and host that CircleCI uses	
JVM_HEAP_SIZE	Set JVM heap size for **all** containers reading this property	Use container specific settings when possible (see files above)

Other Properties and Env Vars

Property	Impact	More info
HTTP_PROXY, NO_PROXY	Proxy for replicated and other services outside CircleCI containers to use	

Service Configuration Overrides

This section describes the configuration interface for overriding services in CircleCI Server.

Customizing your configuration can have potentially damaging consequences, so we recommend contacting support@circleci.com for guidance before making any changes.

Configuration is done by exporting environment variables in files located on the Services machine.

Consider the file “customizations” created at the following path /etc/circleconfig/workflows-conductor:

```
export FOO="bar"
```

The value of FOO will take precedence over the default values set in the default container mapping in the CircleCI Server configuration.

Available Overrides

```
/etc/circleconfig/api-service/customizations
/etc/circleconfig/audit-log-service/customizations
/etc/circleconfig/contexts-service-db-migrator/customizations
/etc/circleconfig/contexts-service/customizations
/etc/circleconfig/cron-service-db-migrator/customizations
/etc/circleconfig/cron-service/customizations
/etc/circleconfig/domain-service-migrator/customizations
/etc/circleconfig/domain-service/customizations
/etc/circleconfig/federations-service-db-migrator/customizations
/etc/circleconfig/federations-service-migrator/customizations
/etc/circleconfig/frontend/customizations
/etc/circleconfig/output-processor/customizations
/etc/circleconfig/permissions-service-migrator/customizations
/etc/circleconfig/permissions-service/customizations
/etc/circleconfig/picard-dispatcher/customizations
/etc/circleconfig/schedulerer/customizations
/etc/circleconfig/test-results/customizations
/etc/circleconfig/vm-gc/customizations
/etc/circleconfig/vm-scaler/customizations
/etc/circleconfig/vm-service-db-migrator/customizations
/etc/circleconfig/vm-service/customizations
/etc/circleconfig/workflows-conductor/customizations
```

Resource Classes

Introduced in CircleCI Server v2.19

You can customize resource classes for your installation to provide developers with CPU/RAM options for the Jobs they configure.

The resources for `machine` executors can't be configured using the method described on this page. To change CPU and memory size for VMs, change `AWS Instance Type` in the VM Provider section of the Management Console. See the VM Service guide for more details.

Once resource classes are set using the steps below, make these options available to developers so they can ensure correct usage.

Following are the steps required to customize resource classes for the Docker executors:

1. SSH into the Services machine.
2. Run the following:

```
sudo mkdir /etc/circleconfig/picard-dispatcher
```

3. Run the following:

```
sudo vim /etc/circleconfig/picard-dispatcher/resource-definitions.edn
```

4. Add your required customizations to the file, then save and exit vim with `:wq` - see below for options and formatting.
5. Run:

```
echo 'export CIRCLE_DISPATCHER_RESOURCE_DEF=/circleconfig/picard-dispatcher/resource-
definitions.edn' | sudo tee /etc/circleconfig/picard-dispatcher/customizations
```

6. Restart the CircleCI Server application. The application can be stopped and started again from the Management Console Dashboard (for example, `<circleci-hostname>.com:8800`).

Below is an example resource class configuration:

Example config:

```
{:default-resource-class :medium

 :resource-classes
 {:docker
  ;; Modify below
  {:small {:id "d1.small" :availability :general :ui {:cpu 2.0 :ram 4096 :class :small} :outer {:cpu
2.0 :ram 4096}}
   :medium {:id "d1.medium" :availability :general :ui {:cpu 4.0 :ram 8192 :class :medium} :outer
{:cpu 4.0 :ram 8192}}
   :massive {:id "d1.massive" :availability :general :ui {:cpu 7.0 :ram 28000 :class :massive}
:outer {:cpu 7.0 :ram 28000}}}
  ;; Modify above

  ;; NOTE: Do not delete or modify the following block: Such attempts will break machine builds.
  :machine
  {:medium {:id "l1.medium" :availability :general :ui {:cpu 2.0 :ram 4096 :class :medium} :outer
{:cpu 1 :ram 512}}
   :large {:id "l1.large" :availability :general :ui {:cpu 4.0 :ram 16384 :class :medium} :outer
{:cpu 1 :ram 512}}
   :windows.medium {:id "windows.medium" :availability :general :ui {:cpu 2.0 :ram 8192 :class
:windows.medium} :outer {:cpu 1 :ram 512}}}}}
```

Let's take a look at one of the options in more detail

```
:medium {:id "d1.medium" :availability :general :ui {:cpu 4.0 :ram 8192 :class :medium} :outer {:cpu
4.0 :ram 8192}
```

- `:medium` - this is the name that your developers will use to refer to the resource class in their config.yml and the is the external facing name of the resource class.
- `:id "d1.medium"` - this is the internal name for the resource class. You can customize this ID for Docker resource classes.
- `:availability :general` - required field
- `:ui {:cpu 4.0 :ram 8192 :class :medium}` - Information used by the CircleCI UI. This this should be kept in parity with :outer - see below.
- `:outer {:cpu 4.0 :ram 8192}` - This defines the CPU and RAM for the resource class.

Jobs can only run if the Nomad client has enough CPU/RAM in order to allocate the resources required. If not, the job will be queued. See our Nomad metrics guide for information on monitoring the capacity of your Nomad cluster, and Nomad Client System Requirements for more about capacity, and how Nomad allocates jobs.

Login Screen

Introduced in CircleCI Server v2.17.3

You can add a banner to your login screen as follows:

1. Access the file: `/etc/circleconfig/frontend/customizations` on the Services machine
2. Add the following line, substituting the text you wish to display in the banner:

```
export CIRCLE__OUTER__LOGIN_BANNER_MESSAGE="<insert-your-message-here>
```

3. Restart CircleCI from the Management Console (your-circleci-hostname.com:8800)

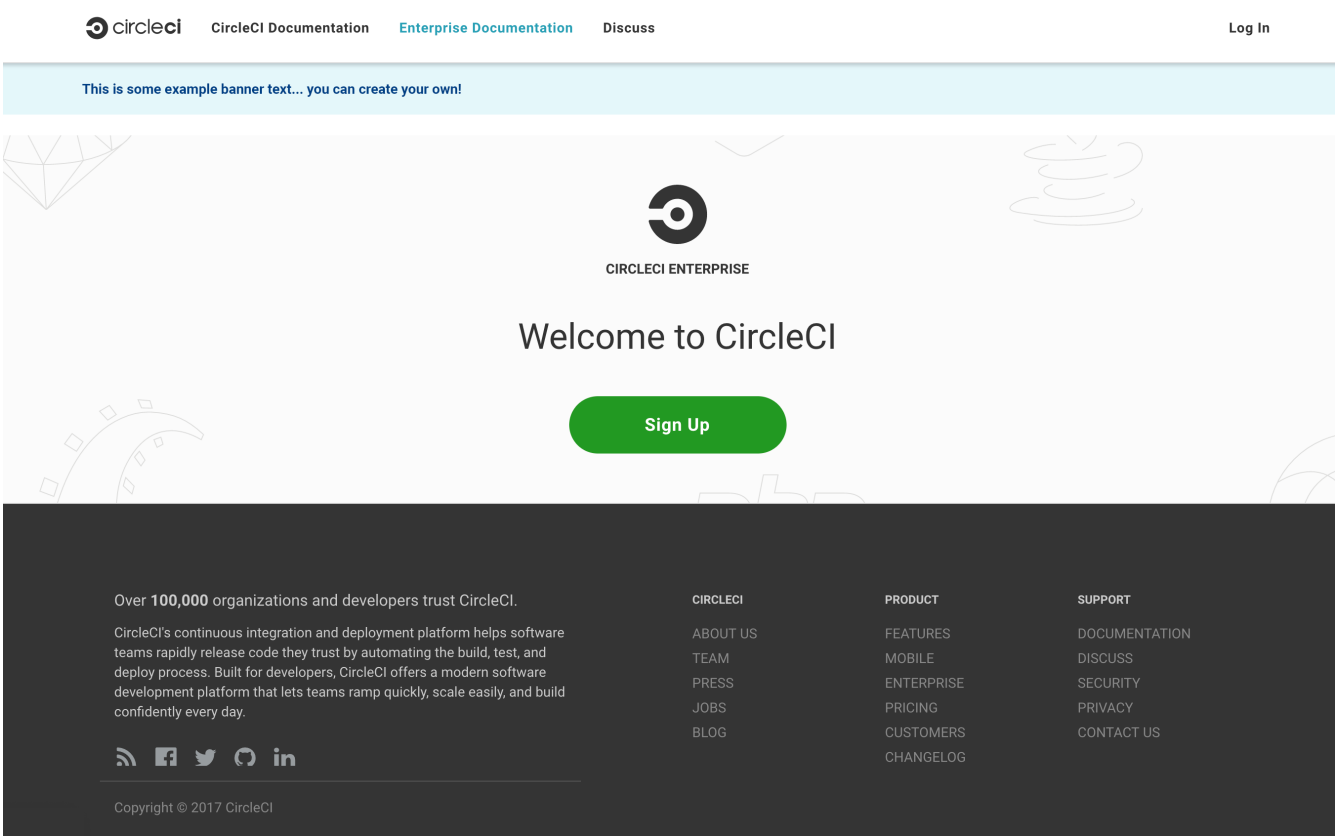


Figure 24. Login Screen Banner Example

CircleCI Server Container Architecture

This document outlines the containerized services that run on the Services machine within a CircleCI Server installation. This is provided both to give an overview of service operation, and to help with troubleshooting in the event of service outages. Supplementary notes and a key are provided below the following table.

Notes

- Database migrator services are listed here with a low failure severity as they only run at startup, however:

  If migrator services are down at startup connected services will fail.

- With a platinum support contract some services can be externalized (marked with * here) and managed to suit your requirements. Externalization provides higher data security and allows for redundancy to be built into your system.

key

Icon	Description
⊘	Failure has a minor affect on production - no loss of data or functioning.
⚠	Failure might cause issues with some jobs, but no loss of data.
☠	Failure can cause loss of data, corruption of jobs/workflows, major loss of functionality.

Containers, Roles, Failure Modes and Startup Dependencies

Container / Image	Role	What happens if it fails?	Failure severity	Startup dependencies
`api-service`	Provides a GraphQL API that provides much of the data to render the web frontend.	Many parts of the UI (e.g. Contexts) will fail completely.	🚒	`postgres`, `frontend`, `contexts-service-migrator`, `contexts-service`, `vault-cci`
`audit-log-service`	Persists audit log events to blob storage for long term storage.	Some events may not be recorded.	✅	`postgres`, `frontend`
`contexts-service`	Stores and provides encrypted contexts.	All builds using Contexts will fail.	⚠️	`postgres`, `frontend`, `contexts-service-migrator`, `vault-cci`
`contexts-service-migrator`	Runs postgresql migrations for the `contexts-service`.	Only runs at startup.	✅	`postgres`, `frontend`
`cron-service`	Triggers scheduled workflows.	Scheduled workflows will not run.	⚠️	`postgres`, `frontend`, `cron-service-migrator`
`cron-service-migrator`	Runs postgresql migrations for the cron-service.	Only runs at startup.	✅	`postgres`, `frontend`
`domain-service`	Stores and provides information about our domain model.	Workflows will fail to start and some REST API calls may fail causing `500` errors in the CircleCI UI. If LDAP authentication is in use, all logins will fail.	🚒	`postgres`, `frontend`, `domain-service-migrator`
`domain-service-migrator`	Runs postgresql migrations for the `domain-service`.	Only runs at startup.	✅	`postgres`, `frontend`
`exim`	Mail Transfer Agent (MTA) used to send all outbound SMTP.	No email notifications will be sent.	✅	None
`federation-service`	Stores user identities (LDAP).	If LDAP authentication is in use, all logins will fail and some REST API calls might fail.	🚒 only if LDAP in use	`postgres`, `frontend`, `federations-service-migrator`

Container / Image	Role	What happens if it fails?	Failure severity	Startup dependencies
`federation-service-migrator`	Runs postgresql migrations for the `federations-service`.	Only runs at startup.	✓	`postgres`, `frontend`
`fileserved`	File storage service used as a replacement for S3 when CircleCI Server is run outside of AWS. Not used if Server is configured to use S3. Stores step output logs, artifacts, test results, caches and workspaces.	If not using S3, builds will produce no outputand some REST API calls might fail.	🧯 if **not** using S3	None
`frontend`	CircleCI web app and www-api proxy.	The UI and REST API will be unavailable and no jobs will be triggered by GitHub/Enterprise. Running builds will be OK but no updates will be seen.	⚠	`postgres`
`mongo` *	Mongo data store.	Potential total data loss. All running builds will fail and the UI will not work.	🧯	`mongodb-upgrader`
`nomad-metrics`	Queries the nomad server for stats and sends them to statsd.	Nomad metrics will be lost, but everything else should run as normal.	✓	None
`output-processor` / `output-processing`	Receives job output & status updates and writes them to MongoDB. Also provides an API to running jobs to access caches, workspaces, store caches, workspaces, artifacts, & test results.	All running builds will either fail or be left in an unfixable, inconsistent state. There will also be data loss in terms of step output, test results and artifacts.	🧯	None

Container / Image	Role	What happens if it fails?	Failure severity	Startup dependencies
`permissions-service`	Provides the CircleCI permissions interface.	Workflows will fail to start and some REST API calls may fail, causing 500 errors in the UI.	⚠	`postgres`, `frontend`, `permissions-service-migrator`
`permissions-service-migrator`	Runs postgresql migrations for the `permissions-service`	Only runs at startup.	✓	`postgres`, `frontend`
`picard-dispatcher`	Splits a job into tasks and sends them to `schedulerer` to be run.	No jobs will be sent to Nomad, the run queue will increase in size but there should be no meaningful loss of data.	⚠	None
`postgres` / `postgres-script-enhance` *	Basic `postgresql` with enhancements for creating required databases when containers are launched.	Potential total data loss. All running builds will fail and the UI will not work.	🚨	None
`rabbitmq` / `rabbitmq-delayed` *	Runs the RabbitMQ server. Most of our services use RabbitMQ for queueing.	Potential total data loss. All running builds will fail and the UI will not work.	🚨	None
`outputRunningRedis` / `redis` *	The Redis key/value store.	Lose output from currently-running job steps. API calls out to GitHub may also fail.	⚠	None
`schedulerer`	Sends tasks to `server-nomad` to run. \	No jobs will be sent to Nomad, the run queue will increase in size but there should be no meaningful loss of data.	⚠	None
`mongodb-upgrader` / `server-mongo-upgrader`	Used to run any mongo conversion/upgrade scripts during mongo version upgrade.	Not required to run all the time. \	✓	None
`nomad_server` / `server-nomad` *	Nomad primary service.	No 2.0 build jobs will run.	🚨	None

Container / Image	Role	What happens if it fails?	Failure severity	Startup dependencies
`ready-agent` / `server-ready-agent`	Called by Replicated to check whether other containers are ready.	Only required on startup. If unavailable on startup the whole system will fail.	✓	None
`server-usage-stats`	Sends the user count to the internal CircleCI “phone home” endpoint.	CircleCI will not receive usage stats for your install but no affect on operation.	✓	None
`shutdown-hook-poller`	Checks the `frontend` container for 1.0 Builder shutdown requests. If a request is found, the 1.0 Builder is shut down.	1.0 Builder lifecycles will not be properly managed, but jobs will continue to run.	✓	None
`slanger`	Provides real-time events to the CircleCI app.	Live UI updates will stop but hard refreshes will still work.	✓	None
`telegraf`	This is the statsd forwarding agent that our local services write to and can be configured to forward to an external metrics service.	Metics will stop working but jobs will continue to run.	✓	None
`tutum/logrotate`	Used to manage log rotations for all containers on the services machine.	If this stays down for a long period the Services machine disk will eventually run out of space and other services will fail.	⚠	None
`test-results`	Parses test result files and stores data.	There will be no test failure or timing data for jobs, but this will be back-filled once the service is restarted.	✓	None

Container / Image	Role	What happens if it fails?	Failure severity	Startup dependencies
`contexts-vault` / `vault-cci` *	Instance of Hashicorp's Vault – an encryption service that provides key-management, secure storage, and other encryption related services. Used to handle the encryption and key store for the `contexts-service`.	`contexts-service` will stop working, and all jobs that use `contexts-service` will fail.	⚠	None
`vm-gc`	Periodically check for stale `machine` and remote Docker instances and request that `vm-service` remove them.	Old vm-service instances might not be destroyed until this service is restarted.	✓	`vm-service-db-migrator`
`vm-scaler`	Periodically requests that `vm-service` provision more instances for running `machine` and remote Docker jobs.	VM instances for `machine` and Remote Docker might not be provisioned causing you to run out of capacity to run jobs with these executors.	⚠	`vm-service-db-migrator`
`vm-service`	Inventory of available `vm-service` instances, and provisioning of new instances.	Jobs that use `machine` or remote Docker will fail.	⚠	`vm-service-db-migrator`
`vm-service-db-migrator`	Used to run database migrations for `vm-service`.	Only runs at startup.	✓	None
`workflows-conductor`	Coordinates and provides information about workflows.	No new workflows will start, currently running workflows might end up in an inconsistent state, and some REST and GraphQL API requests will fail.	🧯	`postgres`, `frontend`, `workflows-conductor-migrator`
`workflows-conductor-migrator`	Runs postgreSQL migrations for the `workflows-conductor`.	Only runs on startup.	✓	`postgres`, `frontend`

CircleCI Server AWS S3 Storage Lifecycle Guide

This guide is intended to help system administrators of CircleCI Server installations on AWS better understand how S3 storage is used. This can help to cut compute resource costs and avoid issues for service users when removing resources.

In CircleCI Server, S3 Lifecycle policies are not configured automatically. The figures provided in this guide are examples to help you create a lifecycle policy for your installation. For more information on the steps required to add policy rules, see the AWS Lifecycle Policy Guide.

Example Lifecycle Policy Configuration for S3 Buckets

Prefix	Tag	Suggested Number of Days	Description of files	Side effects of deletion
`artifacts`	(null)	30	General artifacts (deliverables from `store_artifacts` and `store_test_results`)	Artifacts will disappear from the list of artifacts in job results. URLs will return 404.
`artifacts`	`"circleci.object_type" === "project.cache"`	15	Dependency caches	The first build after deletion will be slow due to cache miss.
`artifacts`	`"circleci.object_type" === "workflow.workspace"`	15	Workspaces	If a job depending on workspaces is rerun (e.g. SSH rerun), `attach_workspace` will restore no files, and subsequent steps will fail.
`artifacts/picard-task-configs`	(null)	1	Task config	None
`action-logs`	(null)	365	Outputs from each step	Outputs will not be shown any longer. Loading indicator will remain even after opening a pull-down for each step.
`cache`	(null)	15	Legacy cache	This folder is not actively used. If a job depends on these files, the first build after deletion will be slow due to cache miss.

Additional Locations Not Listed Above

`audit-logs`

This is where audit logs are saved. Deleting this folder and files under this fodler has no impact on future operation of CircleCI, but deleted logs will be definitely lost and cannot be recovered.

`code-signing-keys`

This is where signing keys for macOS/iOS apps are stored. This folder is no longer used actively.

`test-results`

This is where statistic data collected with `store_test_results` resides. The data is used for future test splitting with `circleci tests split --split-by=timings`. **Deleting test results will cause CIRCLE BUG errors in future jobs using `parallelism`.**