CircleCI Server v2.16 Operations Guide

Reviewed and Tested Documentation

April 3rd, 2019

Contents

Overview

CircleCI is a modern continuous integration and continuous delivery (CI/CD) platform installable inside your private cloud or data center.

CircleCI 2.x provides new infrastructure that includes the following improvements:

- New configuration with any number of jobs and workflows to orchestrate them.
- Custom images for execution on a per-job basis.
- Fine-grained performance with custom caching and per-job CPU or memory allocation.

Refer to the v2.16 Changelog at https://circleci.com/server/changelog/#2-16-00 for what's new in this release.

Build Environments

CircleCI uses Nomad as the primary job scheduler in CircleCI 2.x. Refer to the 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.x 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. The Insights page in the CircleCI UI is a dashboard showing the health of all repositories you are following including median build time, median queue time, last build time, success rate, and parallelism.

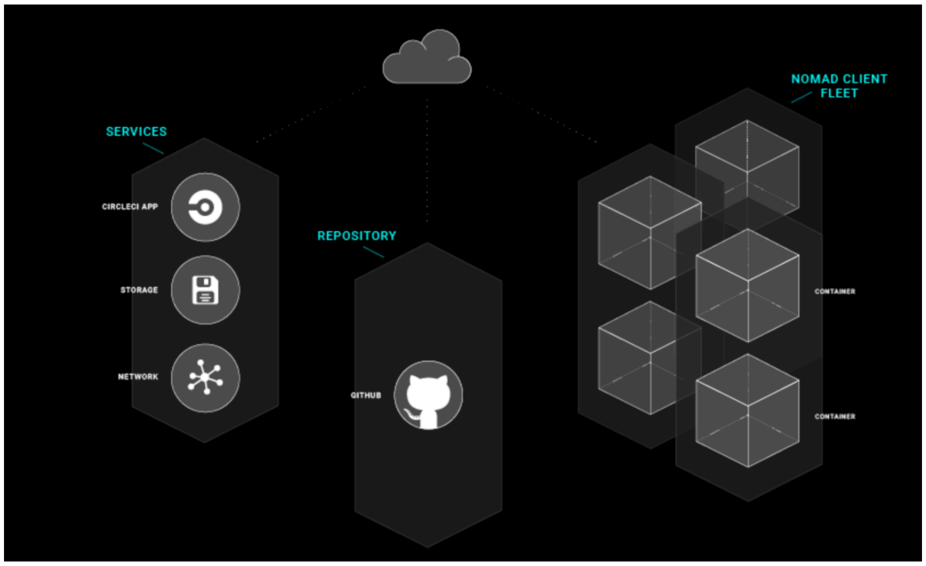


Figure 1: CircleCI Services Architecture

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 in the following architecture diagram.

Services Instance

The machine on which the Services instance runs 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 restart will result in downtime. Refer to the Disaster Recovery chapter for instructions.

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

Source	Ports	Use
End Users	80, 443 , 4434	HTTP/HTTPS Traffic
Administrators	22	SSH

Source	Ports	Use
Administrators	8800	Admin Console
Builder Boxes	all traffic / all ports	Internal Communication
GitHub (Enterprise or .com)	80, 443	Incoming Webhooks

Nomad Clients

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

To ensure that there are enough running to handle all of the builds, track the queued builds and increase the number of Nomad Client machines as needed to balance the load.

Each machine reserves two CPUs 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.

Note: 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 the 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. That is, CircleCI will inherit the authentication supported by your central SSO infrastructure. **Note:** 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 Builder instances.

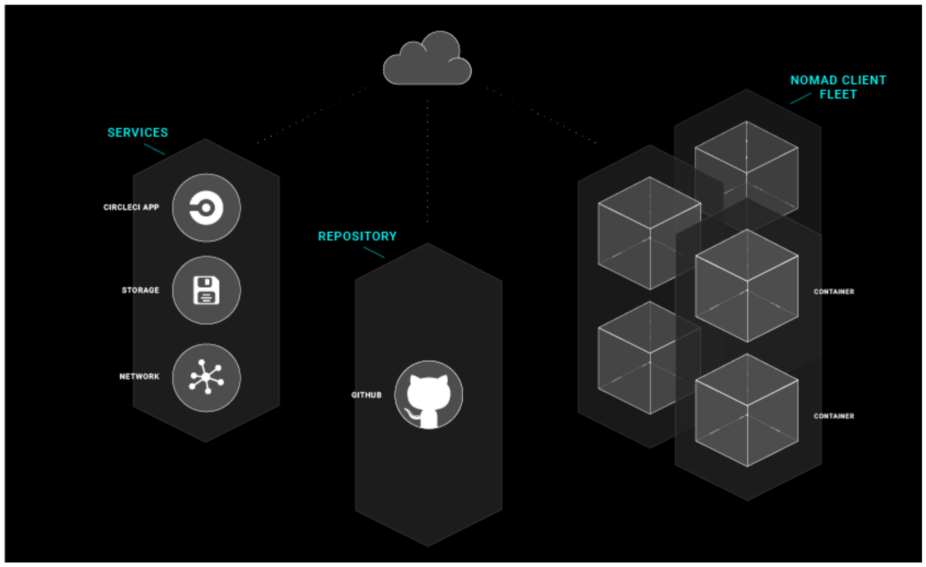


Figure 2: A Diagram of the CircleCI Architecture

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 with CircleCI

This document provides conceptual and procedural information for operating, backing up, monitoring, and configuring a CircleCI server installation.

CircleCI uses Nomad as the primary job scheduler in CircleCI 2.0. 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

- **Nomad Server:** Nomad Servers are the brains of the cluster. It receives and allocates jobs to Nomad clients. In CircleCI, a Nomad server is running in your service box as a Docker Container.
- **Nomad Client:** Nomad Clients execute jobs allocated by Nomad servers. Usually a Nomad client runs on a dedicated machine (often a VM) in order to fully take the advantage of its 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:** Nomad Job is a specification provided by users that declares a workload for Nomad. In CircleCI 2.0, a Nomad job corresponds to an execution of CircleCI job/build. If the job/build uses parallelism, say 10

Basic Operations

This section will give you the 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 `nomad status` command will give you the list of jobs status in your cluster. The `Status` is the most important field in the output with the following status type definitions:

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

Checking the Cluster Status

The `nomad node-status` command will give you the list of Nomad clients. Note that `nomad node-status` command also 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.

The `nomad node-status -self` command will give you more information about the client where you execute the command. Such information includes 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 CircleCI job/build. Therefore, checking logs of Nomad Jobs sometimes helps you to understand the status of CircleCI job/build if there is a problem.

The `nomad logs -job -stderr <nomad-job-id>` command will give you the logs of the job.

Note: Be sure to specify `-stderr` flag as most of logs from Build Agent appears in the `stderr`.

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 of this document. To learn more, please see the official document.

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

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

Scaling Up the Client Cluster

Refer to the Scaling section of the Configuration chapter for details about adding Nomad Client instances to an AWS auto scaling group and using a scaling policy to scale up automatically according to your requirements.

Shutting Down a Nomad Client

When you want to shutdown a Nomad client, you must first set the client to `drain` mode. In the `drain` mode, the client will finish already allocated jobs but will not get allocated 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 with `node-status` command:

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

Alternatively, you can drain a remote node with `nomad node-drain -enable -yes <node-id>`.

Scaling Down the Client Cluster

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

The script should use the above commands to put the instance in drain mode, monitor running jobs on the instance, wait for them to finish and then terminate the instance.

Configuration

This chapter describes configuration, customizations, and metrics.

Server Settings, Auto Scaling, and Monitoring

This section is for System Administrators who are setting environment variables for installed Nomad Clients, scaling their cluster, gathering metrics for monitoring their CircleCI installation, and viewing logs:

Advanced System Monitoring

Enable the ability to forward system and Docker metrics to supported platforms by going to Replicated Admin > Settings and enabling the provider of your choice, for example `https://example.com:8800/settings#cloudwatch_metrics`.

Metrics Details

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

Following are the metrics that are enabled:

- CPU
- Disk
- Memory
- Networking
- Docker

Nomad Job Metrics

In addition to the metrics above, Nomad job metrics are enabled and emitted by the Nomad Server agent. Five types of metrics are reported:

`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.

Supported Platform(s)

There are two built-in platforms; AWS CloudWatch and DataDog.

AWS CloudWatch

Click `Enable` under AWS CloudWatch to begin configuration.

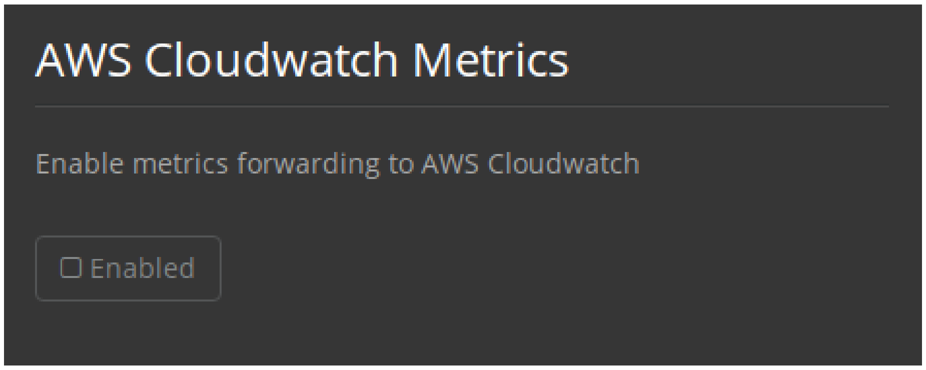


Figure 3: AWS CloudWatch

Configuration

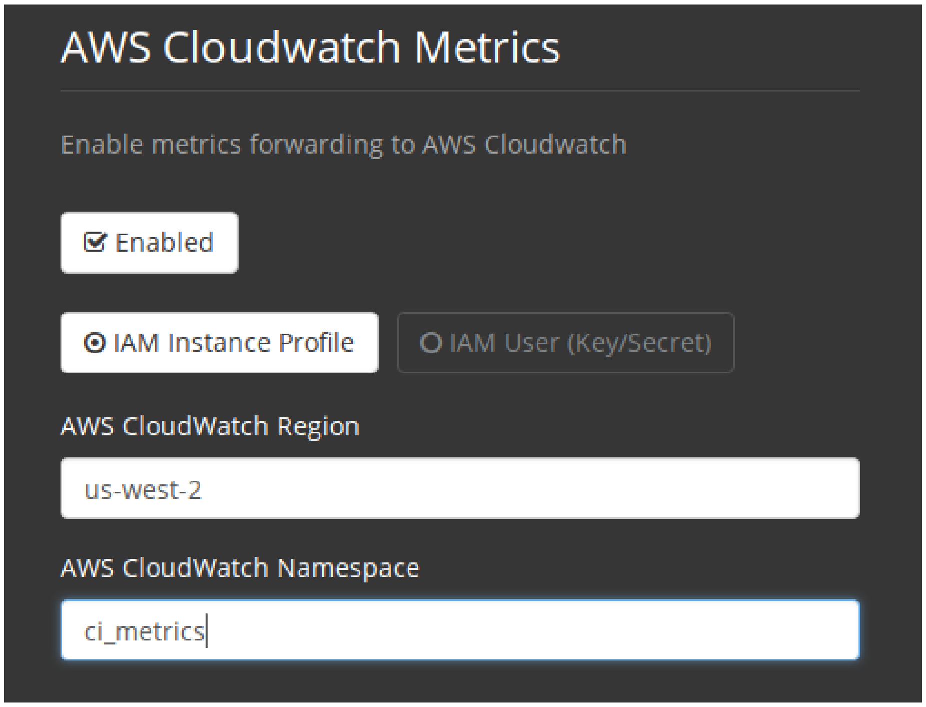
There are two options for configuration:

- Use the IAM Instance Profile of the services box and configure your custom region and namespace.
- Alternatively, you may use your AWS Access Key and Secret Key along with your custom region and namespace.

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

DataDog

Click `Enable` under DataDog Metrics to begin configuration.

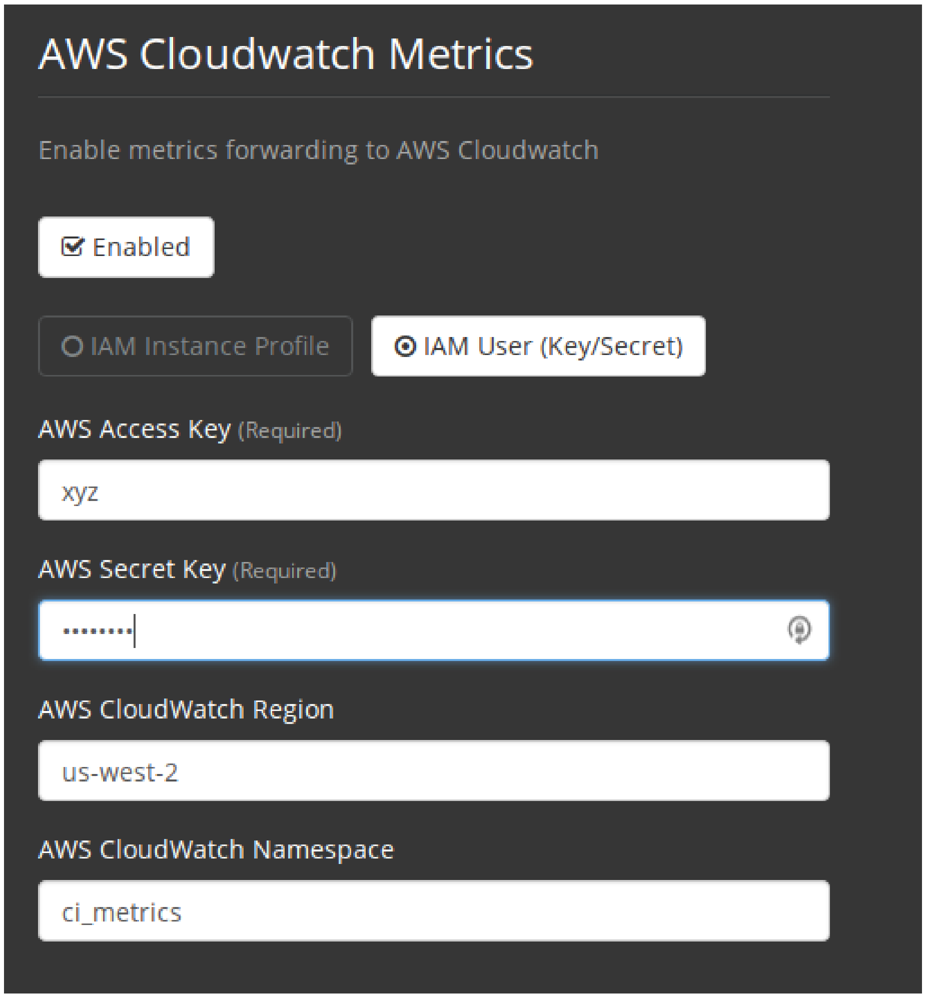

AWS Cloudwatch Metrics

Enable metrics forwarding to AWS Cloudwatch

AWS CloudWatch Region: us-west-2

AWS CloudWatch Namespace: ci_metrics

Figure 4: Configuration IAM



AWS Cloudwatch Metrics

Enable metrics forwarding to AWS Cloudwatch

[✓ Enabled]

IAM Instance Profile [✓ IAM User (Key/Secret)]

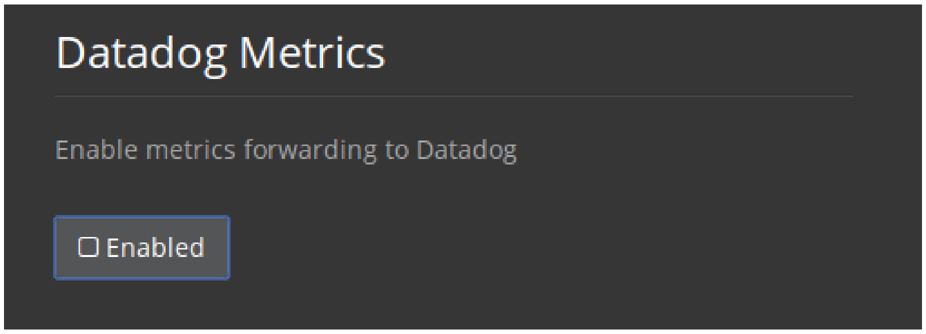
AWS Access Key (Required): xyz

AWS Secret Key (Required): ••••••••

AWS CloudWatch Region: us-west-2

AWS CloudWatch Namespace: ci_metrics

Figure 5: Configuration Alt



Datadog Metrics

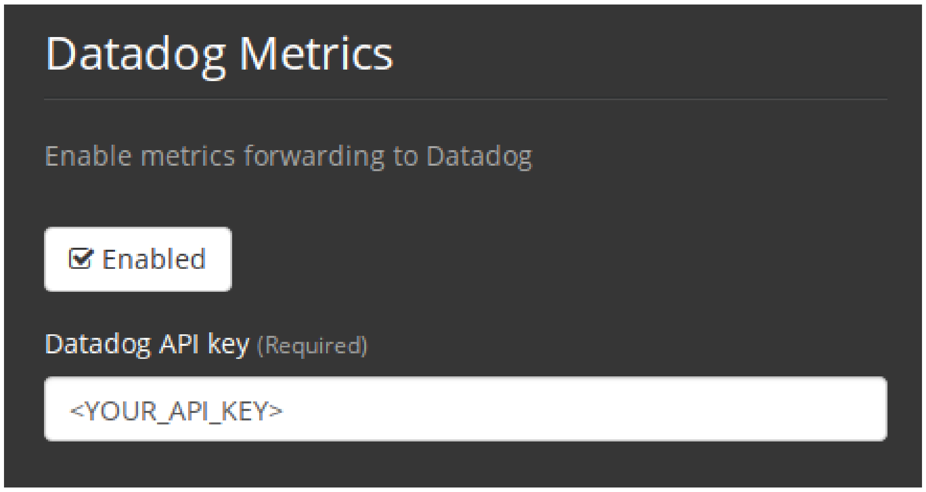
Enable metrics forwarding to Datadog

☐ Enabled

Figure 6: DataDog

Configuration

Enter your DataDog API Key.


Datadog Metrics

Enable metrics forwarding to Datadog

Datadog API key (Required)

<YOUR_API_KEY>

Figure 7: DataDog

After saving you can *verify* that metrics are forwarding by going to the DataDog metrics summary.

Custom Metrics

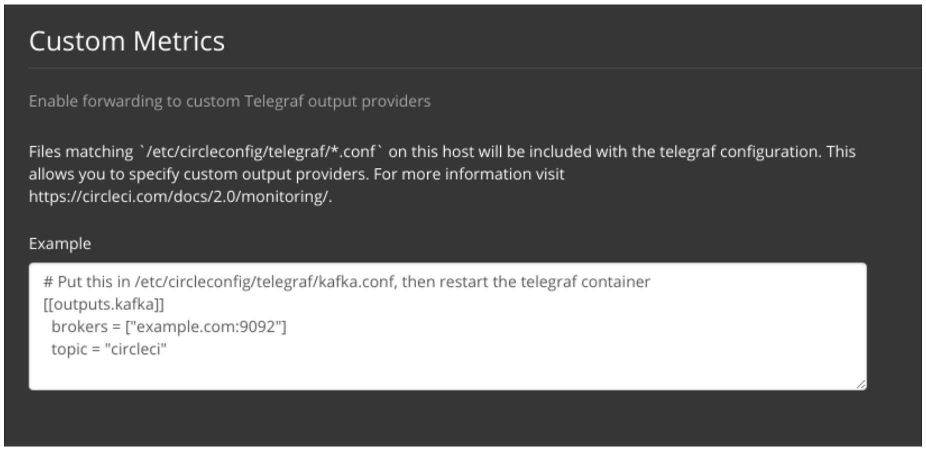
Custom Metrics via Telegraf configuration file may be configured in addition to the predefined CloudWatch and Datadog metric metrics described above. Telegraph can also be used instead of those sections for more fine grained control.

Configuration

Configuration options are based on Telegraf's documented output plugins. See their documentation here.

For example, if you would like to use the InfluxDB Output Plugin you would need to follow these steps; 1. SSH into the Servics Machine 2. cd `/etc/circleconfig/telegraf/influxdb.conf` 3. Adding the desired ouputs, for example

```
[[output.influxdb]]
  url = "http://52.67.66.155:8086"
  database = "testdb"
```


Custom Metrics

Enable forwarding to custom Telegraf output providers

Files matching `/etc/circleconfig/telegraf/*.conf` on this host will be included with the telegraf configuration. This allows you to specify custom output providers. For more information visit https://circleci.com/docs/2.0/monitoring/.

Example

```
# Put this in /etc/circleconfig/telegraf/kafka.conf, then restart the telegraf container
[[outputs.kafka]]
  brokers = ["example.com:9092"]
  topic = "circleci"
```

Figure 8: Custom

4. Run `docker restart telegraf` to restart the container to load or reload any changes.

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.

Note: Any changes to the config will require a restart of the system.

Scheduled Scaling

By default, an Auto Scaling Group (ASG) is created on your AWS account. Go to your EC2 Dashboard and select Auto Scaling Groups from the left side menu. Then, in the Instances tab, set the Desired and Minimum number to define the number Nomad Clients to spin up and keep available. Use the Scaling Policy tab of the Auto Scaling page to scale up your group automatically only at certain times, see below for best practices for defining policies.

Refer to the Shutting Down a Nomad Client section of the Nomad document for instructions on draining and scaling down the Nomad Clients.

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 modeling 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.

Logging For 1.0

Collecting and centralizing logs is an essential component of monitoring. The logs provide audit trails as well as debugging information for infrastructure failures. This document describes how you can integrate CircleCI with your logging solution in the following sections:

Basic System Monitoring with CloudWatch

Enable CloudWatch by going to Replicated Admin > Settings > Enhanced AWS Integration (1.0 Only) > Enable Cloudwatch, for example, `https://example.com:8800/settings#enhan`
Note: CloudWatch does **not** support monitoring of macOS containers.

CloudWatch already monitors the health and basic checks for EC2 instances, for example, CPU, memory, disk space, and basic counts with alerts. Consider upgrading machine types for the Services instance or decrease the number of CPUs per container if CPU or memory become a bottleneck.

Installing Logging Appliance Agents

CircleCI 1.0 Builders store logs in `/var/log/**/*.log` except for Docker, which stores logs in `/var/lib/docker/containers/**/*-json.log`.

Logging appliances generally require installation of a custom agent on each machine and configuration that collects logs and forwards them to a service. Some available logging appliances are:

- Amazon Cloudwatch Logs
- Graylog
- LogDNA
- Logstash
- Splunk

Configure the agent according to the environment, the authentication mechanisms, and centralized logging service discovery mechanism. You can reuse your current practices for setting up the agent and configuration.

If you are using CircleCI Terraform/CloudFormation templates, you can modify the launch configuration to add the hook to install the agent and run it as follows:

```
#!/usr/bin/bash


#### Log configuration - using Amazon CloudWatch as an example

# Install the agent, using Amazon CloudWatch as an example
wget https://s3.amazonaws.com/aws-cloudwatch/downloads/latest/aws
agent-setup.py

# Configure agent
cat <<EOF >/root/awslogs.conf
[general]
state_file = /var/awslogs/state/agent-state

[/var/log/circle-builder/circle.log]
datetime_format = %Y/%m/%d %H:%M:%S
file = /var/log/circle-builder/circle.log
buffer_duration = 5000
log_stream_name = {instance_id}
initial_position = end_of_file
log_group_name = /var/log/circle-builder/circle.log
EOF


## Run agent
```

```
python ./awslogs-agent-setup.py --region us-west-
2 --non-interactive --configfile=/root/awslogs.conf

#### Run CircleCI Builder as typical

curl https://s3.amazonaws.com/circleci-enterprise/init-
builder-0.2.sh | \
   SERVICES_PRIVATE_IP=<private ip address of services box> \
   CIRCLE_SECRET_PASSPHRASE=<passphrase entered on system console (service
     bash
```

If you are using an orchestration tool, for example Chef, Puppet, or SaltStack, it is possible to apply the appropriate recipe or cookbook to the builder instances.

Integrating With Syslog

CircleCI 1.0 Builders integrate with the `syslog` facility. `Syslog` is a widely used standard for logging, and most agents integrate with it seamlessly. Configure the builder machines to emit logs to the `syslog` facility by setting `CIRCLE_LOG_TO_SYSLOG` to `true` in the launch configuration:

```
#!/bin/bash
curl https://s3.amazonaws.com/circleci-enterprise/init-
builder-0.2.sh | \
     CIRCLE_LOG_TO_SYSLOG=true \
   SERVICES_PRIVATE_IP=<private ip address of services box> \
   CIRCLE_SECRET_PASSPHRASE=<passphrase entered on system console (service
     bash
```

Then, configure `syslog` to forward logging to a centralized `rsyslog` server, or configure a local logging agent to monitor the `syslog` rather than monitor files.

The Services machine uses Docker. It is possible to customize the Docker daemon to route logs to your desired supported destination, see the Docker documentation on logging drivers for details.

Note: Many tools default to file-based logging, and *using the syslog facility as the only mode of logging may accidentally ignore important logging info*. Configuring custom agents to watch all of `/var/log/**/*` will result in capturing most logging files including `syslog`.

Setting up HTTP Proxies

This document describes how to configure CircleCI to use an HTTP proxy in the following sections:

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. Run `export NO_PROXY=<services_box_ip>` to add it to the `NO_PROXY` rules. In an ideal case, traffic to S3 will not be proxied, and 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.

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

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

Corporate Proxies

Also note that when the instructions ask you if you use a proxy, they will also prompt you for the address. It is **very important** that you input the proxy in the following format `<protocol>://<ip>:<port>`. If you are missing any part of that, 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 whitelisted 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 (http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-instance-metadata.html). 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 using AWS Terraform install you'll have to 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 using static installation, add 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
```

You'll also need to follow these instructions: https://docs.docker.com/network/proxy/ for making sure that your containers have outbound/proxy access.

Troubleshooting

If you cannot access the page of the CircleCI Replicated management console, but the services machine seems to be running, try to SSH tunnel into the machine doing the following: `ssh -L 8800:<address you want to proxy through>:8800 ubuntu@<ip_of_services_box>`.

Data Persistence

Refer to the following documents for instructions to configure your installation for data persistence:

- Adding External Databases and Vault Hosts to CircleCI v2.15
- Adding External Redis, Rabbitmq, Slanger, and Nomad Services to CircleCI v2.16

Configuring LDAP Authentication

This document describes how to enable, configure, and test CircleCI to authenticate users with OpenLDAP or Active Directory credentials.

Prerequisites

- Install and configure your LDAP server and Active Directory.
- GitHub Enterprise must be configured and is the source of organizations and projects to which users have access.
- Install a new instance of CircleCI 2.0 with no existing users using the Installing CircleCI 2.0 on Amazon Web Services with Terraform document. **Note:** LDAP is not supported with existing installations, only clean installations may use LDAP.
- Contact CircleCI support and file a feature request for CircleCI installed on your own servers.

Note: After completing this configuration, all users must log in to CircleCI with their LDAP credentials. After logging in to CircleCI, each user will then click the Connect button on the Accounts page to connect and authenticate their GitHub account.

Configure LDAP Authentication

This section provides the steps to configure LDAP in the management console (Replicated).

1. Verifying access over the LDAP/AD ports to your LDAP/AD servers.
2. Log in to the management console for a newly installed CircleCI 2.0 instance as the `admin` user.
3. Check the LDAP button on the Settings page. Select OpenLDAP or Active Directory. A Screenshot of the LDAP Settings Page
4. Fill in your LDAP instance Hostname and port number.
5. Select the encryption type (plain text is not recommended).
6. Fill in the Search user field with the LDAP admin username using the format `cn=<admin>,dc=<example>,dc=<org>` replacing `admin`, `example`, and `org` with appropriate values for your datacenter.
7. Fill in the Search password field with the LDAP admin password.
8. Fill in the User search DN field with an approrpiate value using the format `ou=<users>` replacing `users` with the value used in your LDAP instance.
9. Fill in the Username field with an approriate unique identifier used for your users, for example, `mail`.
10. Fill in the Group Membership field with an appropriate value. By default, the value is `uniqueMember` for OpenLDAP and `member` for Active Directory. This field will list member `dn` for a group.
11. Fill in the Group Object Class field with an approrpiate value. By default, the value is `groupOfUniqueNames` for OpenLDAP and `group` for

Active Directory. The value of the `objectClass` field indicates a `dn` is a group.
12. (Optional) Fill in the Test username and Test password fields with a test email and password for an LDAP user you want to test.
13. Save the settings.

A user who logs in will be redirected to the Accounts page of the CircleCI application 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.

Note: 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. A users' ability to see projects or to run builds is defined by their GitHub permissions. 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>
```

Using the `machine` Executor and Remote Docker Jobs

This document outlines how to set up VM service for your CircleCI installation for `machine` executor and remote Docker jobs, as well as how customize your own VM service images. **Note:** This configuration is only available for installations on AWS, please contact your CircleCI account representative to request this configuration for static.

Overview

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

Configuration

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

If you do not provide a custom Amazon Machine Image (AMI) for VM service, `machine` executor and remote Docker jobs on Server will run using the same machine image that we provide by default on Cloud: an Ubuntu 14.04 or 16.04 image with Docker version `17.03.0-ce` and docker-compose version `1.9.0`, along with a selection of common languages, tools, and frameworks. See the `picard-vm-image` branch of our image-builder repository for details.

Customization

It may be beneficial to customize the VM service image for your installation of CircleCI; it will allow you to specify other versions of Docker and docker-compose, as well as install any additional dependencies that may be part of your CI/CD pipeline. Without doing so, you will likely need to run these additional install and update steps on every commit as part of your `config.yml` file.

To build custom VM service images, use the following repository branch: https://github.com/circleci/image-builder/tree/picard-vm-image.

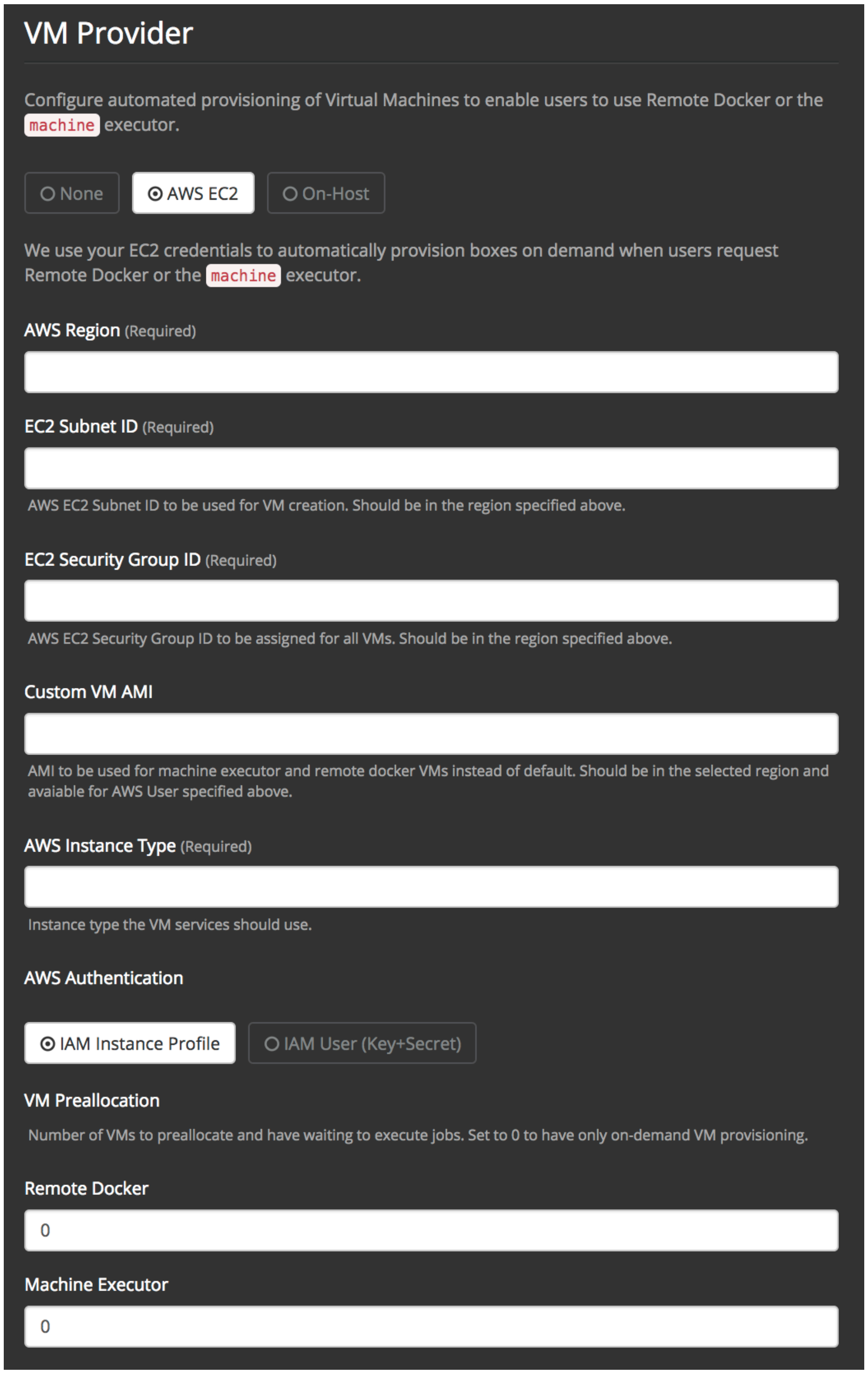
Run the `packer build aws-vm.json` command after filling in the required groups in `aws-vm.json`. It requires an access key and secret key to upload. Handle the key and secret process according to the your requirements, but consider restricting the `ami_groups` to only within your organization.

Refer to https://packer.io/docs/builders/amazon-ebs.html#ami_groups for more information and see https://github.com/circleci/image-builder/blob/picard-vm-image/provision.sh for details about settings.

You will need to associate the `circleci` user with the image you want to use as shown in the following example: https://github.com/circleci/image-builder/blob/picard-vm-image/aws_user_data.

Customizations

This section is a brief summary of key files and variables that impact Server behavior.



VM Provider

Configure automated provisioning of Virtual Machines to enable users to use Remote Docker or the `machine` executor.

None [✓ AWS EC2] On-Host

We use your EC2 credentials to automatically provision boxes on demand when users request Remote Docker or the `machine` executor.

AWS Region (Required)

EC2 Subnet ID (Required)

AWS EC2 Subnet ID to be used for VM creation. Should be in the region specified above.

EC2 Security Group ID (Required)

AWS EC2 Security Group ID to be assigned for all VMs. Should be in the region specified above.

Custom VM AMI

AMI to be used for machine executor and remote docker VMs instead of default. Should be in the selected region and avaiable for AWS User specified above.

AWS Instance Type (Required)

Instance type the VM services should use.

AWS Authentication

[✓ IAM Instance Profile] IAM User (Key+Secret)

VM Preallocation

Number of VMs to preallocate and have waiting to execute jobs. Set to 0 to have only on-demand VM provisioning.

Remote Docker: 0

Machine Executor: 0

Figure 9: Configuring VM Service on CircleCI Server

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

/etc/circle-installation-customizations properties

Note: Every property should be in 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	

Setting Up Certificates

This document provides a script for using a custom Root Certificate Authority and the process for using an Elastic Load Balancing certificate in the following sections:

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 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 (Elastic Load Balancing) 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.

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

1. Open the following ports on your ELB:

Load BalancerProtocol	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

{: class="table table-striped"}

2. Add the following security group on your ELB:

Note: 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. **NOTE:** 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 from within the Replicated console (hostname: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'll be instructed to add a DNS TXT record.
5. After the record is successfully generated, save `fullchain.pem` and `privkey.pem` locally.

If you're 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.

Reference: https://letsencrypt.readthedocs.io/en/latest/using.html#manual

Ensure the hostname is properly configured in the Replicated/management console ~ (hostname:8800/settings) **and** that the hostname used matches the DNS records associated with the TLS certificates.

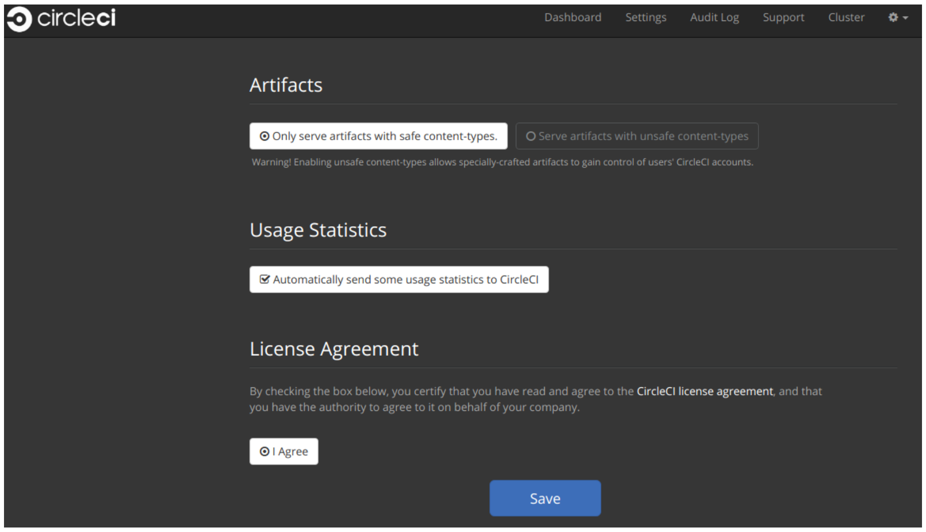
Make sure the Auth Callback URL in Github/Github Enterprise matches the domain name pointing to the services box, including the protocol used, for example **https**://info-tech.io/.

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.

Opt-In to this feature by going to Settings > Usage Statistics on the management console in Replicated. Then, enable the radio button labeled Automatically send some usage statistics to CircleCI as shown in the following screenshot.



Artifacts

[✓ Only serve artifacts with safe content-types.] Serve artifacts with unsafe content-types

Warning! Enabling unsafe content-types allows specially-crafted artifacts to gain control of users' CircleCI accounts.

Usage Statistics

[✓ Automatically send some usage statistics to CircleCI]

License Agreement

By checking the box below, you certify that you have read and agree to the **CircleCI license agreement**, and that you have the authority to agree to it on behalf of your company.

[✓ I Agree]

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	

Name	**Type**	**Purpose**
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	
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.

Disaster Recovery

This chapter describes failover or replacement 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.

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:

Note: If you are running CircleCI in an HA configuration, you must use standard backup mechanisms for the external datastores. See the High Availability 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.

Note: "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.
- **iOS Builds** If you are paying to run iOS builds on CircleCI hardware your source code will be downloaded to a build box on our macOS fleet where it will be compiled and any tests will be run. Similar to our primary build containers that you control, the iOS builds we run are sandboxed such that they cannot be accessed.

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 separarate 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 re-

quest 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).

Troubleshooting

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

FAQ

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 document for details. Refer to the Administrative Variables, Monitoring, and Logging 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)?

If you're experiencing any issues with Replicated, here are a few ways to debug it.

Check the current version of Replicated installed

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

```
replicated -version
```

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 restart replicated-ui
sudo restart replicated
sudo restart replicated-agent
```

For Ubuntu 16.04, run the following commands:

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

Then, go to your services box admin (for example, https://YOUR-CCIE-INSTALL:8800) and try restarting with "Stop Now" and "Start Now".

Try to log into Replicated

Try to log in to Replicated. You can do this by running the following commands on the service box. You will only be asked to enter password, which is the same one used to unlock the admin (i.e.: https://YOUR-CCIE-INSTALL:8800).

```
replicated login
```

If you could login, then please run the following command too and give us the output.

```
sudo replicated apps
```

You are getting Error: `request returned Unauthorized for API route`.. error probably because you are not logged into Replicated, so please check if you are still getting the error after successful login.

Check Replicated logs

You can find Replicated logs under `/var/log/replicated`.

Check output of docker ps

Replicated starts many Docker containers to run CCIE, so it may be useful to check what containers are running.

You should see something similar to this output:

```
 sudo docker ps
CONTAINER ID      IMAGE
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
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
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
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                                                      federati
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
service
eb22d3c10dd8         172.31.72.162:9874/circleci-
audit-log-service:0.1.587-fa47042      "circleci-
service-run"   26 hours
ago      Up 26 hours
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                                                         fron
af34ca3346a7         172.31.72.162:9874/circleci-
picard-dispatcher:0.1.10401-aa50e85     "circleci-
service-run"   26 hours
ago      Up 26 hours
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
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
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
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
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
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
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
3cbbd959f42e      172.31.72.162:9874/telegraf:1.6.4
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                                                    t
15b090e8cc02        172.31.72.162:9874/circleci-
schedulerer:0.1.10388-741e1d1          "circleci-
```

```
service-run"   26 hours
ago     Up 26 hours
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
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
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
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
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
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
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
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
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
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
cci
b3e317c9d62f      172.31.72.162:9874/redis:4.0.10
entrypoint.s…"   26 hours
ago          Up 26 hours           0.0.0.0:6379-
>6379/tcp
f2d3f77891f0         172.31.72.162:9874/circleci-
nomad-metrics:0.1.90-1448fa7         "/usr/local/bin/dock…"  26
ago     Up 26 hours
metrics
1947a7038f24      172.31.72.162:9874/redis:4.0.10
entrypoint.s…"   26 hours
ago          Up 26 hours           0.0.0.0:32846-
>6379/tcp
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
97ebdb831a7e      registry.replicated.com/library/retraced:1.2.2
aud…"   26 hours
ago     Up 26 hours      3000/tcp
processor
a0b806f3fad2      registry.replicated.com/library/retraced:1.2.2
aud…"   26 hours
ago         Up 26 hours          172.17.0.1:32771-
>3000/tcp
api
19dec5045f6e      registry.replicated.com/library/retraced:1.2.2
c '/usr/lo…"   26 hours
ago     Up 26 hours      3000/tcp
cron
7b83a3a193da      registry.replicated.com/library/retraced-
```

```
postgres:10.5-20181009     "docker-entrypoint.s…"  26 hours
ago     Up 26 hours      5432/tcp
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
nsqd
500619f53e80      quay.io/replicated/replicated-
operator:current                "/usr/bin/replicated…"  26 hours
ago     Up 26 hours
operator
e1c752b4bd6c     quay.io/replicated/replicated:current
d"       26 hours
ago     Up 26 hours      0.0.0.0:9874-9879->9874-
9879/tcp
1668846c1c7a      quay.io/replicated/replicated-
ui:current                    "/usr/bin/replicated…"  26 hours
ago         Up 26 hours           0.0.0.0:8800-
>8800/tcp
ui
f958cf3e8762     registry.replicated.com/library/premkit:1.2.0
ago      Up 26 hours      80/tcp, 443/tcp, 2080/tcp, 0.0.0.0:9880-
>2443/tcp
premkit
```

Providing support with the output of `sudo docker ps` in service box will be helpful in diagnosing the problem.

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, please generate a support bundle and contact our Support Engineers by opening a support ticket.

Debugging Queuing Builds

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

Check Dispatcher Logs for Errors

1. 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.

Check Picard-Dispatcher Logs for Errors

1. 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/real
Jan 9 19:34:45 INFO circle.http.builds project GERey/realitycheck a

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

```
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_artifa
```

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 CircleCI support.

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.

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

Check Picard-Scheduler Logs for Errors

1. 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.

Check Nomad Node Status

1. 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 guide for more detailed information on managing and troubleshooting the nomad server.

Note: 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 inidicates 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.

Check Job Processing Status

1. 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.

1. 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.
2. 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. **Note:** The use of `--stderr` is to print the specific error if one exists.
   - Run the `picard-output-processor` command to check those logs for specific errors.

Jobs stay in `queued` status until they fail and never successfully run.

- Check port 8585 if the nomad client logs contain the following type of error message: `{"error":"rpc error: code = Unavailable desc = grpc: the connection is unavailable","level":"warning","msg":"error fetching config, retrying","time":"2018-04-17T18:47:01Z"}`

Server Ports

This chapter provides System Administrators with a complete list of ports for the machines in their CircleCI 2.0 installation:

Machine type	Port number	Protocol	Direction	Source / destination	Use	Notes
Services Machine	80	TCP	Inbound	End users	HTTP web app traffic	
	443	TCP	Inbound	End users	HTTPS web app traffic	
	7171	TCP	Inbound	End users	Artifacts access	
	8081	TCP	Inbound	End users	Artifacts access	
	22	TCP	Inbound	Administrators	SSH	
	8800	TCP	Inbound	Administrators	Admin console	
	8125	UDP	Inbound	Nomad Clients	Metrics	
	8125	UDP	Inbound	Nomad Servers	Metrics	Only if using externalised Nomad Servers
	8125	UDP	Inbound	All Database Servers	Metrics	Only if using externalised databases

Machine type	Port number	Protocol	Direction	Source / destination	Use	Notes
	4647	TCP	Bi-directional	Nomad Clients	Internal communication	
	8585	TCP	Bi-directional	Nomad Clients	Internal communication	
	7171	TCP	Bi-directional	Nomad Clients	Internal communication	
	3001	TCP	Bi-directional	Nomad Clients	Internal communication	
	80	TCP	Bi-directional	GitHub Enterprise / GitHub.com (whichever applies)	Webhooks / API access	
	443	TCP	Bi-directional	GitHub Enterprise / GitHub.com (whichever applies)	Webhooks / API access	
	80	TCP	Outbound	AWS API endpoints	API access	Only if running on AWS
	443	TCP	Outbound	AWS API endpoints	API access	Only if running on AWS
	5432	TCP	Outbound	PostgreSQL Servers	PostgreSQL database connection	Only if using externalised databases. Port is user-defined, assuming the default PostgreSQL port.

Machine type	Port number	Protocol	Direction	Source / destination	Use	Notes
	27017	TCP	Outbound	MongoDB Servers	MongoDB database connection	Only if using externalised databases. Port is user-defined, assuming the default MongoDB port.
	5672	TCP	Outbound	RabbitMQ Servers	RabbitMQ connection	Only if using externalised RabbitMQ
	6379	TCP	Outbound	Redis Servers	Redis connection	Only if using externalised Redis
	4647	TCP	Outbound	Nomad Servers	Nomad Server connection	Only if using externalised Nomad Servers
	443	TCP	Outbound	CloudWatch Endpoints	Metrics	Only if using AWS CloudWatch
Nomad Clients	64535-65535	TCP	Inbound	End users	SSH into builds feature	
	80	TCP	Inbound	Administrators	CircleCI Admin API access	

Machine type	Port number	Protocol	Direction	Source / destination	Use	Notes
	443	TCP	Inbound	Administrators	CircleCI Admin API access	
	22	TCP	Inbound	Administrators	SSH	
	22	TCP	Outbound	GitHub Enterprise / GitHub.com (whichever applies)	Download Code From Github.	
	4647	TCP	Bi-directional	Services Machine	Internal communication	
	8585	TCP	Bi-directional	Services Machine	Internal communication	
	7171	TCP	Bi-directional	Services Machine	Internal communication	
	3001	TCP	Bi-directional	Services Machine	Internal communication	
	443	TCP	Outbound	Cloud Storage Provider	Artifacts storage	Only if using external artifacts storage
	53	UDP	Outbound	Internal DNS Server	DNS resolution	This is to make sure that your jobs can resolve all DNS names that are needed for their correct operation

Machine type	Port number	Protocol	Direction	Source / destination	Use	Notes
GitHub Enterprise / GitHub.com (whichever applies)	22	TCP	Inbound	Services Machine	Git access	
	22	TCP	Inbound	Nomad Clients	Git access	
	80	TCP	Inbound	Nomad Clients	API access	
	443	TCP	Inbound	Nomad Clients	API access	
	80	TCP	Bi-directional	Services Machine	Webhooks / API access	
	443	TCP	Bi-directional	Services Machine	Webhooks / API access	
PostgreSQL Servers	5432	TCP	Inbound	Services Machine	PostgreSQL database connection	Only if using externalised databases. Port is user-defined, assuming the default PostgreSQL port.

Machine type	Port number	Protocol	Direction	Source / destination	Use	Notes
	5432	TCP	Bi-directional	PostgreSQL Servers	PostgreSQL replication	Only if using externalised databases. Port is user-defined, assuming the default PostgreSQL port.
MongoDB Servers	27017	TCP	Inbound	Services Machine	MongoDB database connection	Only if using externalised databases. Port is user-defined, assuming the default MongoDB port.
	27017	TCP	Bi-directional	MongoDB Servers	MongoDB replication	Only if using externalised databases. Port is user-defined, assuming the default MongoDB port.

Machine type	Port number	Protocol	Direction	Source / destination	Use	Notes
RabbitMQ Servers	5672	TCP	Inbound	Services Machine	RabbitMQ connection	Only if using externalised RabbitMQ
	5672	TCP	Bi-directional	RabbitMQ Servers	RabbitMQ mirroring	Only if using externalised RabbitMQ
Redis Servers	6379	TCP	Inbound	Services Machine	Redis connection	Only if using externalised Redis
	6379	TCP	Bi-directional	Redis Servers	Redis replication	Only if using externalised Redis and using Redis replication (optional)
Nomad Servers	4646	TCP	Inbound	Services Machine	Nomad Server connection	Only if using externalised Nomad Servers
	4647	TCP	Inbound	Services Machine	Nomad Server connection	Only if using externalised Nomad Servers

Machine type	Port number	Protocol	Direction	Source / destination	Use	Notes
	4648	TCP	Bi-directional	Nomad Servers	Nomad Servers internal communication	Only if using externalised Nomad Servers