



# OPERATIONS GUIDE

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

Docs Team

Version 2.17.2, July 2nd, 2019

<b>Overview</b>	<b>1</b>
Build Environments	2
Architecture	3
Services Machine	3
<b>Introduction to Nomad Cluster Operation</b>	<b>6</b>
Basic Terminology and Architecture	7
Basic Operations	9
Checking the Jobs Status	9
Checking the Cluster Status	9
Checking Logs	9
Scaling Up the Client Cluster	10
Shutting Down a Nomad Client	10
Scaling Down the Client Cluster	10
System Monitoring	12
VM Service and Docker Metrics	12
Nomad Job Metrics	12
Supported Platforms	13
AWS CloudWatch	13
DataDog	13
Custom Metrics	15
Configuring Custom Metrics	15
Scheduled Scaling	17
Auto Scaling Policy Best Practices	17
Overview	18
Service Machine Proxy Configuration	19
Set up Service Machine Proxy Support	19
Corporate Proxies	20
Nomad Client Configuration	20
Nomad Client Proxy Setup	21
Troubleshooting	21
Data Persistence	22
<b>Authentication</b>	<b>23</b>
Prerequisites	24
Configure LDAP Authentication	25
Troubleshooting	27
Overview	28
Configuration	28
Customization	30
Job and Instance Management	31
Using a Custom Root CA	32

Setting up ELB Certificates .....	33
Using Self-Signed Certificates .....	33
Setting up TLS/HTTPS on CircleCI Server .....	35
Using Certbot .....	35
Adding the certificate to CircleCI Server .....	35

# Overview

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.



CircleCI Server v2.17 uses the CircleCI 2.0 architecture.

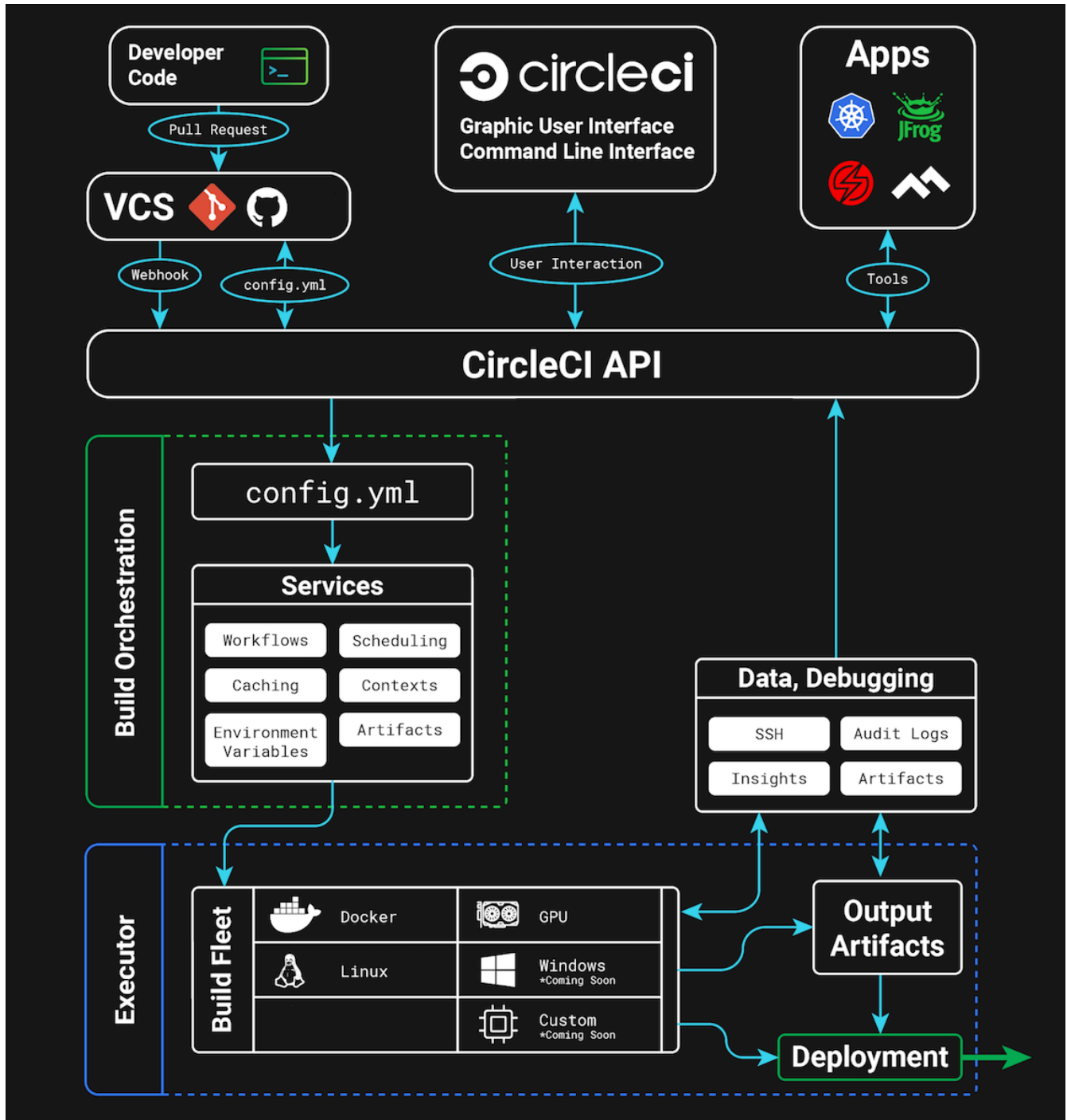


Figure 1. CircleCI Services Architecture

# Build Environments

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

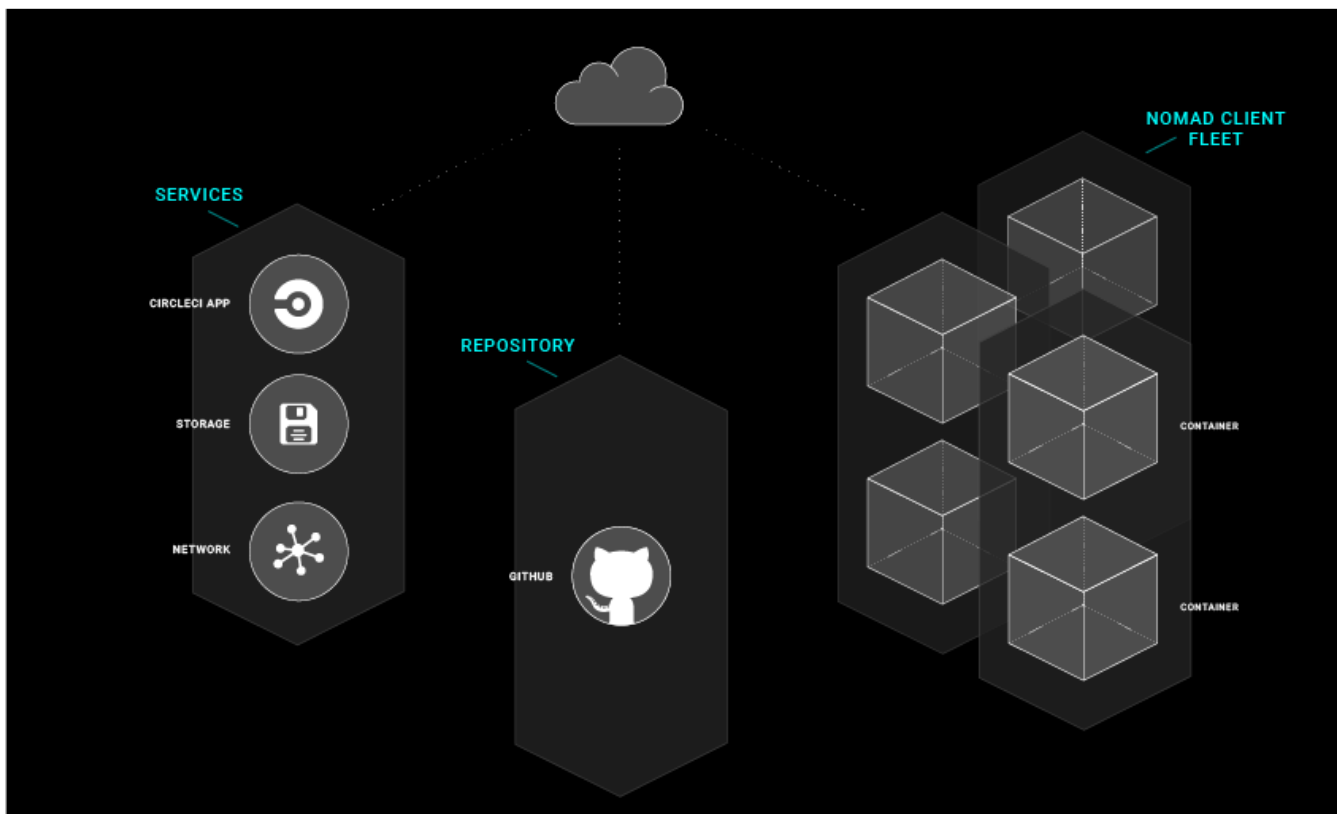
# Architecture

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 in Figure 2.



## 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 [\[Disaster Recovery\]](#) 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 [System Monitoring](#).

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

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

# Basic Terminology and Architecture

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.

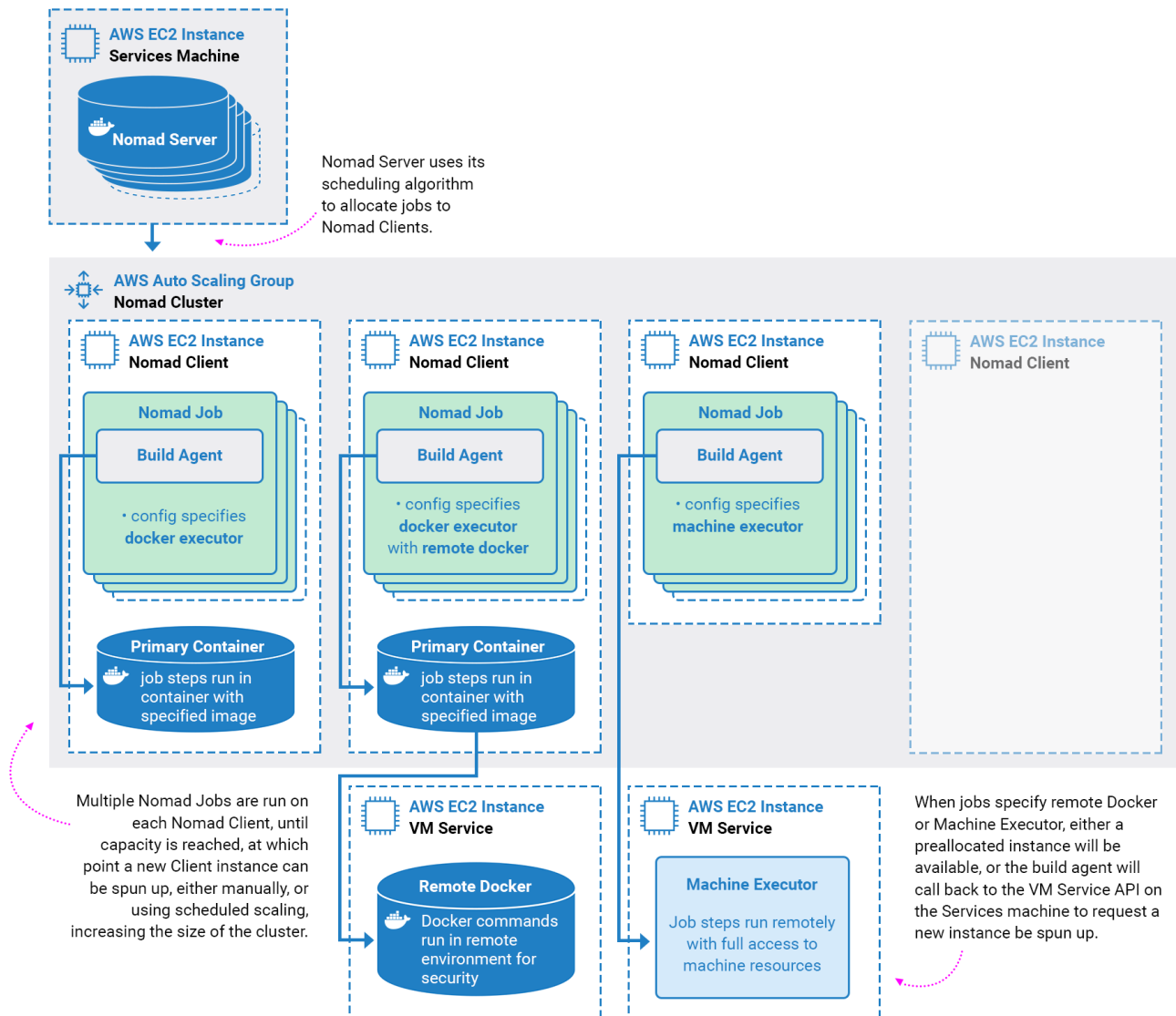


Figure 2. Nomad Cluster Management

- **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

To 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, checking logs of Nomad Jobs sometimes helps you 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 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 `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 `nomad node-drain -enable -yes <node-id>`.

## Scaling Down the Client Cluster

To set up a mechanism for clients to shutdown, first entering `drain` mode, then waiting for all jobs to be finished before terminating the client, you can 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 :page-layout: classic-docs :icons: font :toc: macro :toc-title:

This section includes information on gathering metrics for monitoring your CircleCI installation and scaling your Nomad cluster to meet your needs.

# System Monitoring

To enable system and Docker metrics forwarding to either AWS Cloudwatch or Datadog, navigate to your CircleCI Management Console, select Settings from the menu bar and scroll down to enable the provider of your choice. To get straight to this section use the following URL, substituting in your CircleCI URL:

```
https://<your-circleci-hostname>.com:8800/settings#cloudwatch_metrics
```

## VM Service and Docker Metrics

VM Host and Docker services metrics are forwarded via [Telegraf](#), a plugin-driven server agent for collecting and reporting metrics.

Following are the enabled metrics:

- [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
<code>circle.nomad.server_agent.poll_failure</code>	Returns 1 if the last poll of the Nomad agent failed, otherwise it returns 0.
<code>circle.nomad.server_agent.jobs.pending</code>	Returns the total number of pending jobs across the cluster.
<code>circle.nomad.server_agent.jobs.running</code>	Returns the total number of running jobs across the cluster.
<code>circle.nomad.server_agent.jobs.complete</code>	Returns the total number of complete jobs across the cluster.
<code>circle.nomad.server_agent.jobs.dead</code>	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 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:

```
`https://<your-circleci-  
hostname>.com:8800/settings#cloudwatch_metrics`
```

2. Check Enabled under AWS CloudWatch Metrics to begin configuration.

```
![AWS CloudWatch](images/metrics_aws_cloudwatch1.png){ width=400px }
```

## 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.

```
![Configuration IAM](images/metrics_aws_cloudwatch2a.png){  
width=400px }
```

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

```
![Configuration Alt](images/metrics_aws_cloudwatch2b.png){  
width=400px }
```

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

\pagebreak

## DataDog

To enable Datadog complete the following:



1. Navigate to the settings page within your Management Console. You can use the following URL, substituting your CircleCI URL:

```
`https://<your-circleci-hostname>.com:8800/settings#datadog_metrics`
```

2. Check Enabled under Datadog Metrics to begin configuration

```
![Enable DataDog](images/metrics_datadog1.png){ width=400px }
```

3. Enter your DataDog API Key

```
![DataDog API Key](images/metrics_datadog2.png){ width=400px }
```

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

# Custom Metrics

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

## 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"
```

## Configuring Custom Metrics

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 Services Machine
2. `cd /etc/circleconfig/telegraf/influxdb.conf`
3. Adding the desired outputs, for example:

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

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](<https://github.com/influxdata/influxdb#installation>) for default and advanced installation steps.

Note: Any changes to the config will require a restart of the system. <!--what system? the whole CircleCI system? incurring downtime?-->

```
\label{sec:scaling}
```

# Scheduled Scaling

By default, an Auto Scaling Group (ASG) is created on your AWS account for your Nomad cluster. 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 our guide to [shutting down a nomad client](#) for instructions on draining and scaling down the Nomad Clients.

## Auto Scaling Policy Best Practices

There is a [blog post series](https://circleci.com/blog/mathematical-justification-for-not-letting-builds-queue/) where 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 parallelism. 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 [\[Amazon's Step Policy\]](http://docs.aws.amazon.com/autoscaling/latest/userguide/as-scaling-simple-step.html) instructions to set this up along with Cloudwatch Alarms. = Setting Up HTTP Proxies :page-layout: classic-docs :icons: font :toc: macro :toc-title:

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 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

```



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

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](#). 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 running a 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 will also need to follow [these instructions](#) to make sure your containers have outbound/proxy access.

## Troubleshooting

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>
```



# Data Persistence

Contact [support@circleci.com](mailto:support@circleci.com) to discuss externalizing services for data persistence.

# Authentication

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



LDAP is not supported with existing installations, only clean installations may use LDAP.

# 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, following our [\[install\]](#) guide.
- Contact [CircleCI support](#) and file a feature request for CircleCI Server.

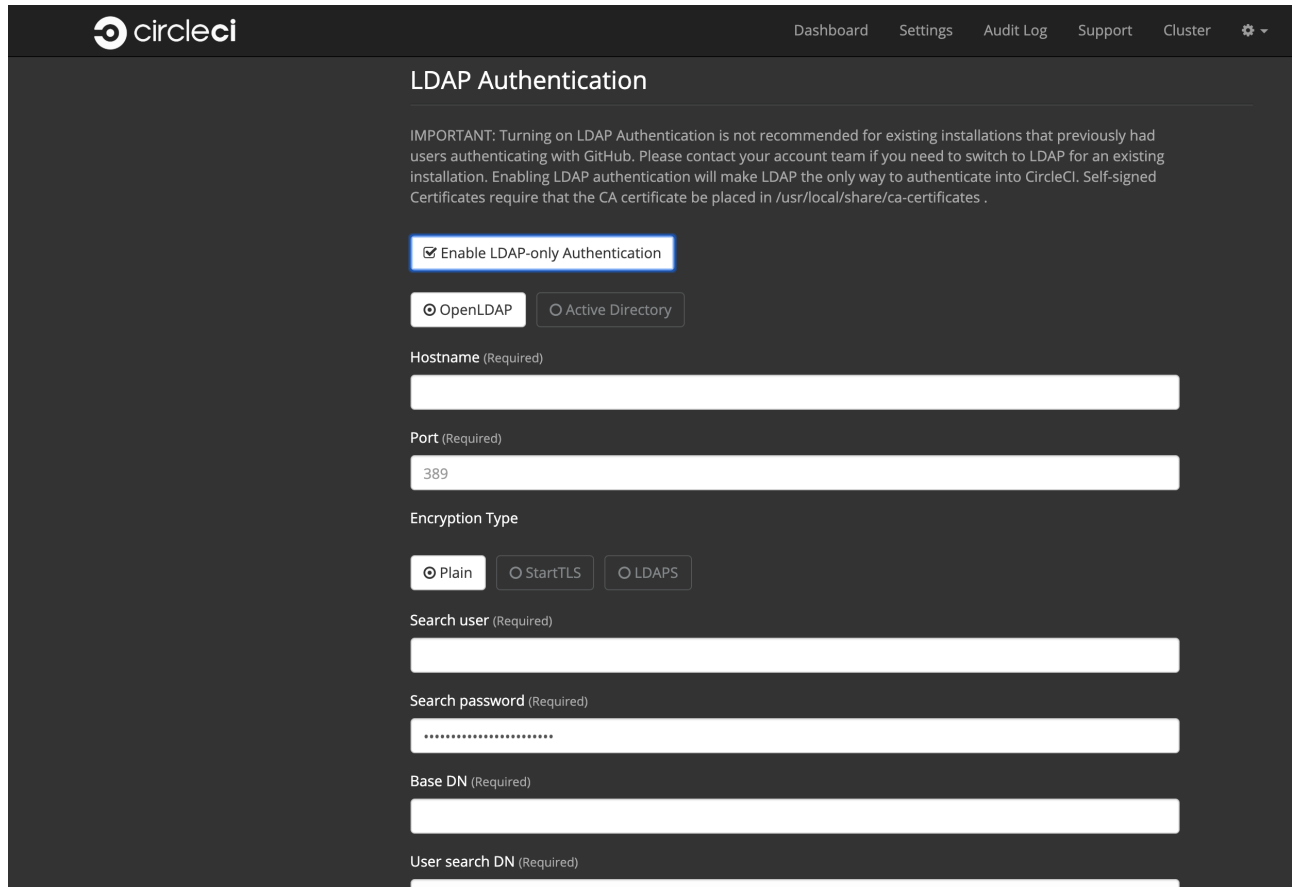


After completing this configuration, all users must log in to CircleCI with their LDAP credentials. After logging in to CircleCI, each user will then click the Connect button on the Accounts page to connect and authenticate their GitHub account.

# Configure LDAP Authentication

This section provides the steps to configure LDAP in the 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 2.0 instance.
3. Navigate to the Settings page and check the Enable LDAP-only Authentication button. Select either OpenLDAP or Active Directory.



The screenshot shows the 'LDAP Authentication' configuration page in the CircleCI management console. The page has a dark theme with a navigation bar at the top containing 'Dashboard', 'Settings', 'Audit Log', 'Support', and 'Cluster'. The main content area is titled 'LDAP Authentication' and includes an important warning: '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.' Below the warning, there is a checkbox labeled 'Enable LDAP-only Authentication' which is checked. Underneath, there are two radio buttons: 'OpenLDAP' (selected) and 'Active Directory'. The form then contains several required fields: 'Hostname (Required)', 'Port (Required)' (with '389' entered), 'Encryption Type' (with 'Plain' selected and 'StartTLS' and 'LDAPS' as options), 'Search user (Required)', 'Search password (Required)' (masked with dots), 'Base DN (Required)', and 'User search DN (Required)'.

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 appropriate value using the format `ou=<users>` replacing `users` with the value used in your LDAP instance.
9. Fill in the Username field with an appropriate 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 appropriate 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.



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> = VM Service :page-layout: classic-docs :icons: font  
:toc: macro :toc-title:
```

This section outlines how to set up VM service for your CircleCI installation for **machine** executor and remote Docker jobs, as well as how to customize your own VM service images.



This configuration is only available for installations on AWS, please contact your CircleCI account representative to request this configuration 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](#).

## Configuration

## 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 available 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**

**Machine Executor**



To configure VM service, it is best practice to select the AWS EC2 option in the Management Console Settings, 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 CircleCI 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. This 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:

1. Clone our image builder repo: <https://github.com/circleci/image-builder/tree/picard-vm-image>
2. Open **aws-vm.json** in your editor and fill in the required groups. An access key and secret key are required to upload. Handle the key and secret process according to your requirements, but consider restricting the **ami\_groups** to only within your organization
3. Run **packer build aws-vm.json**

Refer to [https://packer.io/docs/builders/amazon-ebs.html#ami\\_groups](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 [this](#) example.

The following sections summarize the key files and variables that impact CircleCI Server behavior.

### Notable Files & Folders

Need	Path	More info
General Config	<b>/etc/circle-installation-customizations</b>	See table below for values
JVM Heap Sizes	<b>/etc/circleconfig/XXXX/customizations</b> Supports: frontend, test_results	Adjust heap size for individual containers with <b>JVM_HEAP_SIZE</b>
Custom CA Certs	<b>/usr/local/share/ca-certificates/</b>	
Container Customizations	<b>/etc/circleconfig/XXX/customizations</b>	Used lots of places in replicated

Need	Path	More info
<code>/etc/hosts</code>	<code>/etc/hosts</code>	Respected by several containers including frontend, copied to container's <code>/etc/hosts</code>
<code>/etc/environment</code>	<code>/etc/environment</code>	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 <b>all</b> 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	

## Job and Instance Management

Jobs run using the remote Docker environment, or the **machine** executor are scheduled and dispatched by Nomad, even though the jobs are not run on Nomad clients. 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 don't end up in a dead/bad state. = Setting Up Certificates :page-layout: classic-docs :icons: font :toc: macro :toc-title:

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



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

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 (<http://<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
```

1. You will be instructed to add a DNS TXT record.
2. 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 (Recommended)"
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 (<http://<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/Github Enterprise matches the domain name pointing to the Services machine, including the protocol used, for example **https://info-tech.io/**.