OPERATIONS GUIDE

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

Docs Team

Version 2.17.2, July 2nd, 2019

Table of Contents

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.

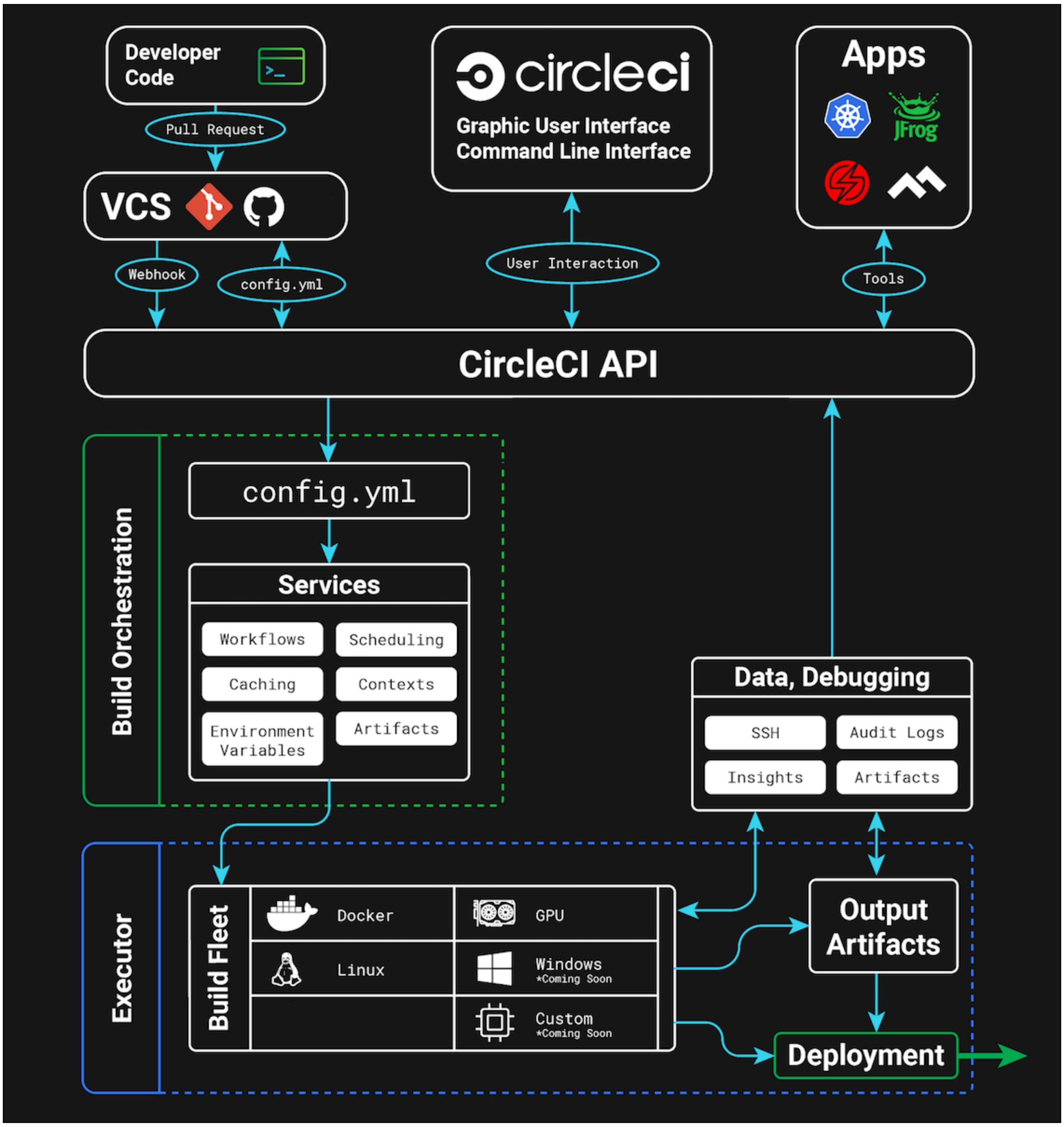


Figure 1. CircleCI Services Architecture

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.

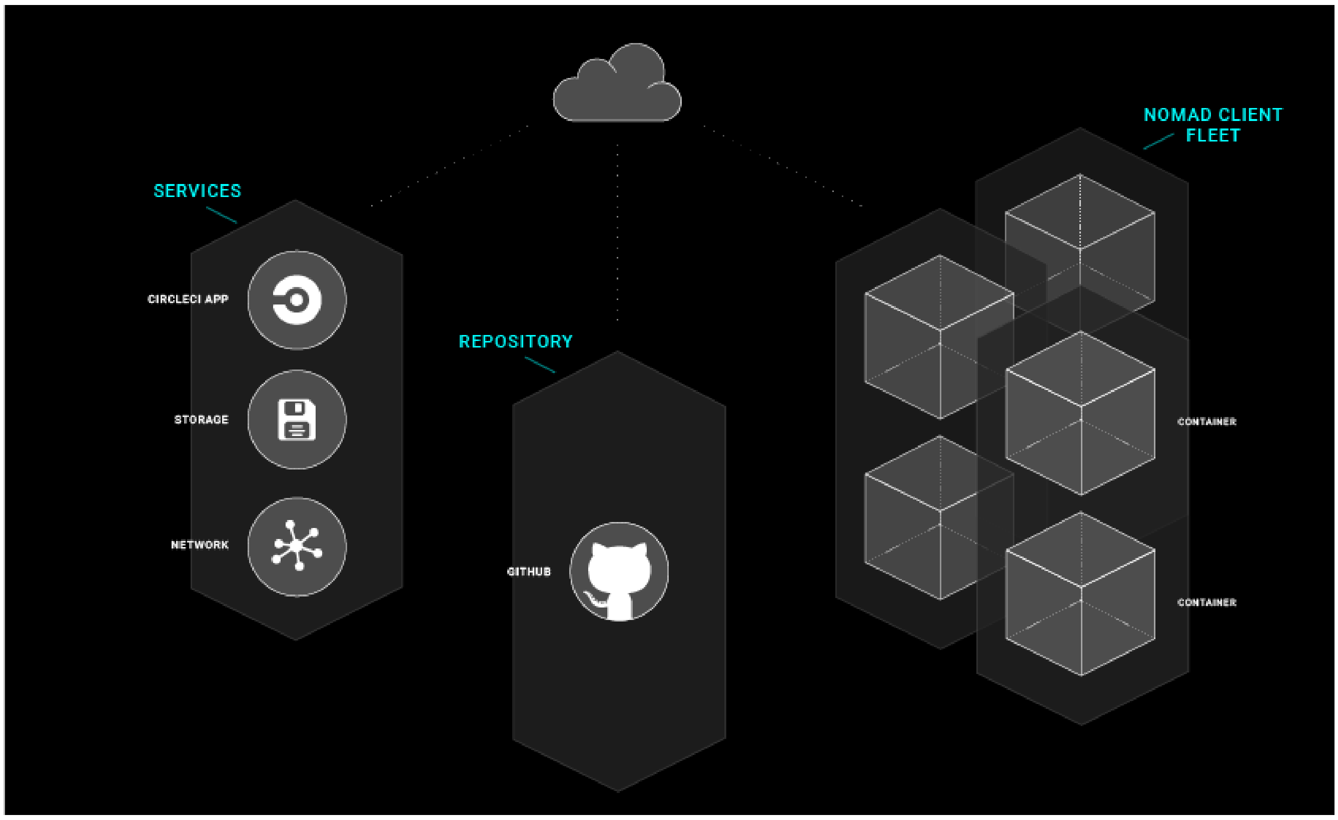
Architecture

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 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.

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.

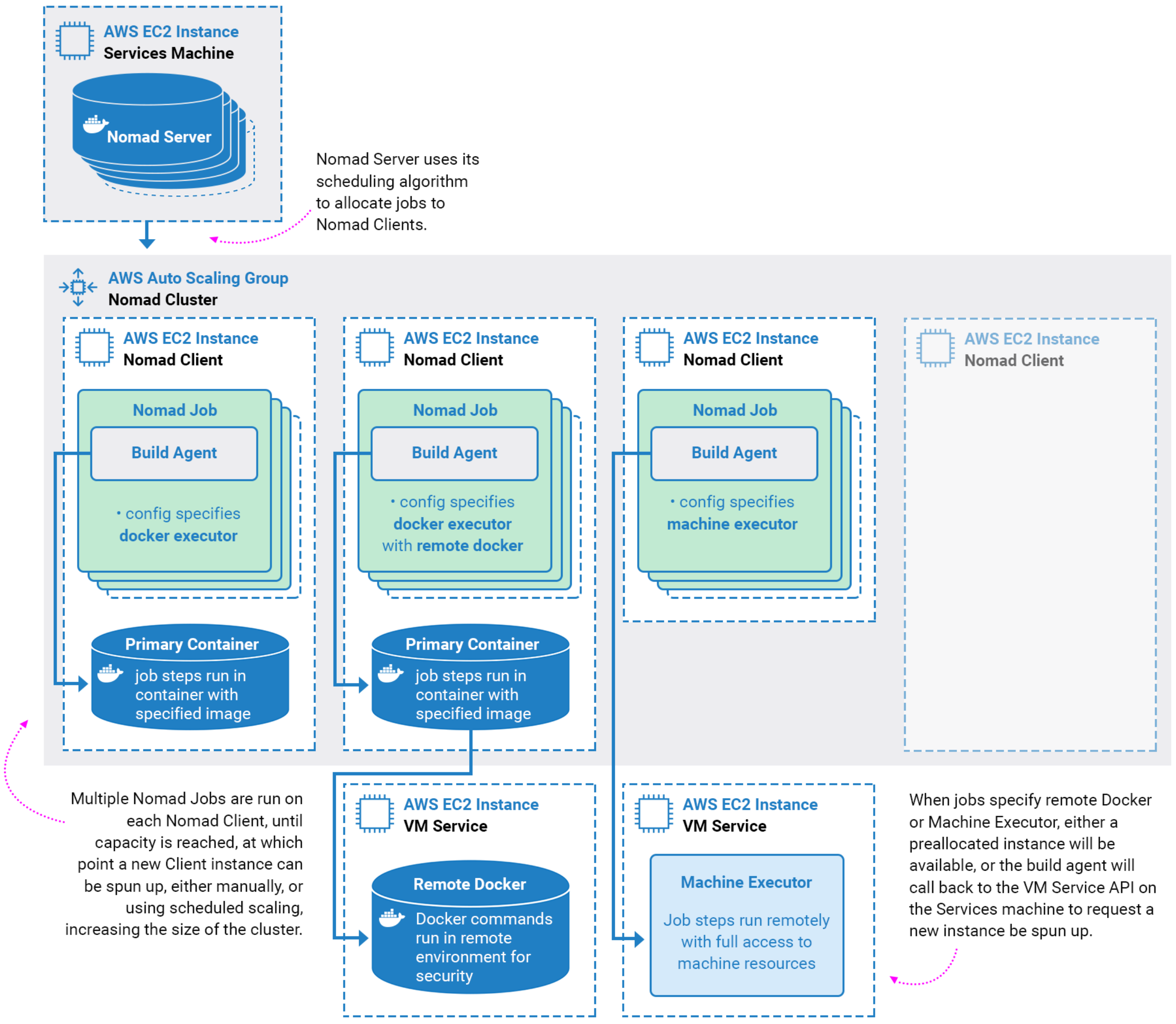

AWS EC2 Instance
Services Machine
Nomad Server
Nomad Server uses its scheduling algorithm to allocate jobs to Nomad Clients.
AWS Auto Scaling Group
Nomad Cluster
AWS EC2 Instance
Nomad Client
Nomad Job
Build Agent
• config specifies docker executor
Primary Container
job steps run in container with specified image
AWS EC2 Instance
Nomad Client
Nomad Job
Build Agent
• config specifies docker executor with remote docker
Primary Container
job steps run in container with specified image
AWS EC2 Instance
Nomad Client
Nomad Job
Build Agent
• config specifies machine executor
AWS EC2 Instance
Nomad Client
Multiple Nomad Jobs are run on each Nomad Client, until capacity is reached, at which point a new Client instance can be spun up, either manually, or using scheduled scaling, increasing the size of the cluster.
AWS EC2 Instance
VM Service
Remote Docker
Docker commands run in remote environment for security
AWS EC2 Instance
VM Service
Machine Executor
Job steps run remotely with full access to machine resources
When jobs specify remote Docker or Machine Executor, either a preallocated instance will be available, or the build agent will call back to the VM Service API on the Services machine to request a new instance be spun up.

Figure 2. Nomad Cluster Management

Basic Terminology and Architecture

- **Nomad Server:** Nomad servers are the brains of the cluster; they receive and allocate jobs to Nomad clients. In CircleCI, a Nomad server runs on your Services machine as a Docker Container.
- **Nomad Client:** Nomad clients execute the jobs they are allocated by Nomad servers. Usually a Nomad client runs on a dedicated machine (often a VM) in order to fully take the advantage of machine power. You can have multiple Nomad clients to form a cluster and the Nomad server allocates jobs to the cluster with its scheduling algorithm.
- **Nomad Jobs:** A Nomad job is a specification, provided by a user, that declares a workload for Nomad. In CircleCI 2.0, a Nomad job corresponds to an execution of a CircleCI job. If the job uses parallelism, say 10 parallelism, then Nomad will run 10 jobs.
- **Build Agent:** Build Agent is a Go program written by CircleCI that executes steps in a job and reports the results. Build Agent is executed as the main process inside a Nomad Job.

Basic Operations

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

The `nomad` CLI is installed in the Service instance. It is pre-configured to talk to the Nomad cluster, so it is possible to use the `nomad` command to run the following commands in this section.

Checking the Jobs Status

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

```
nomad status
```

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

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

Checking the Cluster Status

To get a list of your Nomad clients, run:

```
nomad node-status
```

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

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

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

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

Checking Logs

As noted in the Nomad Jobs section above, a Nomad Job corresponds to an execution of a CircleCI job. Therefore, 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

This section includes information for System Administrators on the following:

- Scaling your cluster
- Gathering metrics for monitoring your CircleCI installation

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
`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 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 }
   ```

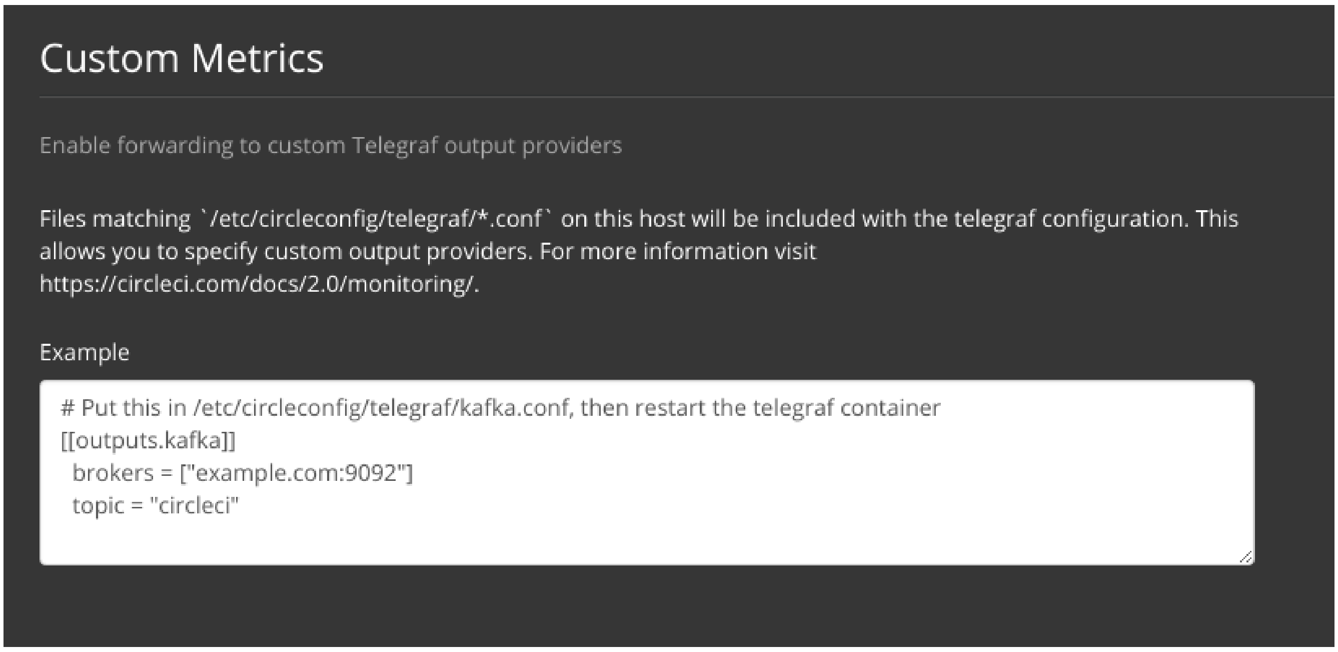
3. Enter your DataDog API Key

   ```
   ![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. Telegraph 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 Servics 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 \hyperref[sec:shut-down-nomad]{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/) wherein CircleCI engineering spent time running simulations of cost savings for the purpose of developing a general set of best practices for Auto Scaling. Consider the following best practices when setting up AWS Auto Scaling:

1. In general, size your cluster large enough to avoid queueing builds. That is, less than one second of queuing for most workloads and less than 10 seconds for workloads run on expensive hardware or at highest parallellism. Sizing to reduce queuing to zero is best practice because of the high cost of developer time. It is difficult to create a model in which developer time is cheap enough for under-provisioning to be cost-effective.
2. Create an Auto Scaling Group with a Step Scaling policy that scales up during the normal working hours of the majority of developers and scales back down at night. Scaling up during the weekday normal working hours and back down at night is the best practice to keep queue times down during peak development, without over provisioning at night when traffic is low. Looking at millions of builds over time, a bell curve during normal working hour emerges for most data sets.

This is in contrast to auto scaling throughout the day based on traffic fluctuations, because modelling revealed that boot times are actually too long to prevent queuing in real time. Use [Amazon’s Step Policy](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

This document 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](https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-proxy.html#cli-configure-proxy-ec2)

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.

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 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](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 installing CircleCI Server on AWS using Terraform, you should add the below to your Nomad client launch configuration <!--how?? oh the next sentence answers how?-� – These instructions should be added to `/etc/environment`.
- If you are using Docker <!--is this an install method we no longer support?-�, refer to the [Docker HTTP Proxy Instructions](https://docs.docker.com/engine/admin/systemd/#/http-proxy) documentation.
- 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](https://docs.docker.com/network/proxy/) for making sure that 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

Refer to the "Adding External Services to CircleCI Server v2.17" document for instructions to configure your installation for data persistence. <!--but this is an internal doc… should this say 'contact support for guidance on configuring for data persistance?-