INSTALLATION GUIDE

A guide for installing or migrating to CircleCI Server v3.1.0 on AWS or GCP

docs@circleci.com

Version , 07/06/2021: FINAL

CircleCI Server v3.x Prerequisites

It is assumed you have already read the server 3.x overview.

In order to configure the CircleCI server application, you will need to ensure the following general and infrastructure-specific requirements are met. You will need:

- An existing Kubernetes cluster (see our guide if you need help creating one), for example:
  - Creating an Amazon EKS cluster - Amazon EKS
    - Using `eksctl` is our recommended option, as it creates a VPC and selects the proper security group for you.
  - Creating clusters - Google GKE
    - Do NOT use an Autopilot cluster. CircleCI requires functionality that is not supported by GKE Autopilot.
- Note your Kubernetes cluster must meet the following minimum overall cluster requirements relative to the number of active CircleCI server users:

Number of daily active CircleCI users	Minimum Nodes	Total CPU	Total RAM	NIC speed
< 500	3	12 cores	32 GB	1 Gbps
500+	3	48 cores	240 GB	10 Gbps

- Your cluster must have outbound access to pull Docker containers and verify your license. If you do not want to provide open outbound access, see our list of ports that will need access.
- You must have appropriate permissions to list, create, edit and delete pods in your cluster. You can verify that you can list these resources by running: `kubectl auth can-i <list|create|edit|delete> pods`.
- A CircleCI License file. Contact CircleCI support for a license.
- The Required Tools tools installed
  - Note that the `kubectl` tool must also be configured to have access to your cluster.
    - Connect to Amazon EKS clusters - Amazon EKS
    - Configuring cluster access for kubectl - Google GKE
- Port access requirements are listed here:
  - Kubernetes Load Balancers
  - Kubernetes Nodes
  - Nomad Clients
  - External VMs

There are no requirements regarding VPC setup or disk size for your cluster. It is, however, recommended that you set up a new VPC rather than use an existing one.

Required Tools

Tool	Version	Used for
Terraform	0.15.4 or greater	Infrastructure Management
kubectl	1.19 or greater	Kubernetes CLI
Helm	3.4.0 or greater	Kubernetes Package Management
Kots	1.44.1 or greater	Replicated Kubernetes Application Management

External Ports

Port number	Protocol	Direction	Source / Destination	Use	Notes
80	TCP	Inbound	End users	HTTP web app traffic	
443	TCP	Inbound	End users	HTTP web app traffic	
8800	TCP	Inbound	Administrators	Admin console	
22	TCP	Inbound	Administrators	SSH	Only required for the bastion host
64535-65535	TCP	Inbound		SSH into builds	Only required for the nomad clients.

What to read next

- Creating your first cluster
- Server 3.x Installation

CircleCI Server v3.x - Creating Your First Cluster (Optional)

If you have never previously set up a Kubernetes cluster, we have provided some tips in this section.

Step 1 - Create

Amazon EKS

CircleCI recommends using `eksctl` to set up your first cluster on AWS. `eksctl` will take care of VPC creation, in addition to security group selection.

Before creating the cluster, make sure you have the following:

1. The latest AWS CLI installed and configured to your AWS account
2. `eksctl` installed
3. `kubectl` installed

Create a cluster using flags

To create a simple cluster, you can run the command:

```
eksctl create cluster
```

Additional flags are available on the command line. See the `eksctl` introduction for more information.

Create cluster using a configuration file

You can also create a config file for your cluster, for example:

```
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: <your-cluster-name>
  region: <aws-region>

managedNodeGroups:
  - name: <nodegroup-name-1>
    instanceType: <instance-type> # i.e., m5.large. see https://aws.amazon.com/ec2/instance-types/
for available instance types
    minSize: 4 # see https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-properties-
as-group.html#cfn-as-group-minsize for more information
    maxSize: 6 # see https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-properties-
as-group.html#cfn-as-group-maxsize for more information
```

For more examples on cluster configuration files, see eksctl.io.

When you are finished with your configuration, save it and run:

```
eksctl create cluster -f <your-cluster-config.yaml>
```

When using the `eksctl` tool to create your cluster, you may receive an AWS STS access error: `AWS STS access – cannot get role ARN for current session: InvalidClientTokenId.`

This may mean your AWS credentials are invalid, or your IAM user does not have permission to create an EKS cluster. Note that the proper IAM permissions are necessary in order to use `eksctl`. See the AWS documentation regarding prerequisite IAM permissions.

Step 2 - Verify

Once your cluster is finished being created, you should be able to run various `kubectl` commands to view your cluster resources.

For instance, to view your cluster's built-in services, you can run:

```
kubectl cluster-info
```

Or, to verify that your cluster has worker nodes attached, run:

```
kubectl get nodes -o wide
```

For more information on exploring your new cluster, see the following:

- Step 2: View resources - AWS docs
- Accessing clusters - Kubernetes docs
- Cluster Management - `kubectl` docs

What to read next

- Server 3.x Installation
- Hardening Your Cluster

CircleCI Server v3.x Installation

Before you begin with the CircleCI server v3.x installation process, ensure all prerequisites are met.

Step 1 - Install KOTS

CircleCI server v3.x uses KOTS from Replicated to manage and distribute server v3.x. KOTS is a `kubectl` plugin. To install the latest version, you can run `curl https://kots.io/install | bash`.

Ensure you are running the minimum KOTS (KOTS 1.44.1) by running `kubectl kots version`.

The KOTS command will open up a tunnel to the admin console. If running on Windows inside WSL2, the port is not available on the host machine. Turning WSL off and back on should resolve the issue. For more information, please see https://github.com/microsoft/WSL/issues/4199.

Next, run:

```
kubectl kots install circleci-server
```

You will be prompted for a:

- namespace for the deployment
- password for the KOTS admin console

When complete, you should be provided with a URL to access the admin console. Usually this is http://localhost:8800.

If you need to get back to the management console at a later date you can run:

```
kubectl kots admin-console -n <namespace kots was installed in>
```

Step 2 - Configure Server (Part 1)

Once you have accessed the URL to the administrative console and uploaded your license file, you will see similar to the following, where you will need to configure CircleCI server.

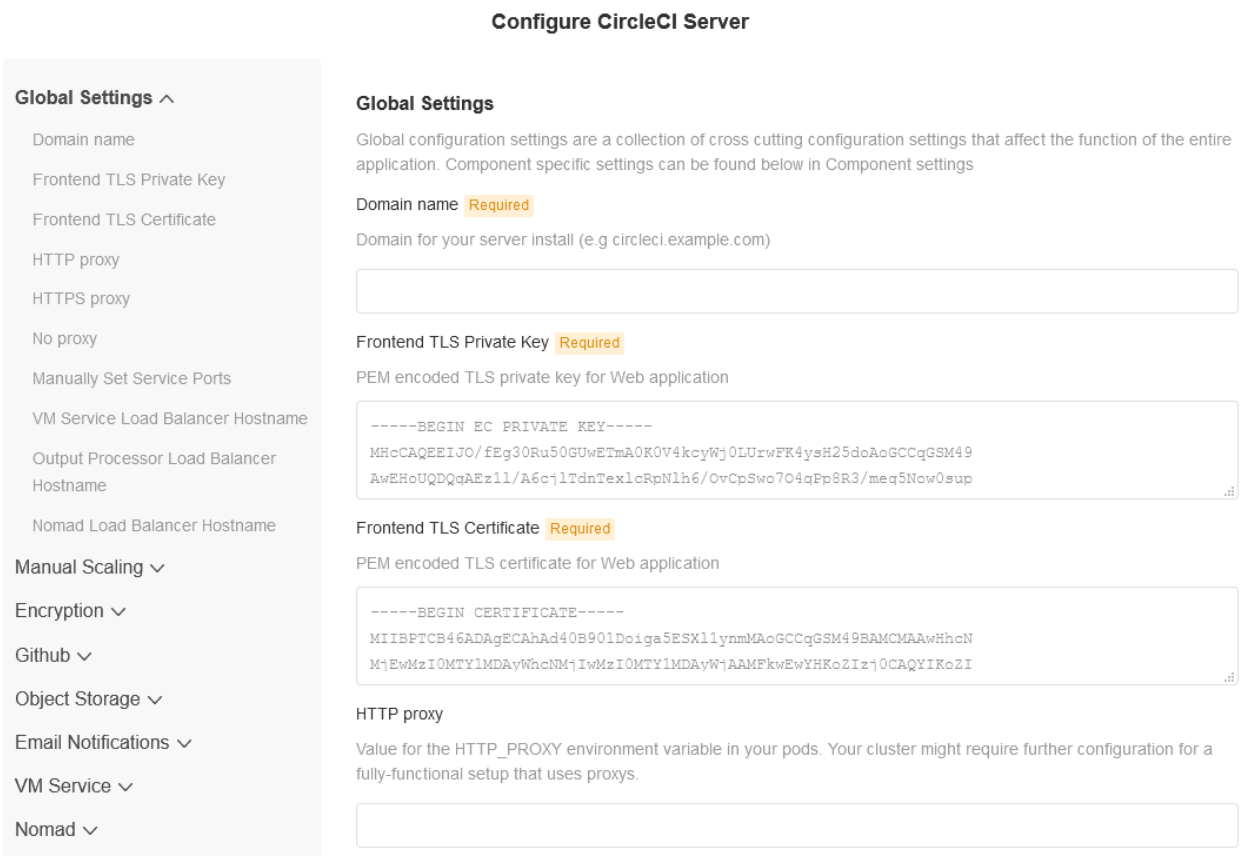

Figure 1. Server v3.1.0 Configuration

Global Settings

Global configuration settings are a collection of cross-cutting configuration settings that affect the function of the entire application as a whole. Component-specific settings can be found below under the **Component Settings** section.

1. **Domain Name** (required): The domain name for your server installation. For example, circleci.yourcompany.com. You will need to configure this domain and point it to the Circleci server Traefik Load Balancer once it is configured in a later step.
2. **Frontend TLS Private Key**: A PEM encoded TLS private key for the web application. A default, self-signed TLS key is provided.
3. **Frontend TLS Certificate**: A PEM encoded TLS certificate for the web application. A default, self-signed TLS certificate is provided.

If you don't already have a TLS certificate for your installation, you can generate one using a tool of your choice. See the Optional: Generate a TLS Certificate section.

4. **Private Load Balancers**: By default, frontend load balancers will be assigned a public IP address. Ticking this box will make the load balancers for the frontend private, resulting in no public IPs being assigned to them. If you change this setting after the initial deployment, you might need to re-deploy the traefik service for it to take effect.

this only works for GKE and EKS installations.

5. **Manually Set Service Ports**: Select to edit the default ports configured for Nomad, Output-Processor or VM Service. Ticking this box will reveal a ports field for each service.
6. **VM Service Load Balancer Hostname** (required): The hostname or IP of your VM Service Load Balancer. If you do not yet have this value, you can deploy the application with any value and then retrieve and update it in Step 3 - Obtain Load Balancer IPs.
7. **Output Processor Load Balancer Hostname** (required): The hostname or IP of your Output Processor load balancer. If you do not yet have this value, you can deploy the application with any value and then retrieve and update it in Step 3 - Obtain Load Balancer IPs.
8. **Nomad Load Balancer Hostname** (required): The hostname or IP of your Nomad load balancer. If you do not yet have this value, you can deploy the application with any value and then retrieve and update it in Step 3 - Obtain Load Balancer IPs.

Optional: Generate a TLS Certificate

By default, CircleCI server will create self-signed certificates to get you started. In production, you should supply a certificate from a trusted certificate authority. The LetsEncrypt certificate authority, for example, can issue a certificate for free using their certbot tool.

For example, if you host your DNS on Google Cloud, the following commands will provision a certification for your installation:

```
DOMAIN=example.com  # replace with the domain of your installation of server
GOOGLE_APPLICATION_CREDENTIALS=/path/to/credentials.json  # Path to GCP credentials
sudo certbot certonly --dns-google \
  --dns-google-credentials ${GOOGLE_APPLICATION_CREDENTIALS} \
  -d "${DOMAIN}" \
  -d "app.${DOMAIN}"
```

If instead you're using AWS Route53 for DNS, execute this example:

```
DOMAIN=example.com # replace with the domain of your installation of server
sudo certbot certonly --dns-route53 \
  -d "${DOMAIN}" \
  -d "app.${DOMAIN}"
```

This will create a private key and certificate (including intermediate certificates) in `/etc/letsencrypt/live/${DOMAIN}/`.

It is important that your certificate contain both your domain and the `app.*` subdomain as subjects. For example, if you host your installation at `server.example.com`, your certificate must cover `app.server.example.com` and `server.example.com`.

PotsgreSQL

Internal: Deploys a pre-configured PostgreSQL 12 instance that runs inside of the CircleCI namespace.

External : Selecting the external option allows operators to connect to a PostgreSQL install running outside of the cluster. CirceCI is tested against PostgreSQL 12.6. We highly suggest that you configure your external PostgreSQL instance prior to deploying the CircleCI application. You can find more information about configuration here. Once you have completed configuring your instance, fill out the following section:

- PostgreSQL Service Domain
  - The domain or IP address of your PostgreSQL instance.
- PostgreSQL Service Port
  - The port of your PostgreSQL instance.
- PostgreSQL Service User
  - A user with the appropriate privileges to access your PostgreSQL instance.
- PostgreSQL Service Password
  - The password of the user used to access your PostgreSQL instance.

MongoDB

Internal: deploys a completely configured MongoDB instance along with your server installation. **External**: allows you to use your own MongoDB instance. CircleCI server is tested to work with MongoDB 3.6. You can customize the setup for your external MongoDB instance with the following settings:

1. MongoDB connection host(s) or IP(s): The hostname or IP of your MongoDB instance. Specifying a port using a colon and multiple hosts for sharded instances are both supported.
2. Use SSL for connection to MongoDB: Whether to use SSL when connecting to your external MongoDB instance
3. Allow insecure TLS connections: If you use a self-signed certificate or one signed by a custom CA, you will need to enable this setting. However, this is an insecure setting and you should use a TLS certificate signed by a valid CA if you can.
4. MongoDB user: The user name for the account to use. This account should have the dbAdmin role.
5. MongoDB password: The password for the account to use.
6. MongoDB authentication source database: The database that holds the account information, usually `admin`.
7. MongoDB authentication mechanism: The authentication mechanism to use, usually `SCRAM-SHA-1`.
8. Additional connection options: Any other connection options you would like to use. This needs to be formatted as a query string (key=value pairs, separated by &, special characters need to be URL encoded). See the MongoDB docs for available options.

Encryption

These keysets are used to encrypt and sign artifacts generated by CircleCI.

1. **Artifact Signing Key** (required): To generate, run:

```
docker run circleci/server-keysets:latest generate signing -a stdout
```

Copy and paste the entirety of the output into the Artifact Signing Key field.

2. **Encryption Signing Key** (required) : To generate, run:

```
docker run circleci/server-keysets:latest generate encryption -a stdout
```

Copy and paste the entirety of the output into the Encryption Signing Key field.

It is recommended to store these securely; if they are lost, job history and artifacts will not be recoverable.

GitHub

These settings control authorization to server using Github OAuth and allow for updates to Github using build status information.

If this instance is being set up in preparation for a migration from 2.19, it is recommended to use a new OAuth application, not the same one used in 2.19.

1. **Github Type**: Select Cloud or Enterprise.
2. **OAuth Client ID** (required): In GitHub, navigate to **Settings** > **Developer Settings** > **OAuth Apps** and select the **Register a new application** button.

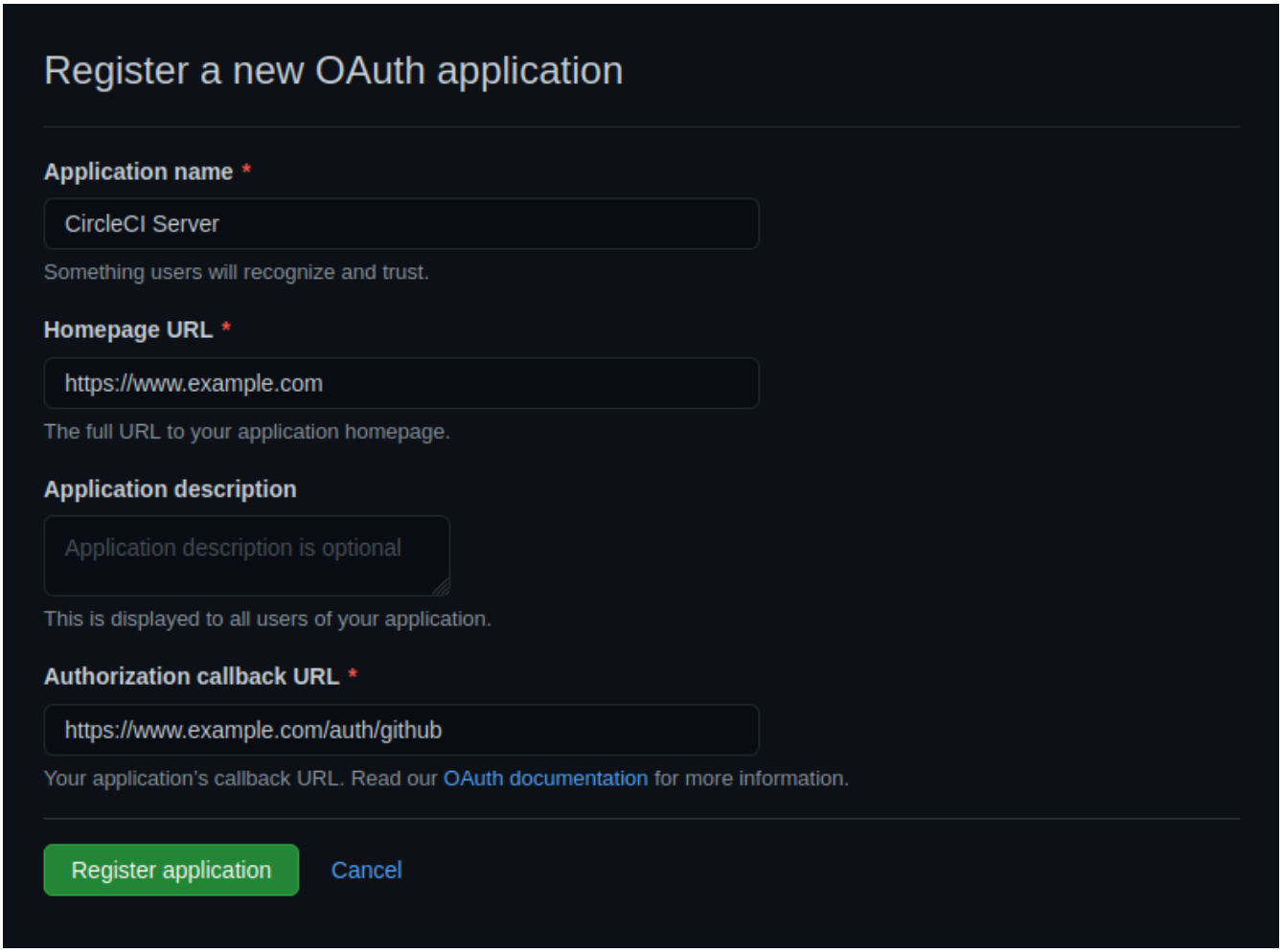

Figure 2. Register a new OAuth application

The domain you selected for your CircleCI installation is the Homepage URL and **<your-circle-ci-domain>/auth/github** is the Authorization callback URL.

3. **OAuth Client Secret** (required): On your Oauth application, you can create one by selecting the **Generate a new client secret** button in GitHub.

If using GitHub Enterprise, you will also need a personal access token and the domain name of your GitHub Enterprise instance. You must also enable HTTP API Rate Limiting from the management console.

MongoDB

Internal will deploy a completely configured MongoDB instance along with your server installation. **External** allows you to use your own MongoDB instance. CircleCI server is tested to work with MongoDB 3.6. You can customize the setup for your external MongoDB instance with the following settings:

1. MongoDB connection host(s) or IP(s): The hostname or IP of your MongoDB instance. Specifying a port using a colon and multiple hosts for sharded instances are both supported.
2. Use SSL for connection to MongoDB: Whether to use SSL when connecting to your external MongoDB instance
3. Allow insecure TLS connections: If you use a self-signed certificate or one signed by a custom CA, you

will need to enable this setting. However, this is an insecure setting and you should use a TLS certificate signed by a valid CA if you can.

4. MongoDB user: The user name for the account to use. This account should have the dbAdmin role.
5. MongoDB password: The password for the account to use.
6. MongoDB authentication source database: The database that holds the account information, usually `admin`.
7. MongoDB authentication mechanism: The authentication mechanism to use, usually `SCRAM-SHA-1`.
8. Additional connection options: Any other connection options you would like to use. This needs to be formatted as a query string (key=value pairs, separated by &, special characters need to be URL encoded). See the MongoDB docs for available options.

Vault

Internal will deploy the default Vault instance inside the CircleCI K8s namespace. The application will be automatically configured. **External** will not install the default Vault instance with the CircleCI application. Select this option if using an existing Vault instance. You need to configure the following settings:

1. URL: ex. `http://vault:8200`
2. Transit Path: The path of the transit secrets engine. The default is `transit`. See the Vault documentation for more details.
3. Token: The Vault token to be used by CircleCI.

Object Storage

Server 3.x hosts build artifacts, test results, and other state in object storage. We support

1. AWS S3
2. Minio
3. Google Cloud Storage

While any S3-compatible object storage may work, we test and support AWS S3 and Minio. For object storage providers that do not support the S3 API, such as Azure blob storage, we recommend using Minio Gateway.

Please choose the one that best suits your needs. A **Storage Bucket Name** is required, in addition to the following fields, depending on if you are using AWS or GCP. Ensure that the bucket name you provide exists in your chosen object storage provider before proceeding.

S3-compatible Object Storage

To configure S3-compatible storage, set the following details in the object storage section of the configuration page:

1. **Storage Bucket Name** (required): The bucket used for server.
2. **Storage Object Expiry** (optional): Number of days to retain your test results and artifacts. Set to 0 to disable and retain objects indefinitely.

3. **AWS S3 Region** (optional): AWS region of bucket if your provider is AWS. S3 Endpoint is ignored if this option is set.
4. **S3 Endpoint** (optional): API endpoint of S3 storage provider. Required if your provider is not AWS. AWS S3 Region is ignored if this option is set.
5. **Access Key ID** (required): Access Key ID for S3 bucket access.
6. **Secret Key** (required): Secret Key for S3 bucket access.

It is recommended to create a new user with programmatic access for this purpose. If your provider support IAM policies, you should fill in `<BUCKET_NAME>` and attach the following policy to the user:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:*"
      ],
      "Resource": [
        "arn:aws:s3:::<BUCKET_NAME>",
        "arn:aws:s3:::<BUCKET_NAME>/*"
      ]
    }
  ]
}
```

Google Cloud Storage

To configure Google Cloud Storage (GCS), set the following details in the object storage section of the configuration page:

1. **Storage Bucket Name** (required): The bucket used for server.
2. **Storage Object Expiry** (optional): Number of days to retain your test results and artifacts. Set to 0 to disable and retain objects indefinitely.
3. **Service Account JSON** (required): A JSON format key of the Service Account to use for bucket access.

A dedicated service account is recommended. Add to it the `Storage Object Admin` role, with a condition on the resource name limiting access to only the bucket specified above. For example, enter the following into the Google's Condition Editor of the IAM console:

```
resource.name.startsWith("projects/_/buckets/<bucket-name>")
```

Use `startsWith` and prefix the bucket name with `projects/_/buckets/`.

Email Notifications

Build notifications are sent via email.

1. **Email Submission server hostname**: Host name of the submission server (e.g., for Sendgrid use smtp.sendgrid.net).
2. **Username**: Username to authenticate to submission server. This is commonly the same as the user's e-mail address.
3. **Password**: Password to authenticate to submission server.
4. **Port**: Port of the submission server. This is usually either 25 or 587. While port 465 is also commonly used for email submission, it is often used with implicit TLS instead of StartTLS. Server only supports StartTLS for encrypted submission.

Outbound connections on port 25 are blocked on most cloud providers. Should you select this port, be aware that your notifications may fail to send.

5. **Enable StartTLS**: Enabling this will encrypt mail submission.

You should only disable this if you can otherwise guarantee the confidentiality of traffic.

VM Service

VM Service configures VM and remote docker jobs. You can configure a number of options for VM service, such as scaling rules.

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

Authentication and Permissions

AWS EC2

You will need the following fields to configure your VM Service to work with AWS EC2. Note that the Access Key and Secret Key used by VM Service differs from the policy used by Object Storage in the previous section. VM Service and Object Storage are kept distinct to allow organizations to utilize different cloud and on-premise providers within the same installation.

1. **AWS Region** (required): This is the region the application is in.
2. **AWS Windows AMI ID** (optional): If you require Windows builders, you can supply an AMI ID for them here.
3. **Subnet ID** (required): Choose a subnet (public or private) where the VMs should be deployed.
4. **Security Group ID** (required): This is the security group that will be attached to the VMs. It must be created manually.

The recommended security group configuration can be found in the Hardening Your Cluster section.

Additionally, the below commands can be run to create the necessary security groups in AWS or GCP.

AWS

```
$ aws ec2 create-security-group \
    --description "CircleCI's VM Service security group" \
    --group-name "circleci-vm-service-sg"
$ aws ec2 authorize-security-group-ingress \
    --group-name "circleci-vm-service-sg" \
    --protocol tcp \
    --port 22 \
    --cidr "<<CIDR of Nomad clients>>"
$ aws ec2 authorize-security-group-ingress \
    --group-name "circleci-vm-service-sg" \
    --protocol tcp \
    --port 22 \
    --cidr "<<CIDR of Kubernetes nodes>>"
$ aws ec2 authorize-security-group-ingress \
    --group-name "circleci-vm-service-sg" \
    --protocol tcp \
    --port 2376 \
    --cidr "<<CIDR of Nomad clients>>"
$ aws ec2 authorize-security-group-ingress \
    --group-name "circleci-vm-service-sg" \
    --protocol tcp \
    --port 2376 \
    --cidr "<<CIDR of Kubernetes nodes>>"
$ aws ec2 authorize-security-group-ingress \
    --group-name "circleci-vm-service-sg" \
    --protocol tcp \
    --port 54782
```

GCP

```
$ gcloud compute firewall-rules create "circleci-vm-service-internal-nomad-fw" \
    --network "<<network for CircleCI; optional if default>>" \
    --action allow \
    --source-ranges "<<CIDR of Nomad clients>>" \
    --rules "TCP:22,TCP:2376"
$ gcloud compute firewall-rules create "circleci-vm-service-internal-k8s-fw" \
    --network "<<network for CircleCI; optional if default>>" \
    --action allow \
    --source-ranges "<<CIDR of Kubernetes nodes>>" \
    --rules "TCP:22,TCP:2376"
$ gcloud compute firewall-rules create "circleci-vm-service-external-fw" \
    --network "<<network for CircleCI; optional if default>>" \
    --action allow \
    --rules "TCP:54782"
```

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

It is recommended to create a new user with programmatic access for this purpose. You should fill in [Security Group ID] and [VPC ARN] and attach the following IAM policy to the user:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": "ec2:RunInstances",
      "Effect": "Allow",
      "Resource": [
        "arn:aws:ec2:*::image/*",
        "arn:aws:ec2:*::snapshot/*",
        "arn:aws:ec2:*:*:key-pair/*",
        "arn:aws:ec2:*:*:launch-template/*",
        "arn:aws:ec2:*:*:network-interface/*",
        "arn:aws:ec2:*:*:placement-group/*",
        "arn:aws:ec2:*:*:volume/*",
        "arn:aws:ec2:*:*:subnet/*",
        "arn:aws:ec2:*:*:security-group/<<Security Group ID>>"
      ]
    },
    {
      "Action": "ec2:RunInstances",
```

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

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

```
            "ec2:Vpc": "<<VPC ARN>>"
          }
        }
      }
    ]
  }
```

Google Cloud Platform

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

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

We recommend you create a unique service account used exclusively by VM Service. The Compute Instance Admin (Beta) role is broad enough to allow VM Service to operate. If you wish to make permissions more granular, you can use the Compute Instance Admin (beta) role documentation as reference.

Configuring VM Service

1. **Number of <VM type> VMs to keep prescaled**: By default, this field is set to 0 which will create and provision instances of a resource type on demand. You have the option of preallocating up to 5 instances per resource type. Preallocating instances lowers the start time allowing for faster machine and `remote_docker` builds. Note, that preallocated instances are always running and could potentially increase costs. Decreasing this number may also take up to 24 hours for changes to take effect. You have the option of terminating those instances manually, if required.
2. **VM Service Custom Configuration**: Custom configuration can fine tune many aspects of your VM service. This is an advanced option and we recommend you reach out to your account manager to learn more.

Nomad

You will configure aspects of your Nomad control plane in Step 3 after completing the Nomad setup in Step 2. This section can be left with its default values until Step 3, with the exception of mTLS, which should be only be enabled after completing Step 4.

Enable Mutual TLS (mTLS)

mTLS encrypts and authenticates traffic between your Nomad control plane and Nomad clients. You should

disable mTLS until you have completed Step 4 - Install Nomad Clients and can obtain the certificate, private key and certificate authority output after completing Step 4.

When all required information has been provided, click the **Continue** button and your CircleCI installation will be put through a set of preflight checks to verify your cluster meets the minimum requirements and attempt to deploy. When completed successfully, you should see something like the following and you can continue to the next step:

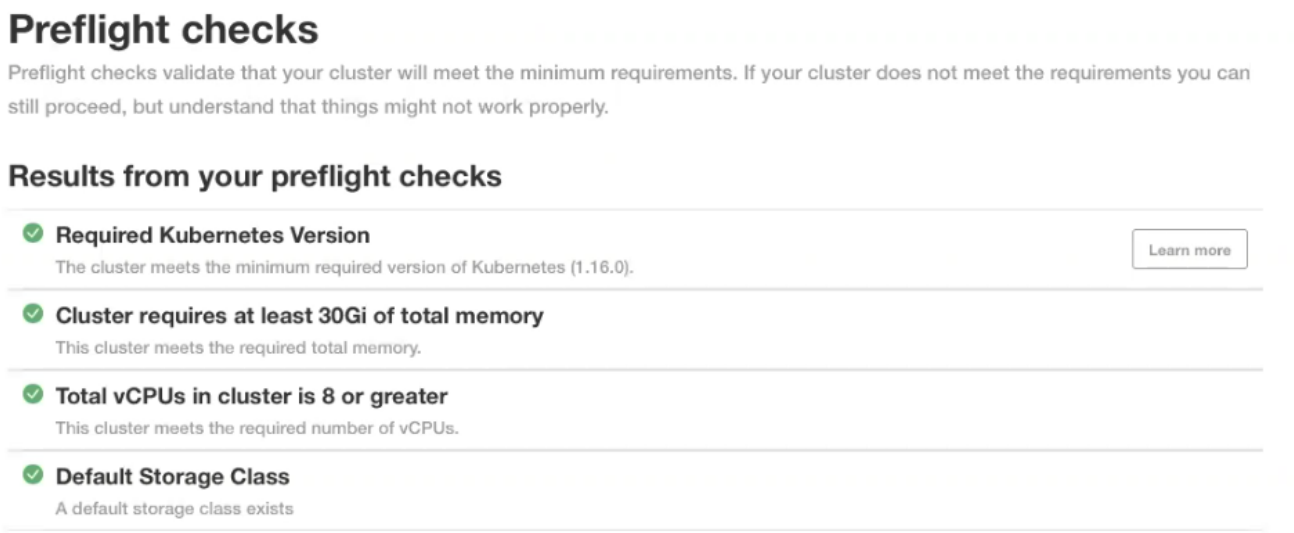

Figure 3. Sever v3.1.0 Preflight Checks

Step 3 - Obtain Load Balancer IPs

Run `kubectl get services` and note the following load balancer addresses. You will need these to finish configuring your installation. If necessary, specify the namespace, `kubectl get services -n <the-namespace-you-installed-circleci>` to get the list of services.

- Circleci server Traefik Proxy
- VM Service Load Balancer URI
- Output Processor Load Balancer URI
- Nomad server Load Balancer URI

Depending on your cloud environment and configuration, your output can contain either an external IP address or a hostname for your load balancers. Either will work.

The values for VM Service, Output Processor and Nomad server should be added into the config as described in Step 2 - Configure Server (Part 1). The value from Circleci server Traefik should be used in Step 5 - Create DNS Entries for the Frontend to create the DNS entry for your applications domain name and sub-domain.

If you had to leave the default value in place for the Nomad `server_endpoint` in the previous step, you can now go back to the terraform repository, fill in the correct value in `terraform.tfvars` and run `terraform apply` again.

At this time you can choose to create DNS entries for each of the load balancers. It is not required, but some users prefer to do so. For example, VM service might be called `vmservice.circleci.yourdomain.com`.

Step 4 - Install Nomad Clients

As mentioned in the Overview, Nomad is a workload orchestration tool that CircleCI uses to schedule (via Nomad server) and run (via Nomad Clients) CircleCI jobs.

Nomad client machines are provisioned outside the cluster and need access to the Nomad Control Plane, Output Processor, and VM Service.

CircleCI curates Terraform modules to help install Nomad clients in your cloud provider of choice. You can browse the modules in our public repository.

AWS

If you would like to install Nomad clients in AWS, create a file `main.tf` file with the following contents:

```
# main.tf
terraform {
  required_version = ">= 0.15.4"
  required_providers {
    aws = {
      source = "hashicorp/aws"
      version = ">=3.0.0"
    }
  }
}
provider "aws" {
# Your region of choice here
region = "us-west-1"
}

module "nomad_clients" {
source = "git::https://github.com/CircleCI-Public/server-terraform.git//nomad-aws?ref=3.1.0"

  # Number of nomad clients to run
  nodes = 4
  subnet = "<< ID of subnet you want to run nomad clients in >>"
  vpc_id = "<< ID of VPC you want to run nomad client in >>"

  server_endpoint = "<< hostname:port of nomad server >>"

  dns_server = "<< ip address of your VPC DNS server >>"
  blocked_cidrs = [
    "<< cidr blocks you'd like to block access to e.g 10.0.1.0/24 >>"
  ]
}

output "nomad_server_cert" {
value = module.nomad_clients.nomad_server_cert
}

output "nomad_server_key" {
value = module.nomad_clients.nomad_server_key
}

output "nomad_ca" {
value = module.nomad_clients.nomad_tls_ca
}
```

To deploy your Nomad clients simply run:

```
terraform init
terraform plan
terraform apply
```

After Terraform is done spinning up the Nomad client(s), it will output the certificates and key needed for Nomad mTLS encryption mentioned in the Nomad configuration section. Make sure to copy them somewhere safe.

Once terraform apply is complete, click on the **Application** tab in the admin console and wait for the deployment Status to show "Ready," then move on to the next step.

Google Cloud Platform

If you'd like to to install Nomad clients in Google Cloud Platform, create a file `main.tf`. An example is provided below to document common settings. For documentation on all available variables please see the module README.

```
# main.tf
provider "google-beta" {
  # Your specific credentials here
  project = "your-project"
  region  = "us-west1"
  zone    = "us-west1-a"
}

module "nomad_clients" {
  # Note the use of ref=<<tag>> to pin the version to a fixed release
  source = "git::https://github.com/CircleCI-Public/server-terraform.git//nomad-gcp?ref=3.1.0"

  zone    = "us-west1-a"
  region  = "us-west1"
  network = "my-network"
  # Only specify a subnet if you use custom subnetworks in your VPC. Otherwise delete the next line.
  subnet  = "my-nomad-subnet"

  # hostname:port of Nomad load balancer, leave port as 4647 unless you know what you are doing
  server_endpoint = "nomad.example.com:4647"

  # Number of nomad clients to run
  min_replicas     = 3
  max_replicas     = 10

  # Example autoscaling policy: scale up if we ever reach 70% cpu utilization
  autoscaling_mode = "ONLY_UP"
  target_cpu_utilization = 0.70

  # Network policy example: block access to 1.1.1.1 from jobs and allow retry
  # with SSH from only 2.2.2.2
  blocked_cidrs = [
    "1.1.1.1/32"
  ]
  retry_with_ssh_allowed_cidr_blocks = [
    "2.2.2.2/32"
  ]
}

output "nomad_server_cert" {
  value = module.nomad_clients.nomad_server_cert
}

output "nomad_server_key" {
  value = module.nomad_clients.nomad_server_key
}

output "nomad_ca" {
  value = module.nomad_clients.nomad_tls_ca
}
```

To deploy your Nomad clients simply run:

```
terraform init
terraform plan
terraform apply
```

After Terraform is done spinning up the Nomad client(s), it will output the certificates and key needed for Nomad mTLS encryption mentioned in the Nomad configuration section. Make sure to copy them somewhere safe.

Once Terraform `apply` is complete, click on the **Application** tab in the admin console and wait for the deployment Status to show "Ready," then move on to the next step.

Optional: Running Jobs Outside the Nomad Client

CircleCI server can run Docker jobs on Nomad clients, but it can also run jobs in a dedicated VM. These VM jobs are controlled by Nomad clients, therefore the Nomad clients must be able to access the VM machines on port 22 for SSH and port 2376 for remote Docker jobs.

The machines for VM jobs are addressed via their external IPs in GCP at the moment. You need to create appropriate ingress rules for TCP port 2376 with the IP addresses of the Nomad clients and Kubernetes nodes as allowed sources.

Step 5 - Create DNS Entries for the Frontend

Next, create a DNS entry for your Traefik load balancer, i.e. `circleci.your.domain.com` and `app.circleci.your.domain.com`. You will recall that in Step 2 - Configure Server (Part 1) we detailed how to create TLS certs for your server install. Although TLS is optional, if it be used, it is important to ensure your TLS certificate covers both the server domain and sub-domain as in the examples provided. Once the user is logged in, all client requests are routed through your Traefik sub-domain, i.e, `app.{your_domain}.com`.

For more information on adding a new DNS record, see the following documentation:

- Managing Records (GCP)
- Creating records by using the Amazon Route 53 Console (AWS)

Step 6 - Configure Server (Part 2) and Deploy

Go back to the **Config** tab in the admin console.

Run `kubectl kots admin-console -n <namespace kots was installed in>` if you need to get back to the admin console.

Global Settings

Enter the values obtained from Step 3 - Obtain Load Balancer IPs into VM Service Load Balancer Hostname, Output Processor Load Balancer Hostname, and Nomad Load Balancer Hostname under Global Settings.

Nomad

mTLS encrypts and authenticates traffic between your Nomad control plane and Nomad clients. If you have already deployed the Nomad clients via terraform in Step 4 - Install Nomad Clients you can and should enable mutual TLS (mTLS).

This should only be disabled if you can guarantee the authenticity of the nodes joining your cluster and confidentiality of traffic from them to the control plane in some other way.

1. **Nomad Server Certificate** (required if mTLS is enabled): Obtained in Step 4 - Install Nomad Clients.
2. **Nomad Server Private Key** (required if mTLS is enabled): Obtained in Step 4 - Install Nomad Clients.
3. **Nomad Server Certificate Authority (CA) Certificate** (required if mTLS is enabled): Obtained in Step 4 - Install Nomad Clients.

Deploy

Click the **Save config** button to update your installation and re-deploy server.

Step 7 - Validate Installation

1. Launch your CircleCI installation in your browser, for example https://hostname.com.
    - . If you are using a self-signed TLS cert, you will see a security warning at this stage. You will need to use proper TLS certs if you want to avoid this.
2. Sign up/Log in into your CircleCI installation. As the first user to log in, you will become the administrator at this point.
3. Take a look at our Getting Started guide to start adding projects.
4. Use the CircleCI realitycheck repository and follow the README to check basic CircleCI functionality.

If you are unable to run your first builds successfully, start with the Troubleshooting guide for general troubleshooting topics, and the Introduction to Nomad Cluster Operation for information about how to check the status of Nomad Clients within your installation.

What to read next

- Hardening Your Cluster
- Server 3.x Migration

CircleCI Server v3.x Migration

Migrating from 2.19.x to 3.x requires you to back up your 2.19 instance data (Mongo, Postgres, and Vault) and then restore that data in a waiting Server 3.x instance. If you run into trouble you can fallback to your 2.19 instance. Migration does require an already operating Server 3.x installation. Depending on the size of your data stores, the migration can take anywhere from a few minutes to a few hours. We recommend using a staging environment before completing this process in a production environment. This will not only allow you to gain a better understanding of the migration process, but will also give you a feel for how long the migration will take to complete.

Prerequisites

1. Your current CircleCI Server installation is 2.19.
2. You have taken a backup of the 2.19 instance. If you are using external datastores, they will need to be backed up separately.
3. You have a new CircleCI Server 3.x installation.
4. You have successfully run reality check with contexts prior to starting.
5. The migration script must be run from a machine with:
    - `kubectl` configured for the server 3.x instance
    - `ssh` access to the 2.19 services box

External Datastores Only

1. Backups have been taken of all external data stores.
2. Postgres has been updated to version 12.

Internal Datastore Only

1. You have taken a backup of the 2.19 instance.
2. You have successfully run reality check on the new server 3.x instance with contexts prior to starting.

Migration

Migrating to server v3.x will shut down your v2.19 application. Your v2.19 application will not be started back up, although you may manually start it back up using the administrative console.

Starting the migration process will cause downtime. It is recommended you schedule a maintenance window.

Running server 2.19 and server 3.x at the same time can cause issues to your 2.19 build data. Server 2.19 should NOT be restarted if server 3.x is running.

Step 1 - Clone the repository and run the migration script

The instructions below will clone the repository containing the server v2.19.x to server v3.x migration script. The migration script will:

- Stop your v2.19.x application
- Perform pre-flight checks to confirm namespace and datastores for 2.19.x.
- Create a tarball of your v2.19.x application's PostgreSQL and Mongo databases.
- Archive existing application data for Vault and CircleCI encryption/signing keys.
- Export the 2.19.x tarball to your v3.x installation. Exported data stores are stored in a directory named `circleci_export`, located relative to wherever the migration script is run from. This can be useful for debugging purposes.
- Perform pre-flight checks to confirm namespace and datastores for 3.x instance.
- Scale v3.x application deployments down to zero.
- Import the data from the previously exported tarball to your new v3.x instance.
- Scale v3.x application deployments up to one.

If you have externalized services then you can run `bash migrate.sh -v -p -m`. These `-v -p -m` flags will skip the migration of Vault, Postgres, and Mongo, respectively. Skipping all three will copy your keys from `/data/circle/circleci-encryption-keys` on the v2.19.x services machine, allowing you to `cat` these files and upload their contents to the 3.x configuration page.

In a terminal:

1. Run `git clone https://github.com/CircleCI-Public/server-scripts`.
2. Change into the `migrate` directory: `cd server-scripts/migrate`.
3. Run the migration script: `./migrate.sh`.
4. You will be prompted for the following information:
    - Username of your server 2.19 installation
    - Hostname of your server 2.19 installation
    - The path to your SSH key file for your server 2.19.x installation
    - Kubernetes namespace of your server 3.x installation
5. After the script has completed, the Signing and Encryption keys from the 2.19 instance will need to be added to the new 3.0 instance via the KOTS Admin Console. The keys will be located in `circleci_export/circle-data`.
6. The 3.x instance will either need to be updated to point at the same storage bucket that the 2.19 instance used, or the data needs to be copied over to a new bucket. The latter will ensure the 2.19 instance continues to work as expected, and so is the recommended approach if this migration is part of a test.

If a different hostname is being used in the 3.x environment, the GitHub webhooks will still be pointing to the hostname used in the 2.19 environment. The easiest way to update this is to click **Stop Building** and then **Set Up Project**. After doing this, the contexts and environment variables associated with the project will still be present.

Step 2 - Validate your migration to Server 3.0

Re-run reality check with contexts on your new server 3.x environment by pushing a fresh commit.

Step 3 - Update your team

Once you have successfully run reality check, notify your team of the new CircleCI UI and URL, if it has changed.

Frequently Asked Questions

Where did all my job and build history go?

- All of your existing jobs and build history have been moved to the Legacy Jobs view. You can view the complete job history using one of the following methods:
  - Selecting Projects → PROJECT_NAME and selecting the `legacy jobs view` link at the bottom of the project's build history
  - Using the following URL pattern: `https://<APP_DOMAIN>/pipelines/github/<ORG>/<PROJECT>/jobs`
  - For a specific job, append a job number to the URL: <code><a href="https://<APP_DOMAIN>/pipelines/github/<ORG>/<PROJECT>/jobs/<JOB#&gt" class="bare">https://<APP_DOMAIN>/pipelines/github/<ORG>/<PROJECT>/jobs/<JOB#&gt;</code>

Why does nothing happen when I select "Start Building" on my project after migration?

- By default, a newly added project (a project that has never been followed) will trigger a build automatically after it has been followed for the first time. If the project was or ever has been followed in 2.0 or 3.0, it will not be considered a new project or first build and a build will not be triggered after follow. To trigger a build, perform an activity that will trigger a Github webhook such as pushing up a new commit or branch.

I got an error "Error from server (NotFound):"

- The script assumes specific naming patterns for your Postgres and MongoDB. If you get this error, it may indicate a non-standard installation, a missing DB migration, or other issues. In this case it is best to contact support with a support bundle and the output from the migration script.

What to read next

- Hardening Your Cluster
- Server 3.x Operator Guide

CircleCI Server v3.x Hardening Your Cluster

This section provides supplemental information on hardening your Kubernetes cluster.

Network Topology

A server installation basically runs three different type of compute instances: The Kubernetes nodes, Nomad clients and external VMs.

It is highly recommended that you deploy these into separate subnets with distinct CIDR blocks. This will make it easier for you to control traffic between the different components of the system and isolate them from each other.

As always, the rule is to make as many of the resources as private as possible, applies. If your users will access your CircleCI server installation via VPN, there is no need to assign any public IP addresses at all, as long as you have a working NAT gateway setup. Otherwise, you will need at least one public subnet for the CircleCI server Traefik load balancer.

However, in this case, it is also recommended to place Nomad clients and VMs in a public subnet to enable your users to SSH into jobs and scope access via networking rules.

Currently, custom subnetting is not supported for GCP. Custom subnetting support will be available in a future update/release.

Network Traffic

This section spells out the minimum requirements that are needed for a server installation to work. Depending on your workloads, you might need to add additional rules to egress for Nomad clients and VMs. Nomenclature between cloud providers differs, therefore, you will probably need to implement these rules using firewall rules and/or security groups.

Where you see "external," this usually means all external IPv4 addresses. Depending on your particular setup, you might be able to be more specific (e.g., if you are using a proxy for all external traffic).

It is assumed that you have configured the load balancers for Nomad, vm-service and output processor to be internal load balancers - this is the default.

The rules spelled out here are assumed to be stateful and for TCP connections only, unless stated otherwise. If you are working with stateless rules, you will need to create matching ingress or egress rules to the ones listed here.

Kubernetes Load Balancers

Depending on your setup, your load balancers might be transparent (i.e. they are not treated as a distinct layer in your networking topology). In this case, you can apply the rules from this section directly to the underlying destination or source of the network traffic. Refer to the documentation of your cloud provider to make sure you understand how to correctly apply networking security rules given the type of load balancing you are using with your installation.

Ingress

If the traffic rules for your load balancers have not been created automatically, here are their respective ports:

Name	Port	Source	Purpose
*-server-traefik	80	External	User Interface & Frontend API
*-server-traefik	443	External	User Interface & Frontend API
vm-service	3000	Nomad clients	Communication with Nomad clients
nomad	4647	Nomad clients	Communication with Nomad clients
output-processor	8585	Nomad clients	Communication with Nomad clients

Egress

The only type of egress needed is TCP traffic to the K8s nodes on the K8s load balancer ports (30000-32767). This is not needed if your load balancers are transparent.

Common Rules for Compute Instances

These rules apply to all compute instances, but not to the load balancers.

Ingress

If you want to access your instances via SSH, you will need to open port 22 for TCP connections for the instances in question. It is recommended to scope the rule as closely as possible to allowed source IPs and/or only add such a rule ad hoc when needed.

Egress

You most likely want all of your instances to access internet resources. This will require you to allow egress for UDP and TCP on port 53 to the DNS server within your VPC, as well as TCP ports 80 and 443 for HTTP and HTTPS traffic, respectively. Instances building jobs (i.e. the Nomad clients and external VMs) also will likely need to pull code from your VCS via SSH (TCP port 22). SSH is also used to communicate with external VMs, so it should be allowed for all instances with the destination of the VM subnet and your VCS at the very least.

Kubernetes Nodes

Intra-node traffic

The traffic within your K8s cluster is regulated by networking policies by default. For most purposes, this should be sufficient to regulate the traffic between pods and there is no additional requirement to pare

down traffic between K8s nodes any further (i.e. it is fine to allow all traffic between K8s nodes).

To make use of networking policies within your cluster, you may need to take additional steps, depending on your cloud provider and setup. Here are some resources to get you started:

- Kuberenetes Network Policy Overview
- Creating a Cluster Network Policy on Google Cloud
- Installing Calico on Amazon EKS

Ingress

If you are using a managed service, you can check the rules created for the traffic coming from the load balancers and the allowed port range. The standard port range for K8s load balancers (30000-32767) should be all that is needed here for ingress. If you are using transparent load balancers, you will need to apply the ingress rules listed for load balancers above.

Egress

Port	Destination	Purpose
2376	VMs	Communication with VMs
4647	Nomad clients	Communication with the Nomad clients
all traffic	other nodes	Allow intra-cluster traffic

Nomad Clients

Nomad clients do not need to communicate with each other; you can block traffic between Nomad client instances completely.

Ingress

Port	Source	Purpose
4647	K8s nodes	Communication with Nomad server
64535-65535	External	Rerun jobs with SSH functionality

Egress

Port	Destination	Purpose
2376	VMs	Communication with VMs
3000	VM Service load balancers	Internal communication
4647	Nomad Load Balancer	Internal communication
8585	Output Processor Load Balancer	Internal communication

External VMs

Similar to Nomad clients, there is no need for external VMs to communicate with each other.

Ingress

Port	Source	Purpose
22	Kubernetes nodes	Internal communication
22	Nomad clients	Internal communication
2376	Kubernetes nodes	Internal communication
2376	Nomad clients	Internal communication
54782	External	Rerun jobs with SSH functionality

Egress

You will only need the egress rules for internet access and SSH for your VCS.

What to read next

- Server 3.x Migration
- Server 3.x Operations