INSTALLATION GUIDE

A guide for installing or migrating to CircleCI Server v3.4.8 on Google Cloud Platform

docs@circleci.com

Version 3.4.8, 02/02/2024: FINAL

CircleCI Server v3.x Installation Phase 1

Phase 1: Prerequisites

CircleCI server v3.x is installed in 4 phases. There is a validation step at the end of each phase, allowing you to confirm success before moving to the next phase. Depending on your requirements, phases 3 and 4 may include multiple steps. This installation guide assumes you have already read the CircleCI Server v3.x Overview.

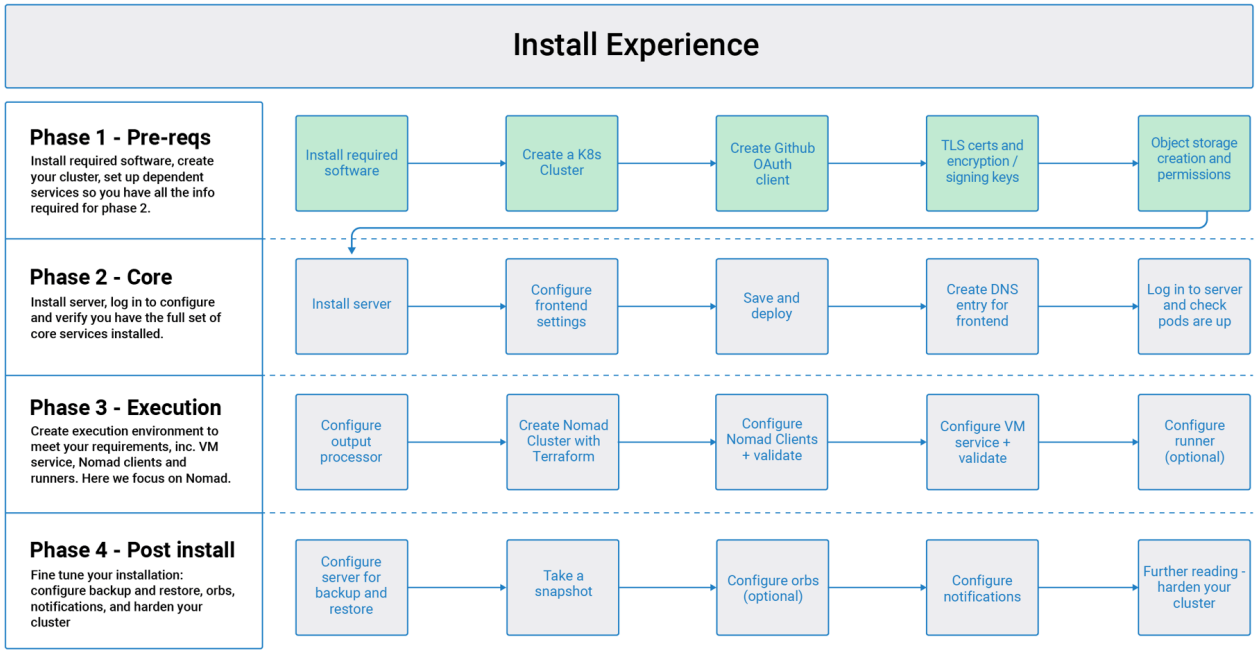


Figure 1. Installation Experience Flow Chart Phase 1

In the following sections, replace any items or credentials displayed between < > with your details.

Install required software

Download and install the following software before continuing:

Tool	Version	Used for	Notes
Terraform	0.15.4 or greater	Infrastructure Management	
kubectl	1.19 or greater	Kubernetes CLI	
Helm	3.9.2 or greater	Kubernetes Package Management	

Tool	Version	Used for	Notes
KOTS: Mac or Linux.	1.65.0 *	Replicated Kubernetes Application Management. KOTS is a `kubectl` plugin.	Once you have extracted `kots` from the tar.gz (`tar zxvf kots_linux_amd64.tar.gz`), run `sudo mv kots /usr/local/bin/kubectl-kots` to install it. Mac users will need to grant a security exception.
Velero CLI	Latest	Backup and restore capability	See Velero's supported providers documentation for further information.

* Please take note of the supported KOTS versions for your Kubernetes cluster. KOTS version compatibility

GCP required software

- `gcloud` and `gsutil`. You can install and set-up these tools up by installing Google Cloud SDK. For further information refer to the Google Cloud SDK docs.

S3 compatible storage required software

- Install and configure MinIO CLI for your storage provider.

Create a Kubernetes cluster

CircleCI server installs into an existing Kubernetes cluster. The application uses a large number of resources. Depending on your usage, your Kubernetes cluster should meet the following requirements:

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

Supported Kubernetes versions:

CircleCI Version	Kubernetes Version
3.0.0 - 3.2.1	< 1.21
3.2.2 - 3.3.0	1.16 - 1.21
3.4.0 - 3.4.6	1.16 - 1.23

Creating a Kubernetes cluster is your responsibility. Please note:

- 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 need access.
- You must have appropriate permissions to list, create, edit, and delete pods in your cluster. Run this command to verify your permissions:

```
kubectl auth can-i <list|create|edit|delete> pods
```

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

GKE

You can learn more about creating a GKE cluster here.

Do not use Autopilot cluster. CircleCI requires functionality that is not supported by GKE Autopilot.

1. Install and configure the GCP CLI for your GCP account. This includes creating a Google Project, which will be required to create a cluster within your project.

When you create your project, make sure you also enable API access. If you do not enable API access, the command we will run next (to create your cluster) will fail.

2. Create your cluster by running the following command:

```
gcloud container clusters create circleci-server --project <YOUR_GOOGLE_CLOUD_PROJECT_ID> --region europe-west1
--num-nodes 3 --machine-type n1-standard-4
```

3. Configure `kubectl` with your your gcloud credentials:

```
gcloud container clusters get-credentials circleci-server --region europe-west1
```

4. Verify your cluster:

```
kubectl cluster-info
```

5. Create a service account for this cluster:

```
gcloud iam service-accounts create <YOUR_SERVICE_ACCOUNT_ID> --description="<YOUR_SERVICE_ACCOUNT_DESCRIPTION>"
--display-name="<YOUR_SERVICE_ACCOUNT_DISPLAY_NAME>"
```

6. Get the credentials for the service account:

```
gcloud iam service-accounts keys create <PATH_TO_STORE_CREDENTIALS> --iam-account
<SERVICE_ACCOUNT_ID>@<YOUR_GOOGLE_CLOUD_PROJECT_ID>.iam.gserviceaccount.com
```

Enable Workload Identities in GKE (optional)

Workload Identities for GKE allow workloads/pods in your GKE cluster to impersonate IAM service accounts to access Google Cloud services without using static service account credentials. In order to use Workload Identities you must enable them on your GKE cluster.

1. Enable Workload Identity on existing cluster

```
gcloud container clusters update "<CLUSTER_NAME>" \
  --region="<REGION>" \
  --workload-pool="<PROJECT_ID>.svc.id.goog"
```

2. Get node pools of existing GKE cluster

```
gcloud container node-pools list --cluster "<CLUSTER_NAME>" --region "<REGION>"
```

3. Update existing node pools

```
gcloud container node-pools update "<NODEPOOL_NAME>" \
  --cluster="<CLUSTER_NAME>" \
  --workload-metadata="GKE_METADATA" \
  --region="<REGION>"
```

You must repeat Step 3 for all the existing node pools. Follow these links for steps to enable Workload Identity for your Kubernetes service accounts: Nomad Autoscaler, VM and Object-Storage

Create a new GitHub OAuth app

If GitHub Enterprise and CircleCI server are not on the same domain, then images and icons from GHE will fail to load in the CircleCI web app.

Registering and setting up a new GitHub OAuth app for CircleCI server allows for authorization control to your server installation using GitHub OAuth and for updates to GitHub projects/repos using build status information.

1. In your browser, navigate to **your GitHub instance** > **Settings** > **Developer Settings** > **OAuth Apps** and click the **New OAuth App** button.

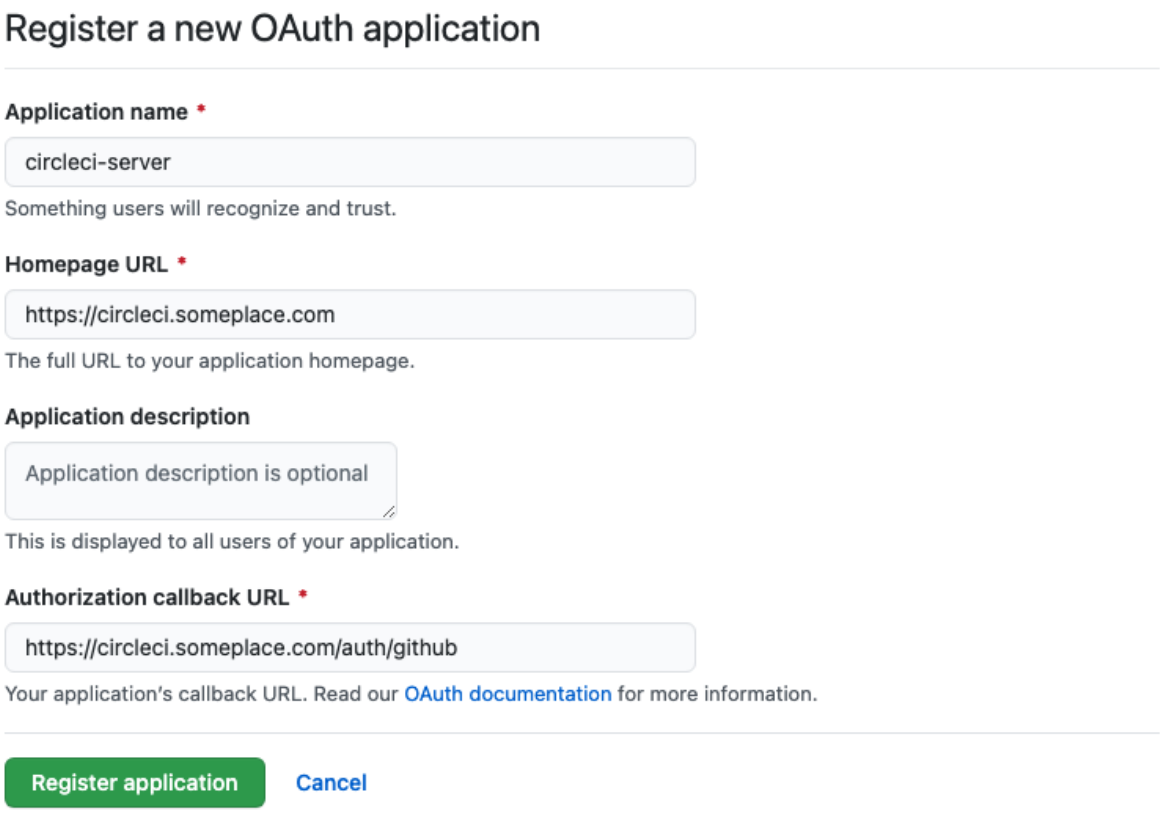


Figure 2. New GitHub OAuth App

2. Complete the following fields, based on your planned installation:
    - **Homepage URL**: The URL of your planned CircleCI installation.
    - **Authorization callback URL**: The authorization callback URL is the URL of your planned CircleCI installation followed by `/auth/github`
3. Once completed, you will be shown the **Client ID**. Select **Generate a new Client Secret** to generate a Client Secret for your new OAuth App. You need these values when you configure CircleCI server.

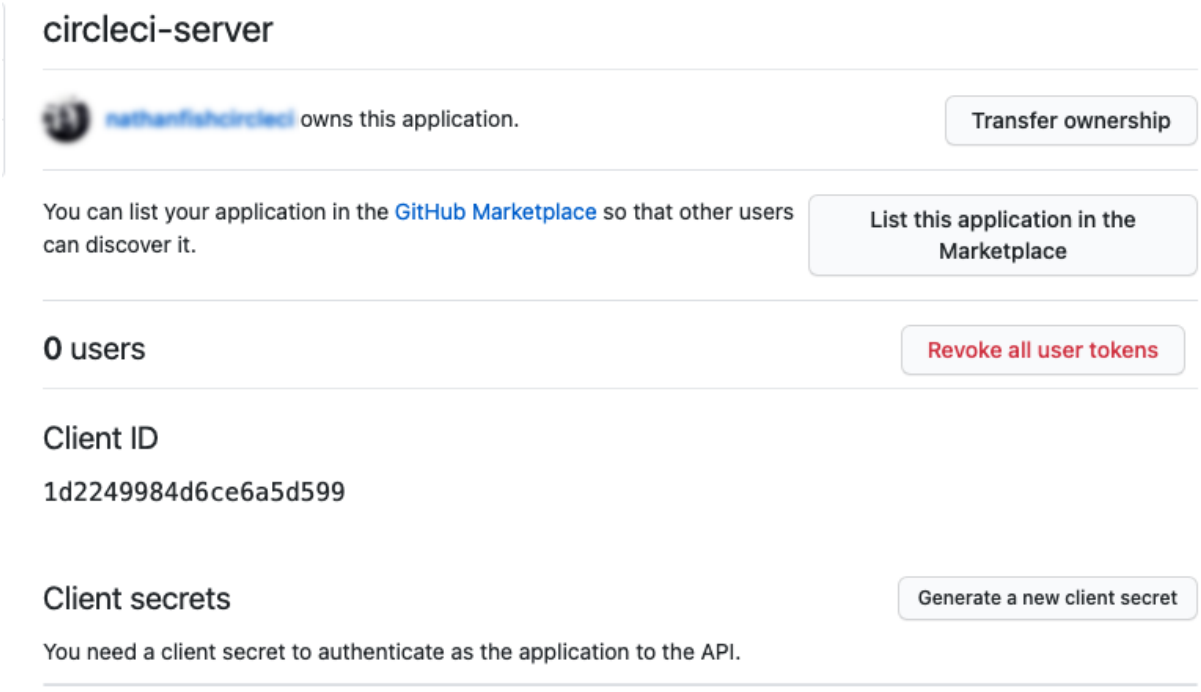


Figure 3. Client ID and Secret

If using GitHub Enterprise, you also need a personal access token and the domain name of your GitHub Enterprise instance.

Frontend TLS certificates

By default, CircleCI server creates self-signed certificates to get you started. In production, you should supply a certificate from a trusted certificate authority. The Let's Encrypt certificate authority, for example, can issue a free certificate using their certbot tool. The sections below cover using Google Cloud DNS and AWS Route 53.

Google Cloud DNS

1. If you host your DNS on Google Cloud, you need the **certbot-dns-google** plugin installed. You can install the plugin with the following command:

```
pip3 install certbot-dns-google
```

2. Then, the following commands will provision a certification for your installation:

```
certbot certonly --dns-google --dns-google-credentials <PATH_TO_CREDENTIALS> -d "<CIRCLECI_SERVER_DOMAIN>" -d
"app.<CIRCLECI_SERVER_DOMAIN>"
```

It is important that your certificate contains 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`.

You will need these certificates later, and they can be retrieved locally with the following commands:

```
ls -l /etc/letsencrypt/live/<CIRCLECI_SERVER_DOMAIN>
```

```
cat /etc/letsencrypt/live/<CIRCLECI_SERVER_DOMAIN>/fullchain.pem
```

```
cat /etc/letsencrypt/live/<CIRCLECI_SERVER_DOMAIN>/privkey.pem
```

Encryption/signing keys

These keysets are used to encrypt and sign artifacts generated by CircleCI. You need these values to configure server.

Store these values securely. If they are lost, job history and artifacts will not be recoverable.

Artifact signing key

To generate an artifact signing key, run the following command:

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

Encryption signing key

To generate an encryption signing key, run the following command:

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

Object storage and permissions

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

- AWS S3
- MinIO
- Google Cloud Storage

While any S3 compatible object storage may work, we test and support AWS S3 and MinIO.

Please choose the option that best suits your needs. A Storage Bucket Name is required, in addition to the fields listed below, depending on whether you are using AWS or GCP. Before proceeding, ensure the bucket name you provide exists in your chosen object storage provider.

If you are installing behind a proxy, object storage should be behind this proxy also. Otherwise proxy details will need to be supplied at the job level within every project `.circleci/config.yml` to allow artifacts, test results, cache save and restore, and workspaces to work. For more information see the Configuring a Proxy guide.

Create a Google Cloud storage bucket

You will need the following details when you configure CircleCI server.

- **Storage Bucket Name** - The bucket used for server.
- You can choose one of the following:
  - **Service Account JSON** - A JSON format key of the Service Account to use for bucket access.
  - **Service Account Email** - Service Account Email id if using Google Workload Identity.

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 in the IAM console:

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

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

1. **Create a GCP bucket**

   If your server installation runs within a GKE cluster, ensure that your current IAM user is a cluster admin for this cluster, as RBAC (role-based access control) objects need to be created. More information can be found in the GKE documentation.

   ```
   gsutil mb gs://circleci-server-bucket
   ```

2. **Create a Service Account**

   ```
   gcloud iam service-accounts create circleci-server --display-name "circleci-server service account"
   ```

   You will need the email for the service account in the next step. Run the following command to find it:

   ```
   gcloud iam service-accounts list \
     --filter="displayName:circleci-server account" \
     --format 'value(email)'
   ```

3. **Grant Permissions to Service Account**

```
gcloud iam roles create circleci_server \
    --project <PROJECT_ID> \
    --title "CircleCI Server"
```

```
gcloud projects add-iam-policy-binding <PROJECT_ID> \
    --member serviceAccount:<SERVICE_ACCOUNT_EMAIL> \
    --role projects/<PROJECT_ID>/roles/circleci_server
```

```
gsutil iam ch serviceAccount:<SERVICE_ACCOUNT_EMAIL>:objectAdmin gs://circleci-server-bucket
```

4. **JSON Key File**

   This step is NOT required if using Workload Identities.

   After running the following command, you should have a file named `circleci-server-keyfile` in your local working directory. You will need this when you configure your server installation.

```
gcloud iam service-accounts keys create circleci-server-keyfile \
    --iam-account <SERVICE_ACCOUNT_EMAIL>
```

5. Enable workload Identity

   This step is required only if you are using Workload Identities for GKE. Steps to enable Workload Identities are here

```
gcloud iam service-accounts add-iam-policy-binding <YOUR_SERVICE_ACCOUNT_EMAIL> \
    --role roles/iam.workloadIdentityUser \
    --member "serviceAccount:<GCP_PROJECT_ID>.svc.id.goog[circleci-server/object-storage]"
```

```
gcloud projects add-iam-policy-binding <GCP_PROJECT_ID> \
    --member serviceAccount:<YOUR_SERVICE_ACCOUNT_EMAIL> \
    --role roles/iam.serviceAccountTokenCreator \
    --condition=None
```

If you are switching from static JSON credentials to Workload Identity, you should delete the keys from GCP as well as from CircleCI KOTS Admin Console.

CircleCI Server v3.x Installation Phase 2

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

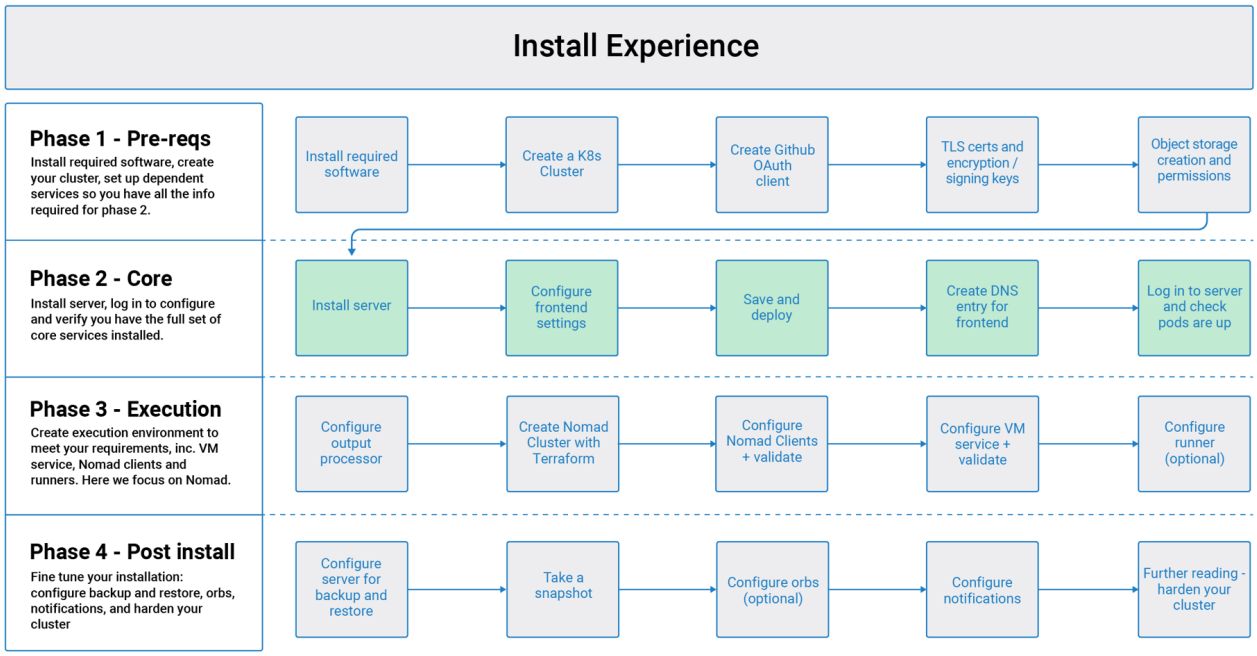


Figure 4. Installation Experience Flow Chart Phase 2

In the following sections replace any items or credentials displayed between < > with your details.

Phase 2: Core services installation

CircleCI server v3.x uses KOTS from Replicated for installation management and distribution.

1. Ensure you are running the minimum KOTS version (1.65.0) by running the command:

```
kubectl kots version
```

The KOTS command opens 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.

2. From the terminal, run (if you are installing behind a proxy see Installing behind HTTP Proxy):

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

You will be prompted for:

- namespace for the deployment
- password for the KOTS Admin Console

3. When complete, you should be provided with a URL to access the KOTS admin console, usually `http://localhost:8800`.

If you need to get back to the KOTS admin console at a later date, run: `kubectl kots admin-console -n <YOUR_CIRCLECI_NAMESPACE>`

Once you have created your namespace, we recommend setting your `kubectl` context too with the following command: `kubectl config set-context --current --namespace <namespace>`

Installing behind an HTTP Proxy (optional)

If you wish to install CircleCI server behind a proxy, use the following command structure should be used for step 2 above (for more information see the KOTS docs here):

```
kubectl kots install circleci-server --http-proxy <YOUR_HTTP_PROXY_URI> --https-proxy <https-proxy> --no-proxy
<YOUR_NO_PROXY_LIST>
```

The load balancer endpoints must be added to the no-proxy list for the following services: `output processor` and `vm-service`. This is because the no-proxy list is shared between the application and build-agent. The application and build-agent are assumed to be behind the same firewall and therefore cannot have a proxy between them.

For further information see the Configuring a Proxy guide.

Frontend Settings

Frontend settings control the web-application-specific aspects of the CircleCI system.

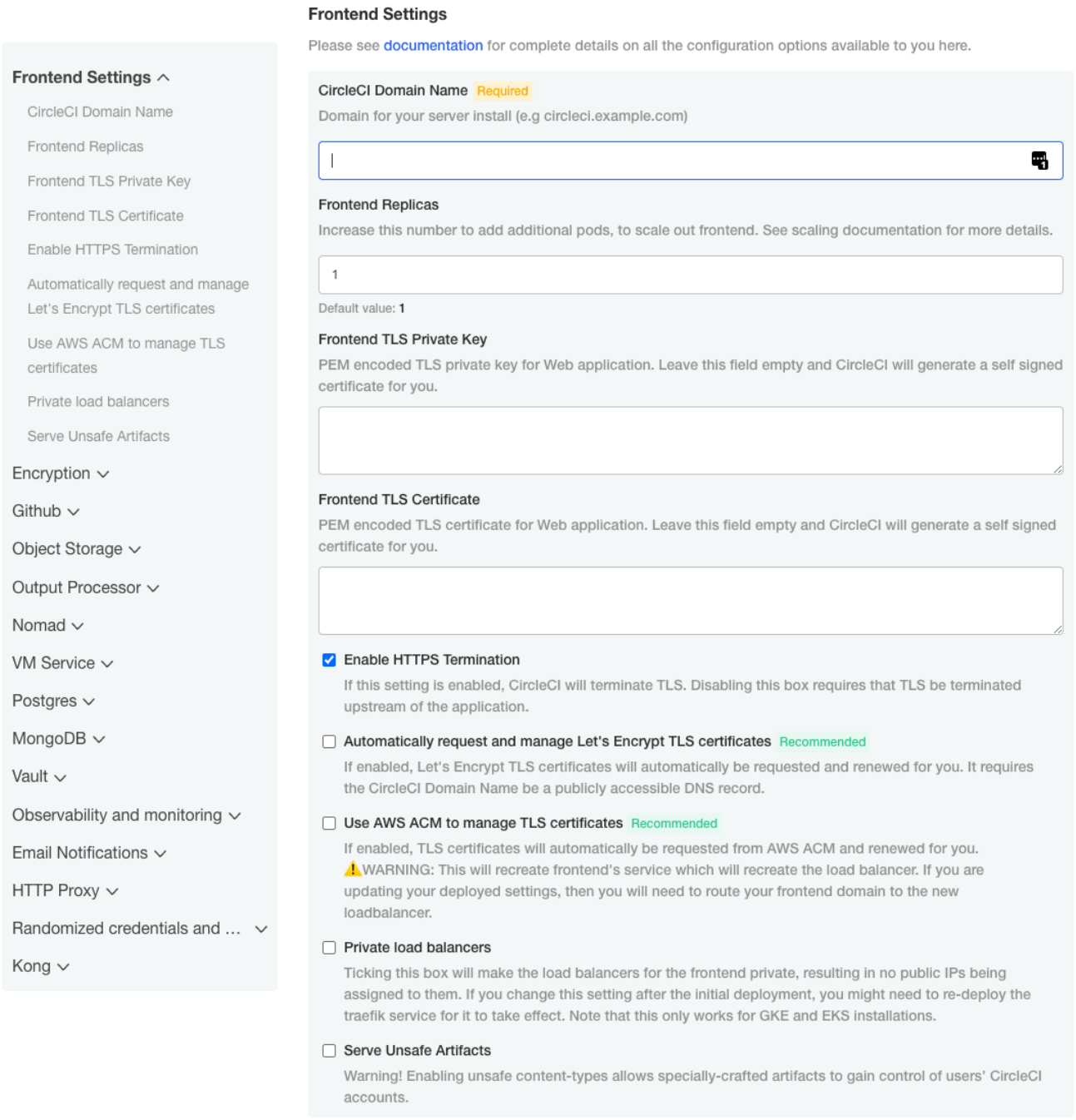


Figure 5. Frontend Settings

Complete the fields described below.

- **CircleCI Domain Name (required)** - Enter the domain name you specified when creating your Frontend TLS key and certificate.
- **Frontend Replicas** - Used to increase the amount of traffic that can be handled by the frontend.
- **Frontend TLS Private Key (required)** - You created this during your prerequisite steps. You can retrieve this value with the following command:

```
cat /etc/letsencrypt/live/<CIRCLECI_SERVER_DOMAIN>/privkey.pem
```

- **Frontend TLS Certificate (required)** - You created this during your prerequisite steps. You can retrieve this value with the following command:

```
cat /etc/letsencrypt/live/<CIRCLECI_SERVER_DOMAIN>/fullchain.pem
```

- **Private Load Balancer (optional)** - Load balancer does not generate external IP addresses.

If you are selecting the option to use private load balancers, the Let's Encrypt option will no longer work and become unavailable.

For the **Frontend TLS private key and certificate** you have 4 options:

- You can supply a private key and certificate
- Check the box that allows Let's Encrypt to automatically request and manage certificates for you.
- Check the box that allows AWS Certificate Manager (ACM) to automatically request and manage certificates for you. For more information about using ACM see the Using ACM TLS certificates section below.
- You can also disable TLS termination at this point, but the system will still need to be accessed over HTTPS.

Using ACM TLS certificates

If you would like to use AWS Certificate Manager (ACM) to manage your TLS certificates, follow the ACM documentation for instructions on how to generate ACM certificates.

Once you have generated your certificates, enable ACM from the KOTS admin console under the Frontend section. Check the ACM box and provide your ACM ARN (Amazon Resource Name).

If you have already deployed CircleCI server, enabling ACM is a destructive change to your frontend service. The service will have to be regenerated to allow the use of your ACM certificates and so the associated loadbalancer will also be regenerated. You will need to reroute your DNS records to the new loadbalancer once you have redeployed CircleCI server.

Encryption

Encryption and artifact signing keys were created during prerequisites phase. You can enter them here now.

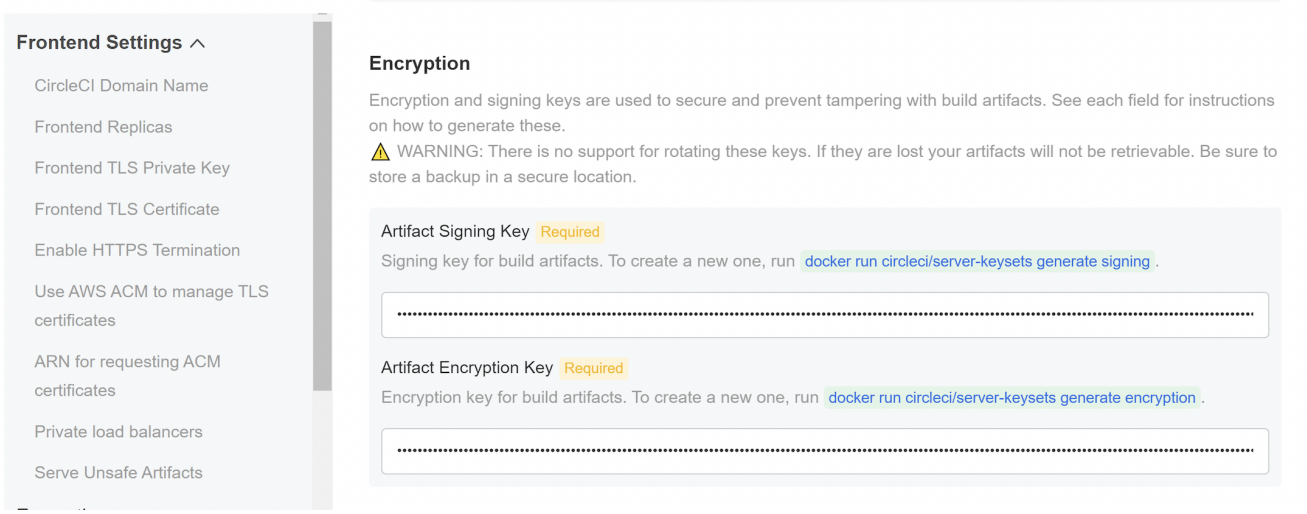


Figure 6. Encryption Settings

Complete the following fields:

- **Artifact Signing Key (required)**
- **Encryption Signing Key (required)**

GitHub

You created your GitHub OAuth application in the prerequisite phase. Use the data to complete the following settings:

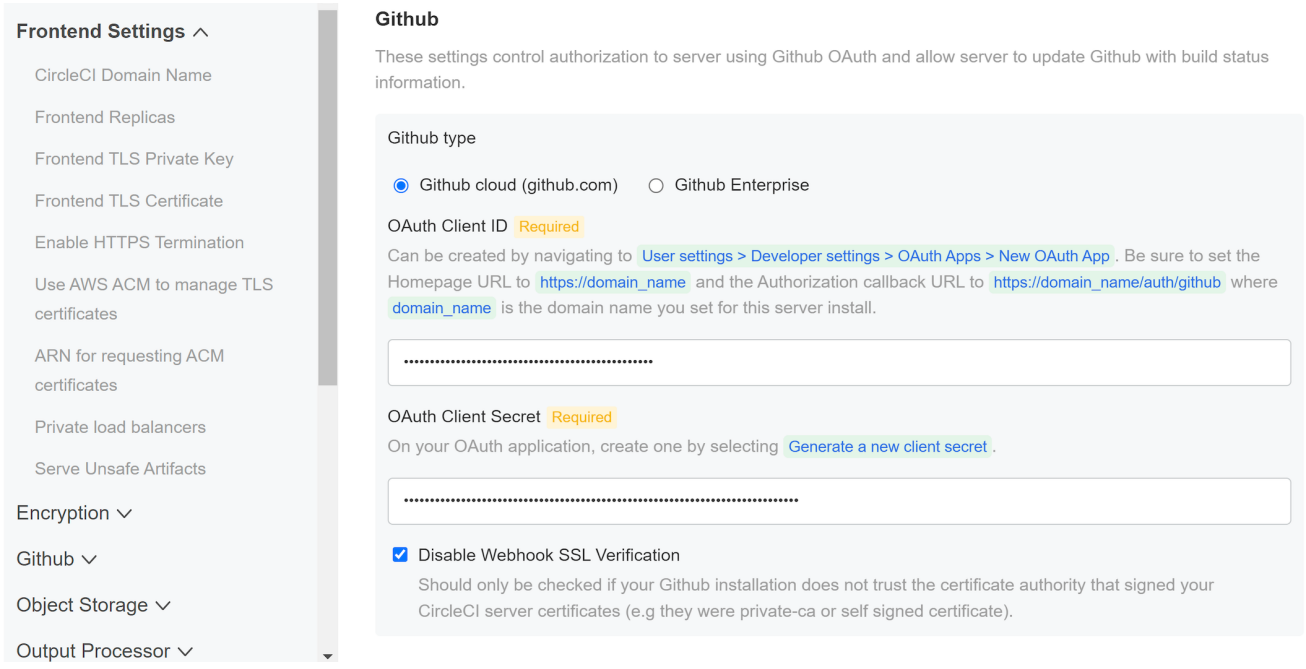


Figure 7. GitHub Settings

- **GitHub Type (required)** - Select Cloud or Enterprise (on premises).
- **OAuth Client ID (required)** - The OAuth Client ID provided by GitHub.
- **OAuth Client Secret (required)** - The OAuth Client Secret provided by GitHub.
- **GitHub Enterprise Fingerprint** - Required when using a proxy. Include the output of `ssh-keyscan github.example.com` in the text field.

Object storage

You created your Object Storage Bucket and Keys in the prerequisite steps. Use the data to complete the following settings depending on your platform.

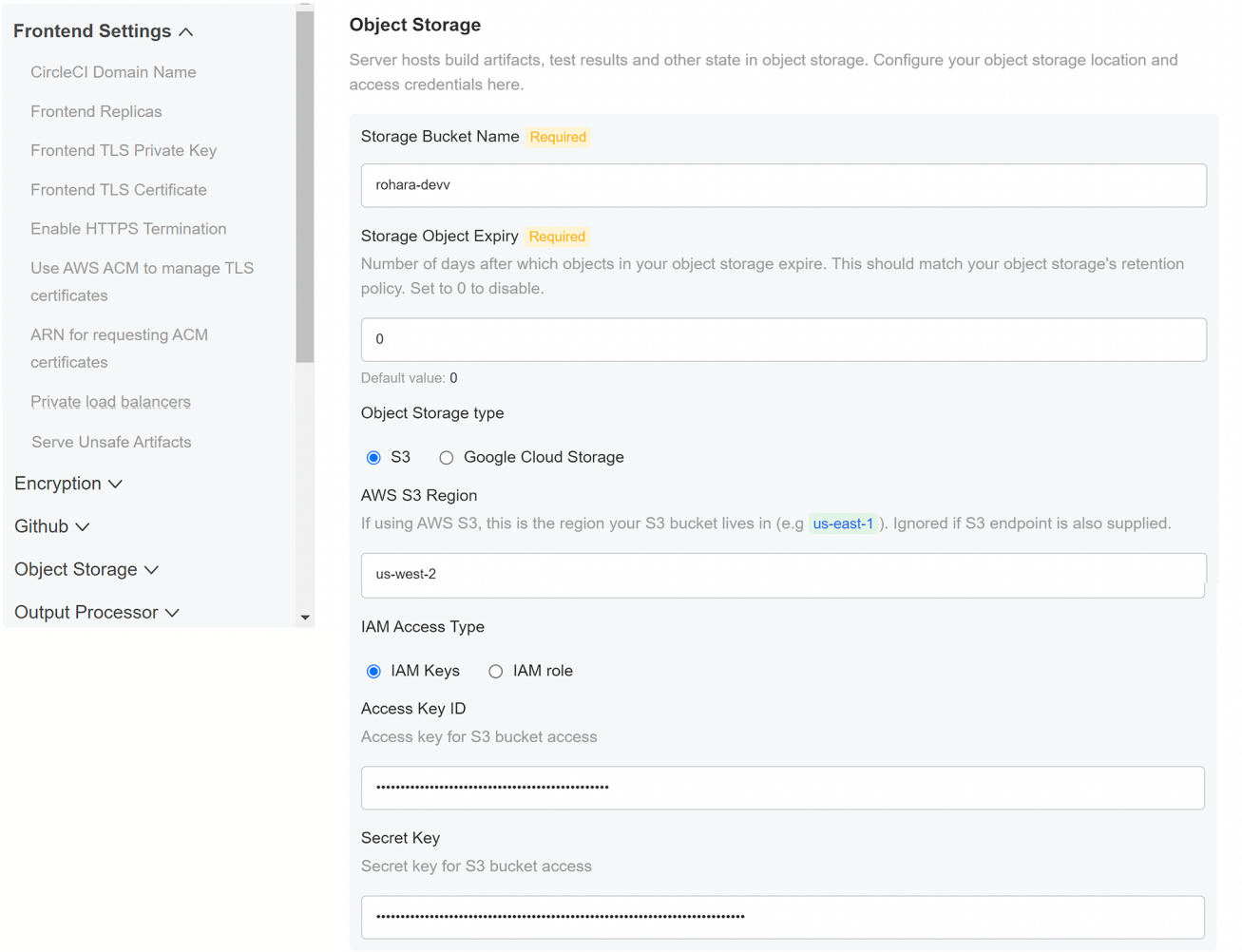


Figure 8. Object Storage Settings

Google Cloud Storage

You should have created your Google Cloud Storage bucket and service account during the prerequisite steps.

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

Authentication

- You can choose one of the following:
  - **Service Account JSON (required)** - A JSON format key of the Service Account to use for bucket access.
  - **Service Account Email (required)** - Service Account Email id if using Google Workload Identity.

Skip over the next few sections - **Output Processor**, **Nomad** and **VM Service**. We will set these up in the next phase of the installation.

Postgres, MongoDB, Vault settings

You can skip these sections unless you plan on using an existing Postgres, MongoDB or Vault instance, in which case, see the Externalizing Services doc. By default, CircleCI server v3.x will create its own Postgres, MongoDB and Vault instances within the CircleCI namespace. The instances inside the CircleCI namespace will be included in the CircleCI backup and restore process.

Save and deploy

Once you have completed the fields detailed above, you can deploy. The deployment installs the core services and provides you with an IP address for the Kong load balancer. That IP address is critical in setting up a DNS record and completing the first phase of the installation.

From server v3.3.0, we have replaced Traefik with Kong as our reverse proxy. However, to minimize disruption when upgrading, we chose not to rename the service used by Kong. Although you will see a service named `circleci-server-traefik`, this service is actually for Kong.

Create DNS entry

Create a DNS entry for your Kong load balancer, for example, `circleci.your.domain.com` and `app.circleci.your.domain.com`. The DNS entry should align with the DNS names used when creating your TLS certificate and GitHub OAuth app during the prerequisites steps. All traffic will be routed through this DNS record.

You need the IP address or, if using AWS, the DNS name of the Kong load balancer. You can find this information with the following command:

```
kubectl get service circleci-server-traefik --namespace=<YOUR_CIRCLECI_NAMESPACE>
```

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)

The Kong load balancer has a healthcheck that serves a JSON payload at https://loadbalancer-address/status

Validation

You should now be able to navigate to your CircleCI server installation and log in to the application successfully.

Now we will move on to build services. It may take a while for all your services to be up. You can periodically check by running the following command (you are looking for the “frontend” pod to show a status of *running* and **ready** should show 1/1):

```
kubectl get pods -n <YOUR_CIRCLECI_NAMESPACE>
```

CircleCI Server v3.x Installation Phase 3

Before you begin with the CircleCI server v3.x execution installation phase, ensure you have run through Phase 1 – Prerequisites and Phase 2 - Core services installation.

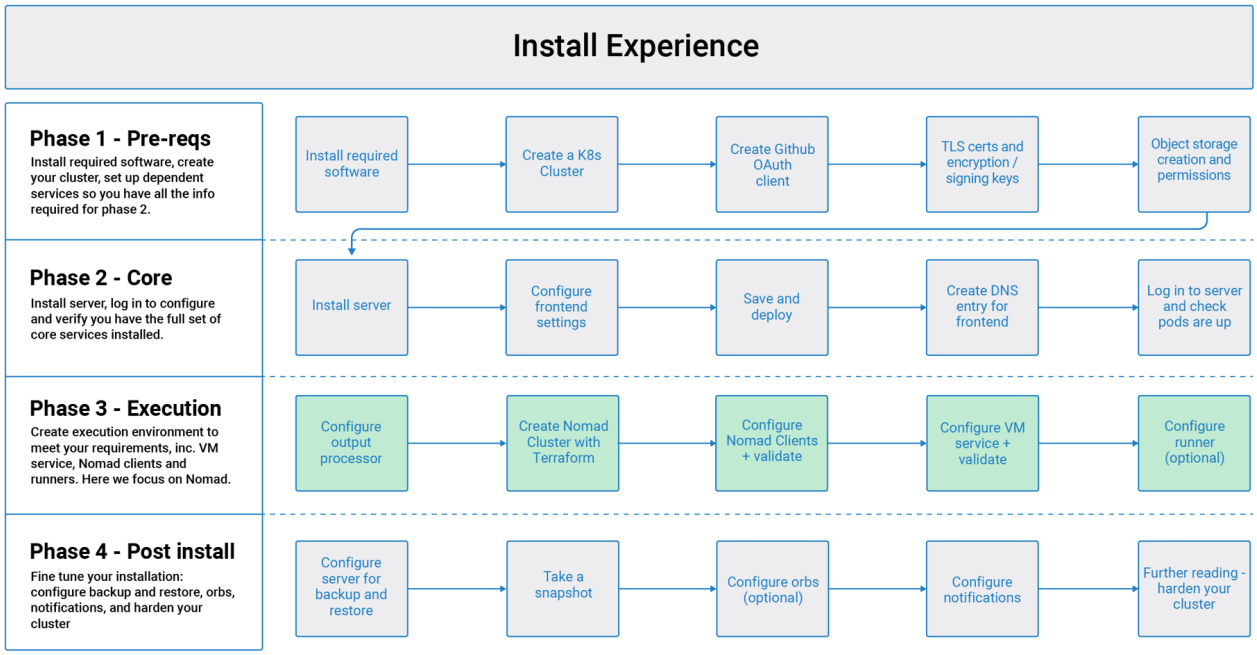


Figure 9. Installation Experience Flow Chart Phase 3

In the following sections, replace any items or credentials displayed between < > with your details.

Phase 3: Execution environment installation

Output Processor

Output processor is responsible for handling the output from Nomad clients. It is a key service to scale if you find your system slowing down. We recommend increasing the output processor replica set to scale the service up to meet demand.

Access the KOTS Admin Console by running the following command, substituting your namespace: `kubectl kots admin-console -n <YOUR_CIRCLECI_NAMESPACE>`

Locate and enter the following in Settings:

1. **Output Processor Load Balancer Hostname** - The following command provides the IP address of the service:

```
kubectl get service output-processor --namespace=<YOUR_CIRCLECI_NAMESPACE>
```

2. **Save your configuration**. You will deploy and validate your setup after you complete Nomad

client setup.

Nomad Clients

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

Nomad clients are installed outside of the Kubernetes cluster, while their control plane (Nomad Server) is installed within the cluster. Communication between your Nomad Clients and the Nomad control plane is secured with mTLS. The mTLS certificate, private key, and certificate authority will be output after you complete installation of the Nomad Clients.

Once completed, you can update your CircleCI server configuration so your Nomad control plane can communicate with your Nomad Clients.

Cluster Creation with Terraform

CircleCI curates Terraform modules to help install Nomad clients in your chosen cloud provider. You can browse the modules in our public repository, including example Terraform config files (`main.tf`) for both AWS and GCP. Some information about your cluster and server installation is required to complete your `main.tf`. How to get this information is described in the following sections.

If you would also like to set up Nomad Autoscaler at this stage, see the Nomad Autoscaler section of this guide, as some of the requirements can be included in this Terraform setup.

GCP

You need the IP address of the Nomad control plane (Nomad Server), which was created when you deployed CircleCI Server. You can get the IP address by running the following command:

```
kubectl get service nomad-server-external --namespace=<YOUR_CIRCLECI_NAMESPACE>
```

You also need the following information:

- The GCP Project you want to run Nomad clients in.
- The GCP Zone you want to run Nomad clients in.
- The GCP Region you want to run Nomad clients in.
- The GCP Network you want to run Nomad clients in.
- The GCP Subnetwork you want to run Nomad clients in.

You can copy the following example to your local environment and fill in the appropriate information for your specific setup.

```
variable "project" {
  type    = string
```

```
  default = "<your-project>"
}

variable "region" {
  type    = string
  default = "<your-region>"
}

variable "zone" {
  type    = string
  default = "<your-zone>"
}

variable "network" {
  type    = string
  default = "<your-network-name>"
  # if you are using a shared vpc, provide the network endpoint rather than the name. eg:
  # default = "https://www.googleapis.com/compute/v1/projects/<host-project>/global/networks/<your-network-name>"
}

variable "subnetwork" {
  type    = string
  default = "<your-subnetwork-name>"
  # if you are using a shared vpc, provide the network endpoint rather than the name. eg:
  # default = "https://www.googleapis.com/compute/v1/projects/<service-project>/regions/<your-region>/subnetworks/<your-
subnetwork-name>"
}


variable "server_endpoint" {
  type    = string
  default = "<nomad-server-loadbalancer>:4647"
}

variable "nomad_auto_scaler" {
  type        = bool
  default     = false
  description = "If true, terraform will create a service account to be used by nomad autoscaler."
}

variable "enable_workload_identity" {
  type        = bool
  default     = false
  description = "If true, Workload Identity will be used rather than static credentials'"
}

variable "k8s_namespace" {
  type        = string
  default     = "circleci-server"
  description = "If enable_workload_identity is true, provide application k8s namespace"
}

provider "google-beta" {
  project = var.project
  region  = var.region
  zone    = var.zone
}


module "nomad" {
  source = "git::https://github.com/CircleCI-Public/server-terraform.git//nomad-gcp?ref=3.4.0"

  zone            = var.zone
  region          = var.region
  network         = var.network
  subnetwork      = var.subnetwork
```

```
  server_endpoint = var.server_endpoint
  machine_type    = "n2-standard-8"
  nomad_auto_scaler         = var.nomad_auto_scaler
  enable_workload_identity  = var.enable_workload_identity
  k8s_namespace             = var.k8s_namespace

  unsafe_disable_mtls    = false
  assign_public_ip       = true
  preemptible            = true
  target_cpu_utilization = 0.50
}

output "module" {
  value = module.nomad
}
```

Once you have filled in the appropriate information, you can deploy your Nomad clients by running the following commands:

```
terraform init
```

```
terraform plan
```

```
terraform apply
```

After Terraform is done spinning up the Nomad client(s), it outputs the certificates and key needed for configuring the Nomad control plane in CircleCI server. Copy them somewhere safe.

Nomad Autoscaler

Nomad provides a utility to automatically scale up or down your Nomad clients, provided your clients are managed by a cloud provider's autoscaling resource. With Nomad Autoscaler, you only need to provide permission for the utility to manage your autoscaling resource and specify where it is located. You can enable this resource via KOTS, which deploys the Nomad Autoscaler service along with your Nomad servers. Below, we go through how to set up Nomad Autoscaler for your provider.

The maximum and minimum Nomad client count overwrite the corresponding values set when you created your autoscaling group or managed instance group. It is recommended that you keep these values and those used in your Terraform the same so that they do not compete.

If you do not require this service, click the **Save config** button to update your installation and redeploy server.

GCP

1. Create a service account for Nomad Autoscaler
    - Our nomad module creates a service acount and outputs a file with the keys if you set the

variables `nomad_auto_scaler = true` and `enable_workload_identity = false`. You may reference the examples in the link for more details. If you have already created the clients, simply update the variable and run `terraform apply`. The created user's key will be available in a file named `nomad-as-key.json`. If you are using GKE Workload Identities, set the variables `nomad_auto_scaler = true` and `enable_workload_identity = true`.

    - You may also create a nomad gcp service account manually. The service account will need the role `compute.admin`. It will also need the role `iam.workloadIdentityUser` if using Workload identities

2. Set Nomad Autoscaler to `enabled`
3. Set Maximum Node Count*
4. Set Minimum Node Count*
5. Select cloud provider: `Google Cloud Platform`
6. Add your Project ID
7. Add Managed Instance Group Name
8. Instance group type: Zonal or Regional.
9. You can choose one of the following:
    a. JSON of GCP service account for Nomad Autoscaler
    b. Or, the Nomad Autoscaler Service Account Email Address if using Workload Identities. Steps to enable Workload Identities on GCP cluster are here.
    c. Enable workload identity for `nomad-autoscaler` (kubernetes) service account

```
gcloud iam service-accounts add-iam-policy-binding <YOUR_SERVICE_ACCOUNT_EMAIL> \
    --role roles/iam.workloadIdentityUser \
    --member "serviceAccount:<GCP_PROJECT_ID>.svc.id.goog[circleci-server/nomad-autoscaler]"
```

If you are switching from static JSON credentials to Workload Identity, you should delete the keys from GCP as well as from CircleCI KOTS Admin Console.

Configure and Deploy

Now that you have successfully deployed your Nomad clients, you can configure CircleCI server and the Nomad control plane. Access the KOTS Admin Console by running the following command, substituting your namespace: `kubectl kots admin-console -n <YOUR_CIRCLECI_NAMESPACE>`

Enter the following in Settings:

- **Nomad Load Balancer (required)**

```
kubectl get service nomad-server-external --namespace=<YOUR_CIRCLECI_NAMESPACE>
```

- **Nomad Server Certificate (required)** - Provided in the output from `terraform apply`

- **Nomad Server Private Key (required)** - Provided in the output from `terraform apply`
- **Nomad Server Certificate Authority (CA) Certificate (required)** - Provided in the output from `terraform apply`
- **Build Agent Image** - If you want to use a custom Docker registry to supply the CircleCI Build Agent, contact customer support for assistance.

Click the **Save config** button to update your installation and redeploy server.

Nomad Clients Validation

CircleCI has created a project called realitycheck which allows you to test your Server installation. We are going to follow the project so we can verify that the system is working as expected. As you continue through the next phase, sections of realitycheck will move from red to green.

To run realitycheck, you need to clone the repository. Depending on your GitHub setup, you can use one of the following commands:

GitHub Cloud

```
git clone -b server-3.0 https://github.com/circleci/realitycheck.git
```

GitHub Enterprise

```
git clone -b server-3.0 https://github.com/circleci/realitycheck.git
git remote set-url origin <YOUR_GH_REPO_URL>
git push
```

Once you have successfully cloned the repository, you can follow it from within your CircleCI server installation. You need to set the following variables. For full instructions please see the repository readme.

Table 1. Environmental Variables

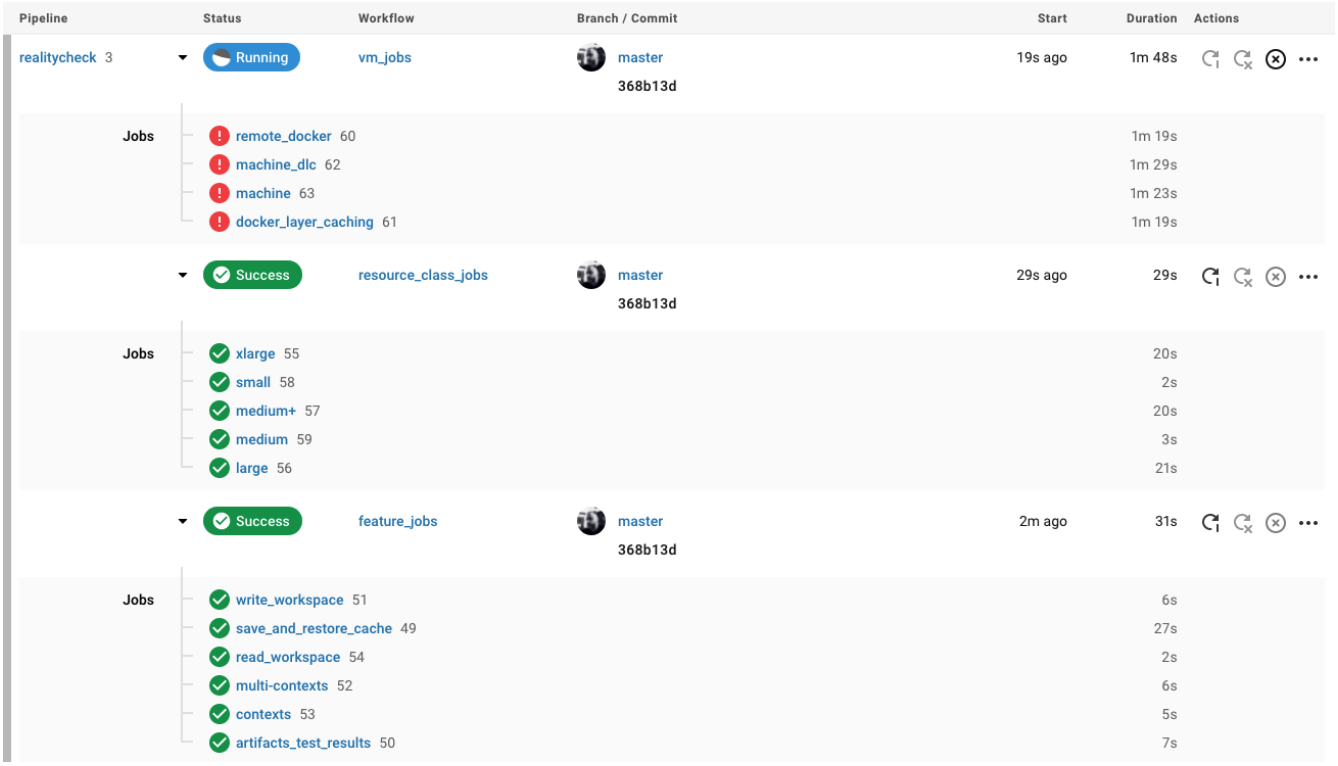
Name	Value
CIRCLE_HOSTNAME	<YOUR_CIRCLECI_INSTALLATION_URL>
CIRCLE_TOKEN	<YOUR_CIRCLECI_API_TOKEN>

Table 2. Contexts

Name	Environmental Variable Key	Environmental Variable Value
org-global	CONTEXT_END_TO_END_TEST_VAR	Leave blank
individual-local	MULTI_CONTEXT_END_TO_END_VAR	Leave blank

Once you have configured the environmental variables and contexts, rerun the realitycheck tests.

You should see the features and resource jobs complete successfully. Your test results should look something like the following:



VM service

VM service configures VM and remote docker jobs. You can configure a number of options for VM service, such as scaling rules. VM service is unique to EKS and GKE installations because it specifically relies on features of these cloud providers.

GCP

You need additional information about your cluster to complete the next section. Run the following command:

```
gcloud container clusters describe
```

This command returns something like the following, which includes network, region and other details that you need to complete the next section:

```
addonsConfig:
  gcePersistentDiskCsiDriverConfig:
    enabled: true
  kubernetesDashboard:
    disabled: true
  networkPolicyConfig:
    disabled: true
clusterIpv4Cidr: 10.100.0.0/14
createTime: '2021-08-20T21:46:18+00:00'
currentMasterVersion: 1.20.8-gke.900
```

```
currentNodeCount: 3
currentNodeVersion: 1.20.8-gke.900
databaseEncryption:
...
```

1. **Create firewall rules**

   Run the following commands to create a firewall rule for VM service in GKE:

   ```
   gcloud compute firewall-rules create "circleci-vm-service-internal-nomad-fw" --network "<network>" --action allow
   --source-ranges "0.0.0.0/0" --rules "TCP:22,TCP:2376"
   ```

   If you have used auto-mode, you can find the Nomad clients CIDR based on the region by referring to the table here.

   ```
   gcloud compute firewall-rules create "circleci-vm-service-internal-k8s-fw" --network "<network>" --action allow
   --source-ranges "<clusterIpv4Cidr>" --rules "TCP:22,TCP:2376"
   ```

   ```
   gcloud compute firewall-rules create "circleci-vm-service-external-fw" --network "<network>" --action allow --rules
   "TCP:54782"
   ```

2. **Create user**

   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.

   ```
   gcloud iam service-accounts create circleci-server-vm --display-name "circleci-server-vm service account"
   ```

   If your are deploying CircleCI server in a shared VCP, you should create this user in the project in which you intend to run your VM jobs.

3. **Get the service account email address**

   ```
   gcloud iam service-accounts list --filter="displayName:circleci-server-vm service account" --format 'value(email)'
   ```

4. **Apply role to service account**

   Apply the Compute Instance Admin (Beta) role to the service account:

   ```
   gcloud projects add-iam-policy-binding <YOUR_PROJECT_ID> --member serviceAccount:<YOUR_SERVICE_ACCOUNT_EMAIL> --role
   roles/compute.instanceAdmin --condition=None
   ```

And

```
gcloud projects add-iam-policy-binding <YOUR_PROJECT_ID> --member serviceAccount:<YOUR_SERVICE_ACCOUNT_EMAIL> --role
roles/iam.serviceAccountUser --condition=None
```

5. **Get JSON Key File**

   If you are using Workload Identities for GKE, this step is not required.

   After running the following command, you should have a file named `circleci-server-vm-keyfile` in your local working directory. You will need this when you configure your server installation.

   ```
   gcloud iam service-accounts keys create circleci-server-vm-keyfile --iam-account <YOUR_SERVICE_ACCOUNT_EMAIL>
   ```

6. **Enable Workload Identity for Service Account**

   This step is required only if you are using Workload Identities for GKE. Steps to enable Workload Identities are here

   ```
   gcloud iam service-accounts add-iam-policy-binding <YOUR_SERVICE_ACCOUNT_EMAIL> \
       --role roles/iam.workloadIdentityUser \
       --member "serviceAccount:<GCP_PROJECT_ID>.svc.id.goog[circleci-server/vm-service]"
   ```

If you are switching from static JSON credentials to Workload Identity, you should delete the keys from GCP as well as from CircleCI KOTS Admin Console.

1. **Configure Server**

   Configure VM Service through the KOTS Admin Console. Details of the available configuration options can be found in the VM Service guide.

   Once you have configured the fields, **save your config** and deploy your updated application.

VM Service Validation

Once you have configured and deployed CircleCI server, you should validate that VM Service is operational. You can rerun the realitycheck project within your CircleCI installation and you should see the VM Service Jobs complete. At this point, all tests should pass.

Runner

Overview

CircleCI runner does not require any additional server configuration. Server ships ready to work with runner. However, you need to create a runner and configure the runner agent to be aware of your server installation. For complete instructions for setting up runner, see the runner

documentation.

Runner requires a namespace per organization. Server can have many organizations. If your company has multiple organizations within your CircleCI installation, you need to set up a runner namespace for each organization within your server installation.

CircleCI Server v3.x Installation Phase 4

Before you begin with the CircleCI server v3.x post installation phase, ensure you have run through Phase 1 – Prerequisites, Phase 2 - Core services installation and Phase 3 - Build services installation.

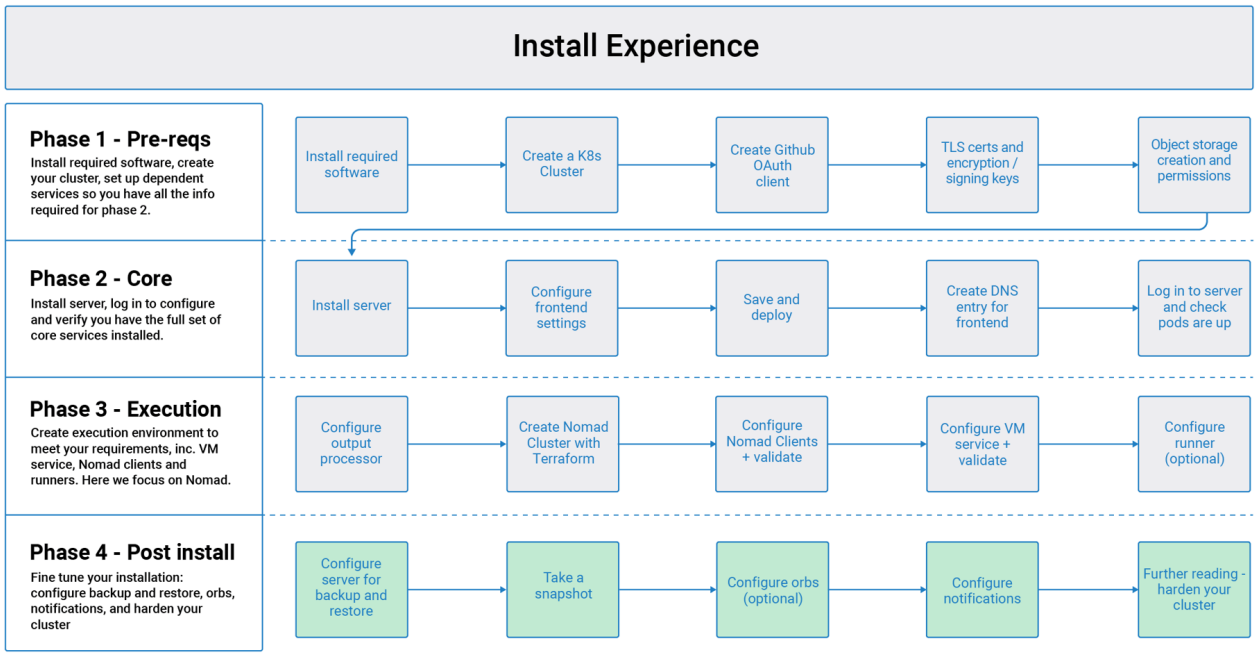


Figure 10. Installation Experience Flow Chart Phase 4

In the following sections, replace any items or credentials displayed between < > with your details.

Phase 4: Post installation

Set up backup and restore

Backups of CircleCI server can be created through KOTS. To enable backup support, you will need to install and configure Velero on your cluster. Velero was listed in the installation prerequisites section, so you should already have it.

Server 3.x backups on GCP

The following steps are specific for Google Cloud Platform and it is assumed you have met the prerequisites.

These instructions were sourced from the documentation for the Velero GCP plugin here.

Step 1 - Create a GCP bucket

To reduce the risk of typos, you can set some of the parameters as shell variables. Should you be unable to complete all the steps in the same session, do not forget to reset variables as necessary before proceeding. In the step below, for example, you can define a variable for your bucket name.

Replace the <YOUR_BUCKET> placeholder with the name of the bucket you want to create for your backups.

```
BUCKET=<YOUR_BUCKET>

gsutil mb gs://$BUCKET/
```

Step 2 - Setup permissions for Velero

If your server installation runs within a GKE cluster, ensure that your current IAM user is a cluster admin for this cluster, as RBAC objects need to be created. More information can be found in the GKE documentation.

1. First, you will set a shell variable for your project ID. To do so, make sure that your `gcloud` CLI points to the correct project by looking at the current configuration:

   ```
   gcloud config list
   ```

2. If the project is correct, set the variable:

   ```
   PROJECT_ID=$(gcloud config get-value project)
   ```

3. Create a service account:

   ```
   gcloud iam service-accounts create velero \
       --display-name "Velero service account"
   ```

   If you run several clusters with Velero, consider using a more specific name for the Service Account besides `velero`, as suggested above.

4. You can check if the service account has been created successfully by running the following command:

   ```
   gcloud iam service-accounts list
   ```

5. Next, store the email address for the Service Account in a variable:

   ```
   SERVICE_ACCOUNT_EMAIL=$(gcloud iam service-accounts list \
     --filter="displayName:Velero service account" \
     --format 'value(email)')
   ```

   Modify the command as needed to match the display name you have chosen for your Service Account.

6. Grant the necessary permissions to the Service Account:

```
ROLE_PERMISSIONS=(
    compute.disks.get
    compute.disks.create
    compute.disks.createSnapshot
    compute.snapshots.get
    compute.snapshots.create
    compute.snapshots.useReadOnly
    compute.snapshots.delete
    compute.zones.get
)

gcloud iam roles create velero.server \
    --project $PROJECT_ID \
    --title "Velero Server" \
    --permissions "$(IFS=","; echo "${ROLE_PERMISSIONS[*]}")"

gcloud projects add-iam-policy-binding $PROJECT_ID \
    --member serviceAccount:$SERVICE_ACCOUNT_EMAIL \
    --role projects/$PROJECT_ID/roles/velero.server

gsutil iam ch serviceAccount:$SERVICE_ACCOUNT_EMAIL:objectAdmin gs://${BUCKET}
```

Now you need to ensure that Velero can use this Service Account.

Option 1: JSON key file

You can simply pass a JSON credentials file to Velero to authorize it to perform actions as the Service Account. To do this, you first need to create a key:

```
gcloud iam service-accounts keys create credentials-velero \
    --iam-account $SERVICE_ACCOUNT_EMAIL
```

After running this command, you should see a file named `credentials-velero` in your local working directory.

Option 2: Workload Identities

If you are already using Workload Identities in your cluster, you can bind the GCP Service Account you just created to Velero's Kubernetes service account. In this case, the GCP Service Account needs the `iam.serviceAccounts.signBlob` role in addition to the permissions already specified above.

If you are switching from static JSON credentials to Workload Identity, you should delete the keys from GCP as well as from CircleCI KOTS Admin Console.

Step 3 - Install and start Velero

- Run one of the following `velero install` commands, depending on how you authorized the service account. This creates a namespace called `velero` and installs all the necessary resources to run Velero.

KOTS backups require restic to operate. When installing Velero, ensure that you have the `--use-restic` flag set.

If using a JSON key file

```
velero install \
    --provider gcp \
    --plugins velero/velero-plugin-for-gcp:v1.2.0 \
    --bucket $BUCKET \
    --secret-file ./credentials-velero \
    --use-restic \
    --wait
```

If using Workload Identities

```
velero install \
    --provider gcp \
    --plugins velero/velero-plugin-for-gcp:v1.2.0 \
    --bucket $BUCKET \
    --no-secret \
    --sa-annotations iam.gke.io/gcp-service-account=$SERVICE_ACCOUNT_EMAIL \
    --backup-location-config serviceAccount=$SERVICE_ACCOUNT_EMAIL \
    --use-restic \
    --wait
```

For more options on customizing your installation, refer to the Velero documentation.

- Once Velero is installed on your cluster, check the new `velero` namespace. You should have a Velero deployment and a restic daemonset, for example:

```
$ kubectl get pods --namespace velero
NAME                      READY   STATUS    RESTARTS   AGE
restic-5vlww              1/1     Running   0          2m
restic-94ptv              1/1     Running   0          2m
restic-ch6m9              1/1     Running   0          2m
restic-mknws              1/1     Running   0          2m
velero-68788b675c-dm2s7   1/1     Running   0          2m
```

As restic is a daemonset, there should be one pod for each node in your Kubernetes cluster.

Server 3.x backups with S3 Compatible Storage

The following steps assume you are using S3-compatible object storage, but not necessarily AWS S3, for your backups. It is also assumed you have met the prerequisites.

These instructions were sourced from the Velero documentation here.

Step 1 - Configure `mc` client

To start, configure `mc` to connect to your storage provider:

```
# Alias can be any name as long as you use the same value in subsequent commands
export ALIAS=my-provider
mc alias set $ALIAS <YOUR_MINIO_ENDPOINT> <YOUR_MINIO_ACCESS_KEY_ID> <YOUR_MINIO_SECRET_ACCESS_KEY>
```

You can verify your client is correctly configured by running `mc ls my-provider` and you should see the buckets in your provider enumerated in the output.

Step 2 - Create a bucket

Create a bucket for your backups. It is important that a new bucket is used, as Velero cannot use a preexisting bucket that contains other content.

```
mc mb ${ALIAS}/<YOUR_BUCKET>
```

Step 3 - Create a user and policy

Next, create a user and policy for Velero to access your bucket.

In the following snippet `<YOUR_MINIO_ACCESS_KEY_ID>` and `<YOUR_MINIO_SECRET_ACCESS_KEY>` refer to the credentials used by Velero to access MinIO.

```
# Create user
mc admin user add $ALIAS <YOUR_MINIO_ACCESS_KEY_ID> <YOUR_MINIO_SECRET_ACCESS_KEY>

# Create policy
cat > velero-policy.json << EOF
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:*"
      ],
      "Resource": [
        "arn:aws:s3:::<YOUR_BUCKET>",
```

```
        "arn:aws:s3:::<YOUR_BUCKET>/*"
      ]
    }
  ]
}
EOF

mc admin policy add $ALIAS velero-policy velero-policy.json

# Bind user to policy
mc admin policy set $ALIAS velero-policy user=<YOUR_VELERO_ACCESS_KEY_ID>
```

Finally, you add your new user's credentials to a file (`./credentials-velero` in this example) with the following contents:

```
[default]
aws_access_key_id=<YOUR_VELERO_ACCESS_KEY_ID>
aws_secret_access_key=<YOUR_VELERO_SECRET_ACCESS_KEY>
```

Step 4 - Install and start Velero

Run the following `velero install` command. This creates a namespace called `velero` and installs all the necessary resources to run Velero.

KOTS backups require restic to operate. When installing Velero, ensure that you have the `--use-restic` flag set, as shown below:

```
velero install --provider aws \
  --plugins velero/velero-plugin-for-aws:v1.2.0 \
  --bucket <YOUR_BUCKET> \
  --secret-file ./credentials-velero \
  --use-volume-snapshots=false \
  --use-restic \
  --backup-location-config region=minio,s3ForcePathStyle="true",s3Url=<YOUR_ENDPOINT> \
  --wait
```

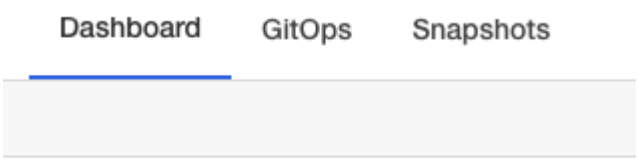
Once Velero is installed on your cluster, check the new `velero` namespace. You should have a Velero deployment and a restic daemonset, for example:

```
$ kubectl get pods --namespace velero
NAME                      READY   STATUS    RESTARTS   AGE
restic-5vlww              1/1     Running   0          2m
restic-94ptv              1/1     Running   0          2m
restic-ch6m9              1/1     Running   0          2m
restic-mknws              1/1     Running   0          2m
velero-68788b675c-dm2s7   1/1     Running   0          2m
```

As restic is a daemonset, there should be one pod for each node in your Kubernetes cluster.

Creating backups

Now that Velero is installed on your cluster, you should see the snapshots option in the navbar of the management console.



If you see this option, you are ready to create your first backup. If you do not see this option, please refer to the troubleshooting section.

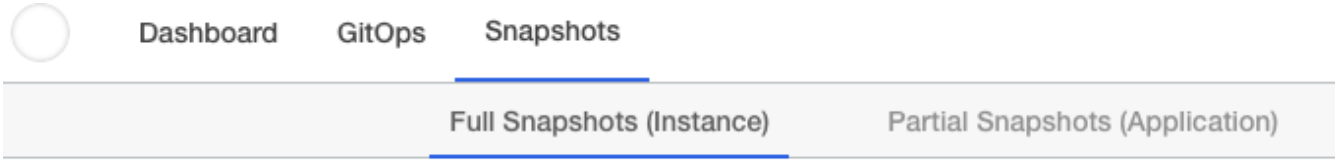
Option 1 - Create a backup with KOTS CLI

To create the backup, run the following command:

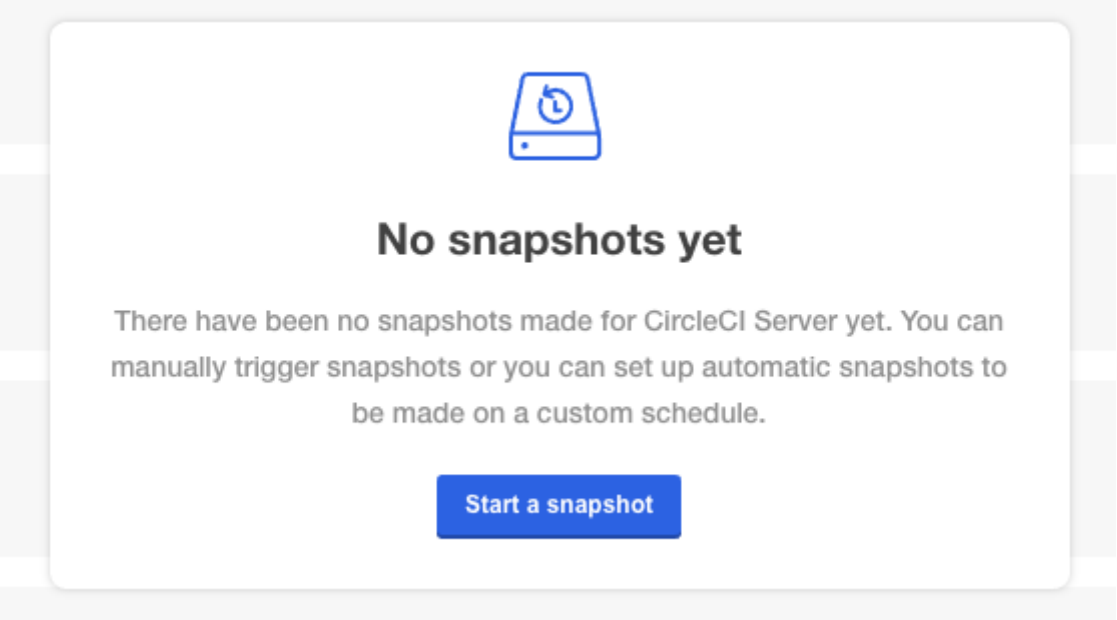
```
kubectl kots backup --namespace <your namespace>
```

Option 2 - Create a backup with KOTS Admin Console

Select **Snapshots** from the navbar. The default selection should be **Full Snapshots**, which is recommended.



Click the **Start a snapshot** button.



Orbs

Server installations include their own local orb registry. This registry is private to the server installation. All orbs referenced in project configs reference the orbs in the *server* orb registry. You are responsible for maintaining orbs. This includes:

- Copying orbs from the public registry.
- Updating orbs that may have been copied previously.
- Registering your company's private orbs, if you have any.

For more information and steps to complete these tasks, see the Orbs on Server guide.

Email Notifications

Build notifications are sent by email. This section has details on how to set up build notifications by email.

Access the KOTS admin console. Get to the KOTS admin console by running the following, substituting your namespace:

```
kubectl kots admin-console -n <YOUR_CIRCLECI_NAMESPACE>
```

Locate the **Email Notifications** section in **Settings** and fill in the following details to configure email notifications for your installation.

Locate the **Email Notifications** section in **Settings** and fill in the following details to configure email notifications for your installation:

- **Email Submission server hostname (required)** - Host name of the submission server (for example, for Sendgrid use smtp.sendgrid.net).
- **Username (required)** - Username to authenticate to submission server. This is commonly the

same as the user's email address.

- **Password (required)** - Password to authenticate to submission server.
- **Port (optional)** - 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.

- **Enable StartTLS** - Enabling this will encrypt mail submission.

StartTLS is used to encrypt mail by default, and you should only disable this if you can otherwise guarantee the confidentiality of traffic.

- **Email from address (required)** - The *from* address for the email.

Click the **Save config** button to update your installation and redeploy server.

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 experience problems, you can fall back to your 2.19 instance. Migration requires 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 not only allows you to gain a better understanding of the migration process, but also gives you an idea of 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 need to be backed up separately.
3. You have a new CircleCI Server 3.x installation.
4. You have successfully run realitycheck with contexts before 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 realitycheck 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 restarted, although you may manually restart it using the KOTS Admin 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 with 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 preflight 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 preflight 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, 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 option ensures 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 realitycheck with contexts on your new server 3.x environment by pushing a fresh commit.

Step 3 - Update your team

Once you have successfully run realitycheck, 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: `https://<APP_DOMAIN>/pipelines/github/<ORG>/<PROJECT>/jobs/<JOB_NUMBER>`

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) triggers 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 a follow. To trigger a build, perform an activity that triggers 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, contact support with a support bundle and the output from the migration script.

Transitioning to pipelines

When migrating from a server v2.x to a v3.x installation you will have project configurations made before the introduction of pipelines. Pipelines are automatically enabled for server v3.x installations so all you need to do is change your project configurations (`.circleci/_config.yml`) to `version: 2.1` to access all the features described in the section above.

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.

From server v3.3.0, we have replaced Traefik with Kong as our reverse proxy. However, in order to minimize disruption when upgrading, we chose not to rename the service used by Kong. Although you will see a service named `circleci-server-traefik`, this service is actually for Kong.

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 explains the minimum requirements for a server installation to work. Depending on your workloads, you might need to add additional rules to egress for Nomad clients and VMs. As nomenclature between cloud providers differs, 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 (for example, 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 explained here are assumed to be stateful and for TCP connections only, unless stated otherwise. If you are working with stateless rules, you need to create matching ingress or egress rules for the ones listed here.

Kubernetes Load Balancers

Depending on your setup, your load balancers might be transparent (that is, 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 Kubernetes nodes on the Kubernetes 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 using 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 when needed.

Egress

You most likely want all of your instances to access internet resources. This requires 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 (that is, the Nomad clients and external VMs) also will likely need to pull code from your VCS using 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

By default, the traffic within your Kuberntes cluster is regulated by networking policies. For most purposes, this should be sufficient to regulate the traffic between pods and there is no additional requirement to reduce traffic between Kubernetes nodes any further (it is fine to allow all traffic between Kubernetes 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 Kubernetes load balancers (30000-32767) should be all that is needed here for ingress. If you are using transparent load balancers, you 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

Port	Source	Purpose
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.