

INSTALLATION GUIDE

A guide for installing or migrating to CircleCI
Server v3.3.0 on AWS or GCP

docs@circleci.com

Version 3.3.0, 01/21/2022: FINAL

CircleCI Server v3.x Installation Phase 1	1
Phase 1: Prerequisites	1
CircleCI Server v3.x Installation Phase 2	16
Phase 2: Core services installation	16
CircleCI Server v3.x Installation Phase 3	22
Phase 3: Execution environment installation	22
CircleCI Server v3.x Installation Phase 4	39
Phase 4: Post installation	39
CircleCI Server v3.x Migration	51
Prerequisites	51
Migration	51
Frequently Asked Questions	53
CircleCI Server v3.x Hardening Your Cluster	54
Network Topology	54
Network Traffic	54
Kubernetes Load Balancers	54
Common Rules for Compute Instances	55
Kubernetes Nodes	55
Nomad Clients	56
External VMs	57

CircleCI Server v3.x Installation Phase 1

Phase 1: Prerequisites

Install CircleCI server 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 server 3.x [overview](#).



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

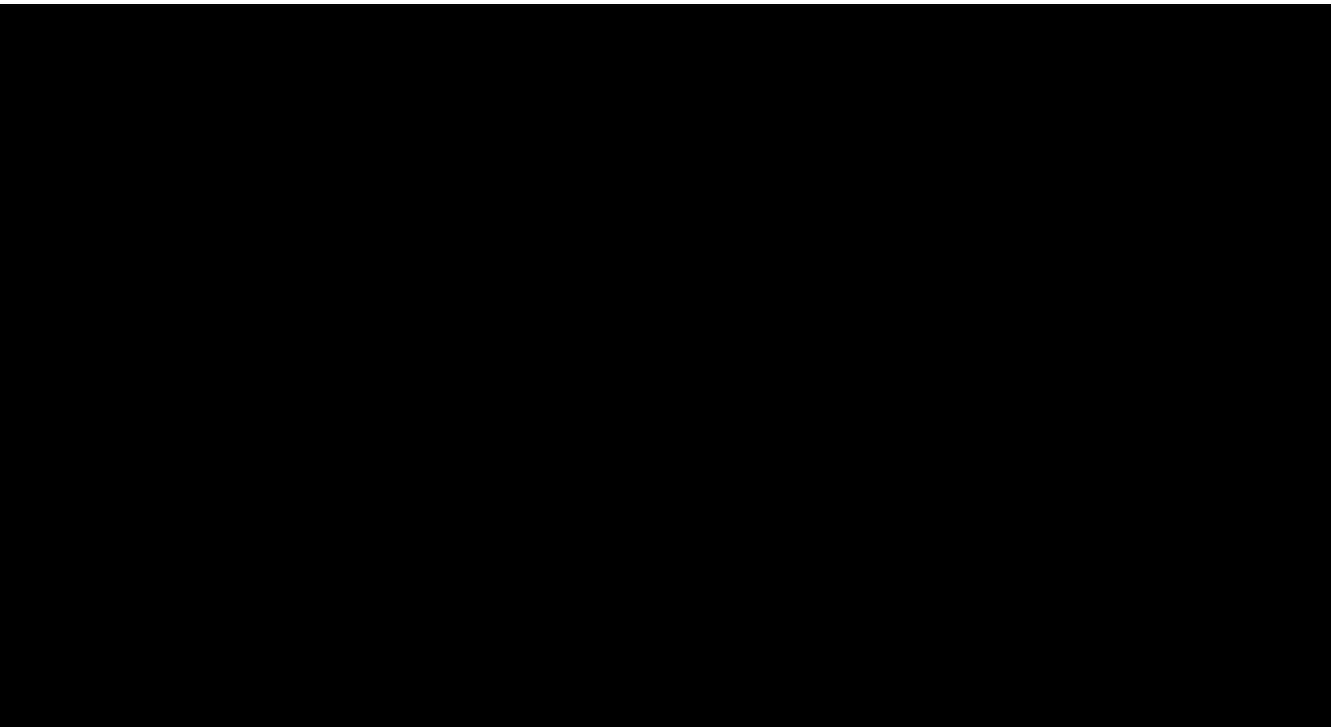


Figure 1. Installation Experience Flow Chart Phase 1

Install required software

Download and install the following software before continuing:

Note about KOTS installation: 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.

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: Mac or Linux .	1.47.3	Replicated Kubernetes Application Management

Tool	Version	Used for
Velero CLI	Latest	Backup and restore capability

AWS required software

¥ [AWS CLI](#) is installed.

GCP required software

¥ `gcloud` and `gsutil` are installed. You can set them up by installing Google Cloud SDK, which includes both, by referring to the [documentation](#).

For more information, see Velero's [supported providers](#) documentation.

Below, you will find instructions for creating a server 3.x backup on AWS and GCP.

S3 compatible storage required software

¥ [minio CLI](#) is installed and configured 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
3.3.x+	>= 1.16

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 will 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.

EKS

You can learn more about creating an Amazon EKS cluster [here](#). We recommend using `eksctl` to create your cluster, which will create a VPC and select the proper security groups for you.

1. [Install](#) and [configure](#) the AWS CLI for your AWS account.
2. Install `eksctl`.
3. Create your cluster by running the following (Cloud formation with `eksctl` and EKS can take more than 20 minutes to complete):

```
eksctl create cluster --name=circl-eci -server --nodes 4 --node-type m5.xlarge
```

4. Once the cluster has been created, you can use the following command to configure `kubectl` access:

```
eksctl utils write-kubeconfig --name circl-eci -server
```



You may see the following error `AWS STS Access - cannot get role ARN for current session: InvalidClientTokenID`. This means your AWS credentials are invalid, or your IAM user does not have permission to create an EKS cluster. Proper IAM permissions are necessary in order to use `eksctl`. See the AWS documentation regarding [IAM permissions](#).

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 enabled 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 entering running the following command:

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

3. Configure `kubectl` with your your credentials 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_DISPLAY_NAME> --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_GOOGL_CLOUD_PROJECT_ID>.iam.gserviceaccount.com
```

Create a new GitHub OAuth app



If GitHub Enterprise and CircleCI server are not on the same domain then images will fail to load.

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 will be 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 will need these values when you configure CircleCI server.

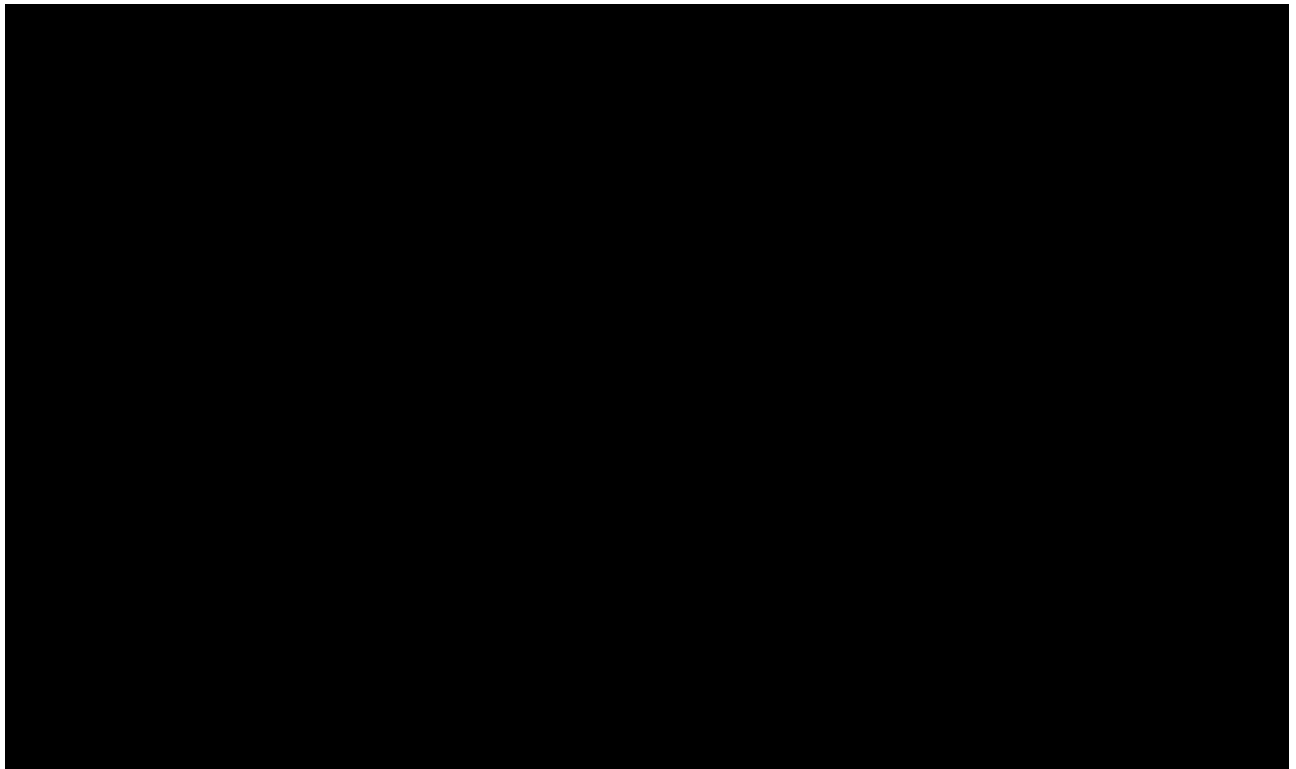


Figure 3. Client ID and Secret



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

Frontend TLS certificates

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 [Let's Encrypt](#) certificate authority, for example, can issue a free certificate using their [certbot](#) tool. In the sections below we cover using Google Cloud DNS and AWS Route53.

Google Cloud DNS

1. If you host your DNS on Google Cloud, you will 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>"
```


AWS Route53

1. If you are using AWS Route53 for DNS, you will need the certbot-route53 plugin installed. You can install the plugin with the following command:

```
pip3 install certbot-dns-route53
```

2. Then execute this example to create a private key and certificate (including intermediate certificates) locally in `/etc/letsencrypt/live/<CI_RCLECI_SERVER_DOMAIN>`:

```
certbot certonly --dns-route53 -d "<CI_RCLECI_SERVER_DOMAIN>" -d "app.<CI_RCLECI_SERVER_DOMAIN>"
```

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

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

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

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



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`

Encryption/signing keys

These keysets are used to encrypt and sign artifacts generated by CircleCI. You will 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, run the following command:

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

Encryption signing key

To generate, run the following command:

```
docker run ci rcl eci /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. For object storage providers that do not support S3 API, such as Azure blob storage, we recommend using Minio Gateway.

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. Ensure the bucket name you provide exists in your chosen object storage provider before proceeding.



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 `.ci rcl eci /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 an S3 storage bucket

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

¥ Storage Bucket Name - The bucket name to be used for server.

¥ Access Key ID - Access Key ID for S3 bucket access.

¥ Secret Key - Secret Key for S3 bucket access.

¥ AWS S3 Region - AWS region of bucket if your provider is AWS. You will either have an AWS region or S3 Endpoint depending on your specific setup.

¥ S3 Endpoint - API endpoint of S3 storage provider, when your storage provider is not Amazon S3.

Step 1: Create AWS S3 Bucket

```
aws s3api create-bucket \
  --bucket <YOUR_BUCKET_NAME> \
  --region <YOUR_REGION> \
  --create-bucket-configuration LocationConstraint=<YOUR_REGION>
```



us-east-1 does not support a LocationConstraint. If your region is `us-east-1`, omit the bucket configuration

Step 2: Create an IAM user for CircleCI server

```
aws iam create-user --user-name circleci-server
```

Step 3: Create a policy document *policy.json*

If using IAM Roles for Service Accounts (IRSA) for authentication, use the following content

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:PutAnalyticsConfiguration",
        "s3:GetObjectVersionTagging",
        "s3:CreateBucket",
        "s3:GetObjectAcl",
        "s3:GetBucketObjectLockConfiguration",
        "s3:DeleteBucketWebsite",
        "s3:PutLifecycleConfiguration",
        "s3:GetObjectVersionAcl",
        "s3:PutObjectTagging",
        "s3:DeleteObject",
        "s3:DeleteObjectTagging",
        "s3:GetBucketPolicyStatus",
        "s3:GetObjectRetention",
        "s3:GetBucketWebsite",
        "s3:GetJobTagging",
        "s3:DeleteObjectVersionTagging",
        "s3:PutObjectLegalHold",
        "s3:GetObjectLegalHold",
        "s3:GetBucketNotification",
        "s3:PutBucketCORS",
        "s3:GetReplicationConfiguration",
        "s3:ListMultipartUploadParts",
        "s3:PutObject",
        "s3:GetObject",

```

```

Ê      "s3: PutBucketNoti fi cati on" ,
Ê      "s3: Descri beJob" ,
Ê      "s3: PutBucketLoggi ng" ,
Ê      "s3: GetAnal yti csConfi gurati on" ,
Ê      "s3: PutBucketObj ectLockConfi gurati on" ,
Ê      "s3: GetObj ectVersi onForRepl i cati on" ,
Ê      "s3: GetLi fecycl eConfi gurati on" ,
Ê      "s3: GetI nventoryConfi gurati on" ,
Ê      "s3: GetBucketTaggi ng" ,
Ê      "s3: PutAccel erateConfi gurati on" ,
Ê      "s3: Del eteObj ectVersi on" ,
Ê      "s3: GetBucketLoggi ng" ,
Ê      "s3: Li stBucketVersi ons" ,
Ê      "s3: Repl i cateTags" ,
Ê      "s3: RestoreObj ect" ,
Ê      "s3: Li stBucket" ,
Ê      "s3: GetAccel erateConfi gurati on" ,
Ê      "s3: GetBucketPol i cy" ,
Ê      "s3: PutEncrypti onConfi gurati on" ,
Ê      "s3: GetEncrypti onConfi gurati on" ,
Ê      "s3: GetObj ectVersi onTorrent" ,
Ê      "s3: AbortMul ti partUpl oad" ,
Ê      "s3: PutBucketTaggi ng" ,
Ê      "s3: GetBucketRequestPayment" ,
Ê      "s3: GetAccessPoi ntPol i cyStatus" ,
Ê      "s3: GetObj ectTaggi ng" ,
Ê      "s3: GetMetri csConfi gurati on" ,
Ê      "s3: PutBucketVersi oni ng" ,
Ê      "s3: GetBucketPubl i cAccessBl ock" ,
Ê      "s3: Li stBucketMul ti partUpl oads" ,
Ê      "s3: PutMetri csConfi gurati on" ,
Ê      "s3: PutObj ectVersi onTaggi ng" ,
Ê      "s3: GetBucketVersi oni ng" ,
Ê      "s3: GetBucketAcl " ,
Ê      "s3: PutI nventoryConfi gurati on" ,
Ê      "s3: GetObj ectTorrent" ,
Ê      "s3: PutBucketWebsi te" ,
Ê      "s3: PutBucketRequestPayment" ,
Ê      "s3: PutObj ectRetenti on" ,
Ê      "s3: GetBucketCORS" ,
Ê      "s3: GetBucketLocati on" ,

```

```

    "s3: GetAccessPoi ntPol i cy",
    "s3: GetObj ectVersi on",
    "s3: GetAccessPoi nt",
    "s3: GetAccountPubl i cAccessBl ock",
    "s3: Li stAl l MyBuckets",
    "s3: Li stAccessPoi nts",
    "s3: Li stJobs"
  ],
  "Resource": [
    "arn: aws: s3: : : <YOUR_BUCKET_NAME>",
    "arn: aws: s3: : : <YOUR_BUCKET_NAME>/*"
  ]
}
{
  "Effect": "Al l ow",
  "Acti on": [
    "i am: GetRol e",
    "sts: AssumeRol e"
  ],
  "Resource": "<YOUR_OBJECT_STORAGE_ROLE>"
},
]
}

```

Otherwise, if using IAM keys for authentication, use the following content

```

{
  "Versi on": "2012-10-17",
  "Statement": [
    {
      "Effect": "Al l ow",
      "Acti on": [
        "s3: PutAnal yti csConfi gurati on",
        "s3: GetObj ectVersi onTaggi ng",
        "s3: CreateBucket",
        "s3: GetObj ectAcl ",
        "s3: GetBucketObj ectLockConfi gurati on",
        "s3: Del eteBucketWebsi te",
        "s3: PutLi fecycl eConfi gurati on",
        "s3: GetObj ectVersi onAcl ",
        "s3: PutObj ectTaggi ng",

```

```

Ê      "s3: DeleteObject",
Ê      "s3: DeleteObjectTagging",
Ê      "s3: GetBucketPolicyStatus",
Ê      "s3: GetObjectRetention",
Ê      "s3: GetBucketWebsite",
Ê      "s3: GetJobTagging",
Ê      "s3: DeleteObjectVersionTagging",
Ê      "s3: PutObjectLegalHold",
Ê      "s3: GetObjectLegalHold",
Ê      "s3: GetBucketNotification",
Ê      "s3: PutBucketCORS",
Ê      "s3: GetReplicationConfiguration",
Ê      "s3: ListMultipartUploadParts",
Ê      "s3: PutObject",
Ê      "s3: GetObject",
Ê      "s3: PutBucketNotification",
Ê      "s3: DescribeJob",
Ê      "s3: PutBucketLogging",
Ê      "s3: GetAnalyticsConfiguration",
Ê      "s3: PutBucketObjectLockConfiguration",
Ê      "s3: GetObjectVersionForReplication",
Ê      "s3: GetLifecycleConfiguration",
Ê      "s3: GetInventoryConfiguration",
Ê      "s3: GetBucketTagging",
Ê      "s3: PutAccelerateConfiguration",
Ê      "s3: DeleteObjectVersion",
Ê      "s3: GetBucketLogging",
Ê      "s3: ListBucketVersions",
Ê      "s3: ReplicateTags",
Ê      "s3: RestoreObject",
Ê      "s3: ListBucket",
Ê      "s3: GetAccelerateConfiguration",
Ê      "s3: GetBucketPolicy",
Ê      "s3: PutEncryptionConfiguration",
Ê      "s3: GetEncryptionConfiguration",
Ê      "s3: GetObjectVersionTorrent",
Ê      "s3: AbortMultipartUpload",
Ê      "s3: PutBucketTagging",
Ê      "s3: GetBucketRequestPayment",
Ê      "s3: GetAccessPointPolicyStatus",
Ê      "s3: GetObjectTagging",

```

```

    "s3: GetMetricsConfiguration",
    "s3: PutBucketVersioning",
    "s3: GetBucketPublicAccessBlock",
    "s3: ListBucketMultipartUploads",
    "s3: PutMetricsConfiguration",
    "s3: PutObjectVersionTagging",
    "s3: GetBucketVersioning",
    "s3: GetBucketAcl",
    "s3: PutInventoryConfiguration",
    "s3: GetObjectTorrent",
    "s3: PutBucketWebsite",
    "s3: PutBucketRequestPayment",
    "s3: PutObjectRetention",
    "s3: GetBucketCORS",
    "s3: GetBucketLocation",
    "s3: GetAccessPointPolicy",
    "s3: GetObjectVersion",
    "s3: GetAccessPoint",
    "s3: GetAccountPublicAccessBlock",
    "s3: ListAllMyBuckets",
    "s3: ListAccessPoints",
    "s3: ListJobs"
  ],
  "Resource": [
    "arn:aws:s3:::<YOUR_BUCKET_NAME>",
    "arn:aws:s3:::<YOUR_BUCKET_NAME>/*"
  ]
}
]
}

```

Step 4: Attach policy to user

```

aws iam put-user-policy \
  --user-name circleci-server \
  --policy-name circleci-server \
  --policy-document file://policy.json

```

Step 5: Create Access Key for user circleci-server



You will need this when you configure your server installation later.

```
aws iam create-access-key --user-name ci rcl eci -server
```

The result should look like this:

```
{
  "AccessKey": {
    "UserName": "ci rcl eci -server",
    "Status": "Active",
    "CreateDate": "2017-07-31T22: 24: 41. 576Z",
    "SecretAccessKey": <AWS_SECRET_ACCESS_KEY>,
    "AccessKeyId": <AWS_ACCESS_KEY_ID>
  }
}
```

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.

¥ Service Account JSON - 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 in the IAM console:



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

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

Step 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://ci rcl eci -server-bucket
```


Step 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)'
```

Step 3: Grant Permissions to Service Account

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

Step 4: JSON Key File

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>
```

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.



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

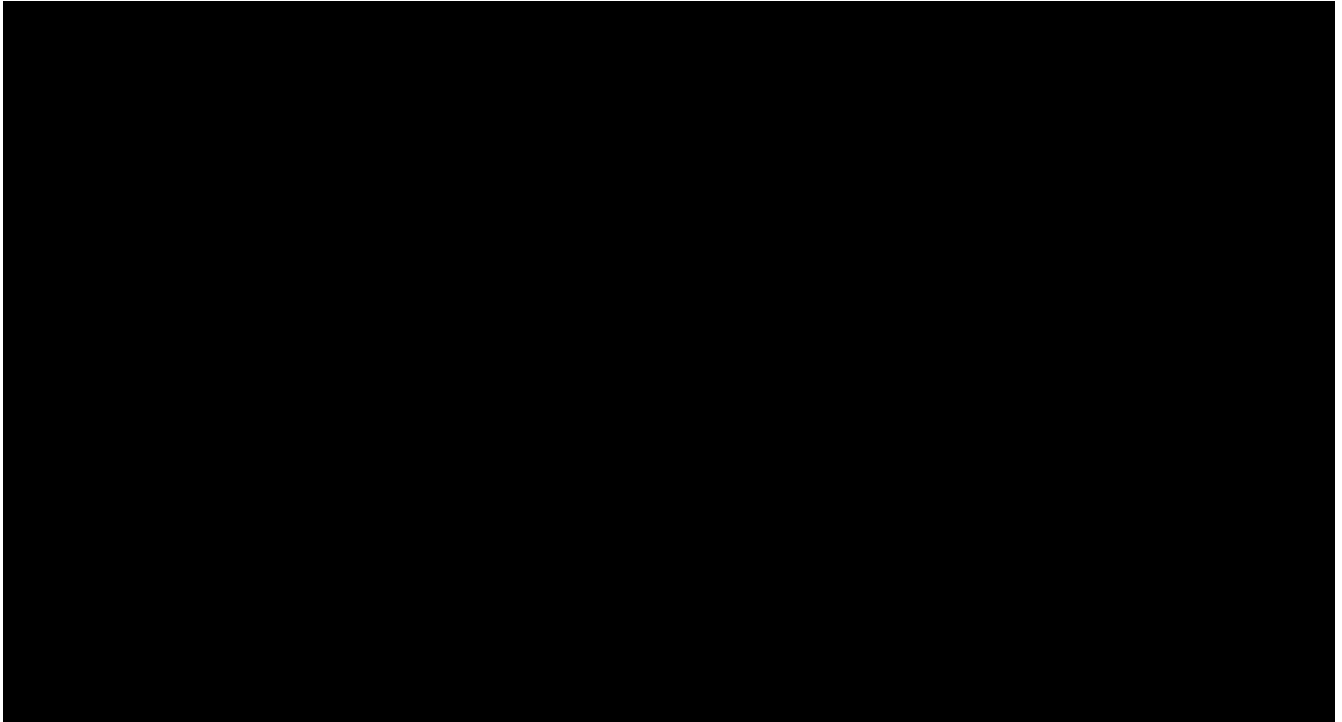


Figure 4. Installation Experience Flow Chart Phase 2

Phase 2: Core services installation

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 version (1.47.3) by running the command:

```
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>.

From the terminal run (if you are installing behind a proxy see [\[Installing behind HTTP Proxy\]](#)):

```
kubectl kots install ci rcl eci -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 KOTS admin console, usually `http://localhost:8800`.

!

If you need to get back to the KOTS admin console at a later date you can 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

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

```
kubectl kots install ci rcl eci -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. You can either supply a private key and certificate, or check the box to allow Let's Encrypt to automatically request and manage certificates for you. You can also disable TLS termination at this point, but the system will still need to be accessed over HTTPS.



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

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

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

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

Artifact and Encryption Signing Settings

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

Complete the following fields:

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

Github Settings

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

- ¥ 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.

Object Storage Settings

You created your Object Storage Bucket and Keys in the prerequisite steps. Use the data to complete the following:

S3 Compatible

You should have created your S3 Compatible bucket and optional IAM account during the prerequisite steps.

- ¥ Storage Bucket Name (required) - The bucket used for server.
- ¥ AWS S3 Region (optional) - AWS region of bucket if your provider is AWS. S3 Endpoint is ignored if this option is set.
- ¥ 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.
- ¥ Storage Object Expiry (required) - Number of days to retain your test results and artifacts. Set to 0 to disable and retain objects indefinitely.

Authentication

One of the following is required. Either select IAM keys and provide:

- ¥ Access Key ID (required) - Access Key ID for S3 bucket access.
- ¥ Secret Key (required) - Secret Key for S3 bucket access.

Or select IAM role and provide:

¥ Role ARN - [Role ARN for Service Accounts](#) (Amazon Resource Name) for S3 bucket access.

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

¥ Service Account JSON (required) - A JSON format key of the Service Account to use for bucket access.

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 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, it is time to deploy. The deployment will install the core services and provide you with an IP address for the Traefik load balancer. That IP address will be critical in setting up a DNS record and completing the first phase of the installation.



In this first stage we skipped a lot of fields in the config. We will revisit those in the next stages of installation.

Create DNS entry

Create a DNS entry for your Traefik load balancer, for example, `ci rcl eci . your . domai n . com` and `app . ci rcl eci . your . domai n . 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 will need the IP address or, if using AWS, the DNS name of the Traefik load balancer. You can find this with the following command:

```
kubectl get service ci rcl eci -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 Traefik 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](#).



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

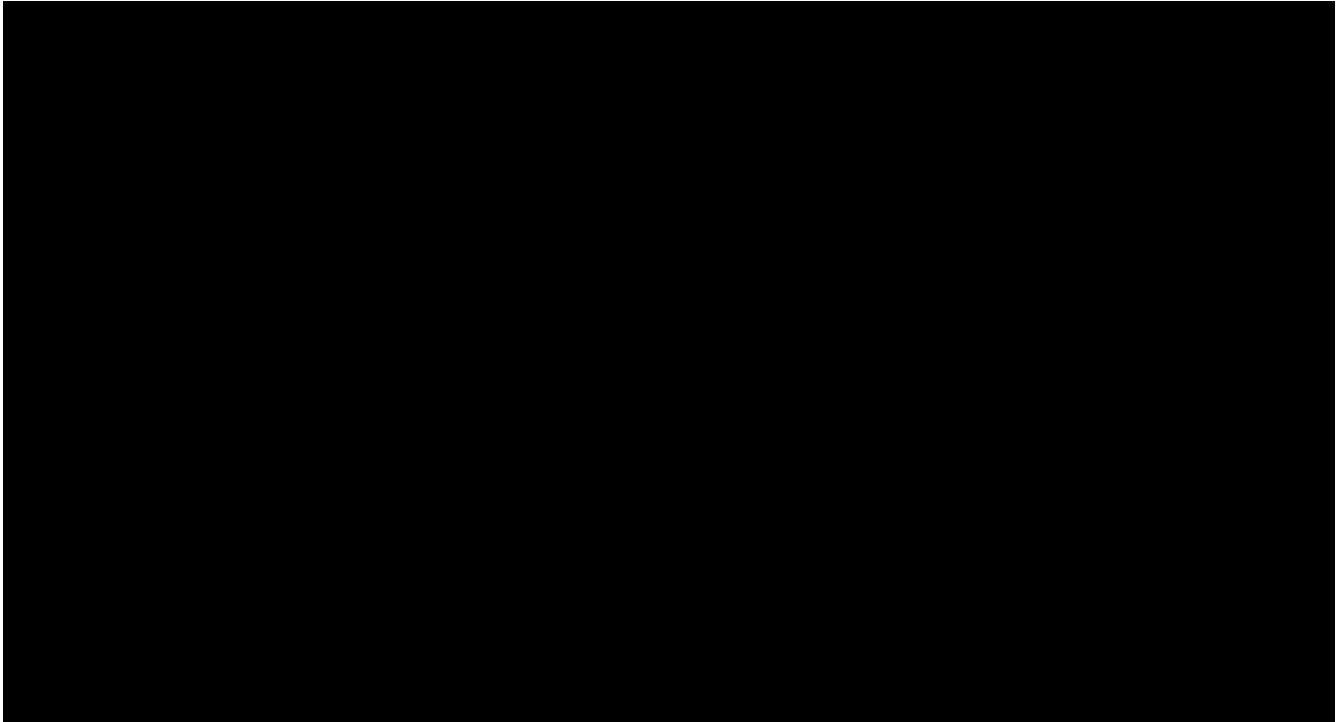


Figure 6. Installation Experience Flow Chart Phase 3

Phase 3: Execution environment installation

Output Processor

Overview

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. Get to the KOTS admin console by running the following, substituting your namespace: `kubectl kots admin-console -n <YOUR_CIRCLECI_NAMESPACE>`

Locate and enter the following in Settings:

1. Output Processor Load Balancer (required) - The following command will provide 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

Overview

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 the Nomad Clients installation.

Once completed, you will be able to update your CircleCI server configuration so your Nomad control plane is able to communicate with your Nomad Clients.

Cluster Creation with Terraform

CircleCI curates Terraform modules to help install Nomad clients in your cloud provider of choice. 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 time, see the [Nomad Autoscaler](#) section of this guide, as some of the requirements can be included in this Terraform setup.

AWS

You will need some information about your cluster and server installation to complete the required fields for the terraform configuration file (`main.tf`). A full example, as well as a full list of variables, can be found [here](#).

- ¥ `Server_endpoint` - You will need to know the Nomad Server endpoint, which is the external IP address of the nomad-server-external Loadbalancer. You can get this information with the following command:

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

- ¥ Subnet ID (`subnet`), VPC ID (`vpcId`), and DNS server (`dns_server`) of your cluster. Run the following command to get the cluster VPC ID (`vpcId`), CIDR block (`serviceIpv4Cidr`), and subnets (`subnetIds`):

```
aws eks describe-cluster --name=<YOUR_CLUSTER_NAME>
```

This will return something similar to the following:

```
{ ...
  "resourcesVpcConfig": {
    "subnetIds": [
      "subnet-033a9fb4be69",
      "subnet-04e89f9eef89",
      "subnet-02907d9f35dd",
      "subnet-0fbc63006c5f",
      "subnet-0d683b6f6ba8",
      "subnet-079d0ca04301"
    ],
    "clusterSecurityGroupId": "sg-022c1b544e574",
    "vpcId": "vpc-02fdfff4c",
    "endpointPublicAccess": true,
    "endpointPrivateAccess": false
  },
  "kubernetesNetworkConfig": {
    "serviceIpv4Cidr": "10.100.0.0/16"
  },
  ...
}
```

Then, using the VPCID you just found, run the following command to get the CIDR Block for your cluster. For AWS, the DNS Server is the third IP in your CIDR block (CidrBlock), for example your CIDR block might be 10.100.0.0/16 so the third IP would be 10.100.0.2.

```
aws ec2 describe-vpcs --filters Name=vpc-id,Values=<YOUR_VPCID>
```

This will return something like the following:

```
{ ...
  "CidrBlock": "192.168.0.0/16",
  "DhcpOptionsId": "dopt-9cff",
  "State": "available",
  "VpcId": "vpc-02fdfff4c"
  ...
}
```

Once you have filled in the appropriate information, you can deploy your Nomad clients by running the following command from within the directory of the `main.tf` file.

```
terraform init
```

```
terraform plan
```

```
terraform apply
```

After Terraform is done spinning up the Nomad client(s), it will output the certificates and keys needed for configuring the Nomad control plane in CircleCI server. Make sure to copy them somewhere safe. The apply process usually only takes a minute.

GCP

You will 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 will also need the following information:

- ¥ The GPC Project you want to run Nomad clients in.
- ¥ The GPC Zone you want to run Nomad clients in.
- ¥ The GPC Region you want to run Nomad clients in.
- ¥ The GPC Network you want to run Nomad clients in.
- ¥ The GPC 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"
}

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

module "nomad" {
  source = "git::https://github.com/CiurleCI-Public/server-terraform.git//nomad-gcp?ref=3.3.0"

  zone = var.zone
  region = var.region
  network = var.network
  subnetwork = var.subnetwork
  server_endpoint = var.server_endpoint
  machine_type = "n2-standard-8"

  unsafe_disable_mtls = true
  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 will output the certificates and key needed for configuring the Nomad control plane in CircleCI server. Make sure to 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 where it is located. You can enable this resource via KOTS, which will deploy the Nomad Autoscaler service along with your Nomad servers. Below we will go through how to set up Nomad Autoscaler for your provider.



The maximum and minimum Nomad client count will 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 the two don't compete.

If you do not require this service then feel free to jump to the Save config button to update your installation and redeploy server.

AWS

1. Create an IAM user or role and policy for Nomad Autoscaler. You may take one of the following approaches:

- ! Our [nomad module](#) creates an IAM user and outputs the keys if you set variable `nomad_auto_scaler = true`. You may reference the example in the link for more details. If you've already created the clients, you can update the variable and run `terraform apply`. The created user's access key and secret will be available in Terraform's output.
- ! You may also create a Nomad Autoscaler IAM user manually with the IAM policy below. Then you will need to generate an access and secret key for this user.
- ! You may create a [Role for Service Accounts](#) for Nomad Autoscaler and attach the following IAM policy:

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "VisualEditor0",
      "Effect": "Allow",
      "Action": [
        "autoscaling:CreateOrUpdateTags",
        "autoscaling:UpdateAutoScalingGroup",
        "autoscaling:TerminateInstanceAutoScalingGroup"
      ],
      "Resource": "<<Your Autoscaling Group ARN>>"
    },
    {
      "Sid": "VisualEditor1",
      "Effect": "Allow",
      "Action": [
        "autoscaling:DescribeScalingActivities",
        "autoscaling:DescribeAutoScalingGroups"
      ],
      "Resource": "*"
    }
  ]
}

```

2. In your KOTS admin console, set Nomad Autoscaler to `enabled`
3. Set Max Node Count* - This will overwrite what is currently set as the max for you ASG. It is recommended to keep this value and what was set in your Terraform as the same.
4. Set Min Node Count* - This will overwrite what is currently set as the max for you ASG. It is recommended to keep this value and what was set in your Terraform as the same.
5. Select cloud provider: `AWS EC2`
6. Add the region of the auto scaling group
7. You can chose one of the following:
 - a. Add the Nomad Autoscaler user's access key and secret key
 - b. Or, the Nomad Autoscaler role's ARN
8. Add the name of the autoscaling Group your Nomad clients were created in

GCP

1. Create a service account for Nomad Autoscaler

! Our [nomad module](#) creates a service account and outputs a file with the keys if you set variable `nomad_auto_scaler = true`. You may reference the examples in the link for more details. If you've already created the clients, you can simply update the variable and run `terraform apply`. The created user's key will be available in a file named `nomad-as-key.json`.

! You may also create a nomad gcp service account manually. The service account will need the role `compute.admin`.

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. JSON of GCP service account for Nomad Autoscaler

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. Get to 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`

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 will need to clone the repository. Depending on your Github setup you can do one of the following.

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 will 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.

EKS

1. Get the Information Needed to Create Security Groups

The following command will return your VPC ID (vpcId), CIDR Block (serviceIpv4Cidr), Cluster Security Group ID (clusterSecurityGroupId) and Cluster ARN (arn) values, which you will need throughout this section:

```
aws eks describe-cluster --name=<your-cluster-name>
```

2. Create a security group

Run the following commands to create a security group for VM service.

```
aws ec2 create-security-group --vpc-id "<YOUR_VPCID>" --description "CircleCI VM Service security group" --group-name "circleci-vm-service-sg"
```

This will output a GroupID to be used in the next steps:

```
{
  "GroupID": "sg-0cd93e7b30608b4fc"
}
```

3. Apply security group Nomad

Use the security group you just created and CIDR block values to apply the security group to the following:

```
aws ec2 authorize-security-group-ingress --group-id "<YOUR_GroupID>" --protocol tcp --port 22
--cidr "<YOUR_service_ipv4CIDR>"
```

```
aws ec2 authorize-security-group-ingress --group-id "<YOUR_GroupID>" --protocol tcp --port 2376
--cidr "<YOUR_service_ipv4CIDR>"
```



If you created your Nomad Clients in a different subnet from CircleCI server, you will need to rerun the above two commands with each subnet CIDR.

4. Apply the security group for SSH

Run the following command to apply the security group rules so users can SSH into their jobs:

```
aws ec2 authorize-security-group-ingress --group-id "<YOUR_GroupID>" --protocol tcp --port 54782
```

5. Create user

Create a new user with programmatic access:

```
aws iam create-user --user-name circleci-server-vm-service
```

Optionally, vm-service does support the use of a [service account role](#) in place of AWS keys. If you would prefer to use a role, follow these [instructions](#) using the policy in step 6 below. Once done, you may skip to step 9 which is enabling vm-service in KOTS.

1. Create policy

Create a `policy.json` file with the following content. You should fill in Cluster Security Group ID (`clusterSecurityGroupID`) and Cluster ARN (`arn`) below.

```
{
  "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/<YOUR_clusterSecurityGroupID>"
  ],
},
{
  "Action": "ec2: RunInstances",
  "Effect": "Allow",
  "Resource": "arn:aws:ec2:*:*:instance/*",
  "Condition": {
    "StringEquals": {
      "aws:RequestTag/ManagedBy": "ci rcl eci -vm-servi ce"
    }
  }
},
{
  "Action": [
    "ec2: CreateVolume"
  ],
  "Effect": "Allow",
  "Resource": [
    "arn:aws:ec2:*:*:volume/*"
  ],
  "Condition": {
    "StringEquals": {
      "aws:RequestTag/ManagedBy": "ci rcl eci -vm-servi ce"
    }
  }
},
{
  "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": "ci rcl eci -vm-service"
      }
    },
    {
      "Action": [
        "ec2:RunInstances",
        "ec2:StartInstances",
        "ec2:StopInstances",
        "ec2:TerminateInstances"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:ec2:*:*:subnet/*",
      "Condition": {
        "StringEquals": {
          "ec2:Vpc": "<YOUR_arn>"
        }
      }
    }
  ]
}

```

2. Attach policy to user

Once you have created the policy.json file attach it to an IAM policy and created user.

```

aws iam put-user-policy --user-name ci rcl eci -vm-service --policy-name ci rcl eci -vm-service
--policy-document file://policy.json

```

3. Create an access key and secret for the user

If you have not already, you will need an access key and secret for the ci rcl eci -vm-service user. You can create that by running the following command:

```

aws iam create-access-key --user-name ci rcl eci -vm-service

```

4. 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.

GKE

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

```
gcloud container clusters describe
```

This command will return something like the following, which will include network, region and other details that you will 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:
  É
```

1. Create firewall rules

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

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



You can find the Nomad clients CIDR based on the region by referring to the [table here](#) if you have used auto-mode.

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

```
gcloud compute firewall-rules create "ci-rcl-eci-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 ci-rcl-eci-server-vm --display-name "ci-rcl-eci-server-vm service account"
```



If you are deploying CircleCI server in a shared VCP, you will want to create this user in the project that you intend to have your VM jobs run.

3. Get the service account email address

```
gcloud iam service-accounts list --filter="displayName: ci-rcl-eci-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

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

```
gcloud iam service-accounts keys create ci-rcl-eci-server-vm-keyfile --iam-account <YOUR_SERVICE_ACCOUNT_EMAIL>
```

6. 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 reality checker project within your CircleCI installation and you should see the VM Service Jobs complete with green. At this point all tests should pass with green.

Runner

Overview

CircleCI runner does not require any additional server configuration. Server ships ready to work with runner. However, you do 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 will 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](#).



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

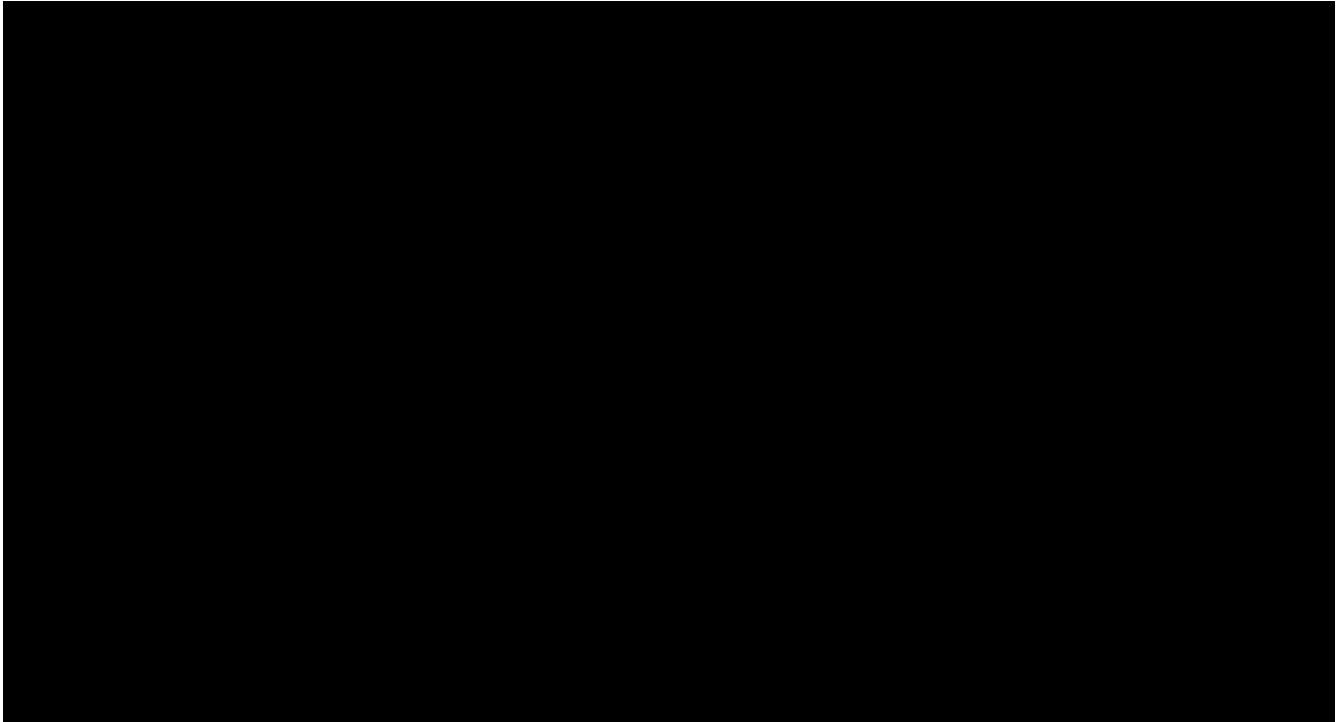


Figure 7. Installation Experience Flow Chart Phase 4

Phase 4: Post installation

Set up backup and restore

Backups of CircleCI server can be created quite easily through [KOTS](#). However, 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 AWS

The following steps will assume AWS as your provider.

These instructions were sourced from the Velero documentation [here](#).

Step 1 - Create an AWS S3 bucket

```

BUCKET=<YOUR_BUCKET>
REGI ON=<YOUR_REGI ON>
aws s3api create-bucket \
  Ê --bucket $BUCKET \
  Ê --regi on $REGI ON \
  Ê --create-bucket-confi gurati on Locati onConstrai nt=$REGI ON

```



us-east-1 does not support a [LocationConstraint](#). If your region is `us-east-1`, omit the bucket configuration.

Step 2 - Set up permissions for Velero

¥ Create an IAM user

```
aws iam create-user --user-name vel ero
```

¥ Attach policies to give user `vel ero` the necessary permissions:

```

cat > vel ero-pol i cy. j son <<EOF
{
  Ê "Versi on": "2012-10-17",
  Ê "Statement": [
  Ê   {
  Ê     "Effect": "Al l ow",
  Ê     "Acti on": [
  Ê       "ec2: Descri beVol umes",
  Ê       "ec2: Descri beSnapshots",
  Ê       "ec2: CreateTags",
  Ê       "ec2: CreateVol ume",
  Ê       "ec2: CreateSnapshot",
  Ê       "ec2: Del eteSnapshot"
  Ê     ],
  Ê     "Resource": "*"
  Ê   },
  Ê   {
  Ê     "Effect": "Al l ow",
  Ê     "Acti on": [
  Ê       "s3: Get0bj ect",
  Ê       "s3: Del ete0bj ect",
  Ê       "s3: Put0bj ect",

```

```

    "s3: AbortMul ti partUpl oad",
    "s3: Li stMul ti partUpl oadParts"
  ],
  "Resource": [
    "arn: aws: s3: :: ${BUCKET}/*"
  ]
},
{
  "Effect": "Al l ow",
  "Acti on": [
    "s3: Li stBucket"
  ],
  "Resource": [
    "arn: aws: s3: :: ${BUCKET}"
  ]
}
]
}
EOF

```

```

aws i am put-user-pol i cy \
  --user-name vel ero \
  --pol i cy-name vel ero \
  --pol i cy-document fi l e: //vel ero-pol i cy. j son

```

¥ Create an access key for user `vel ero`

```

aws i am create-access-key --user-name vel ero

```

The result should look like this:

```
{
  "AccessKey": {
    "UserName": "velero",
    "Status": "Active",
    "CreateDate": "2017-07-31T22:24:41.576Z",
    "SecretAccessKey": <AWS_SECRET_ACCESS_KEY>,
    "AccessKeyId": <AWS_ACCESS_KEY_ID>
  }
}
```

¥ Create a Velero-specific credentials file (eg: `./credentials-velero`) in your local directory, with the following contents:

```
[default]
aws_access_key_id=<AWS_ACCESS_KEY_ID>
aws_secret_access_key=<AWS_SECRET_ACCESS_KEY>
```

where the `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` placeholders are values returned from the `create-access-key` request in the previous step.

Step 3 - Install and start Velero

¥ Run the following `velero install` command. This will create a namespace called `velero` and install all the necessary resources to run Velero. Make sure that you pass the correct file name containing the AWS credentials that you have created in [Step 2](#).



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 $BUCKET \
  --backup-location-config region=$REGION \
  --snapshot-location-config region=$REGION \
  --secret-file ./credentials-velero \
  --use-restic \
  --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.

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, we will 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, we will 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, we will set a shell variable for your project ID. To do so, first 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, you might want to 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, we 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 have 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 will need the `iam.serviceAccounts.signBlob` role in addition to the permissions already specified above.

Step 3 - Install and start Velero

¥ Run one of the following `velero install` commands, depending on how you authorized the service account. This will create a namespace called `velero` and install 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 will assume you're 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, we add our 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 will create a namespace called `velero` and install 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:

```
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 before
- ¥ 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. Access the KOTS admin console. Get to the KOTS admin console by running the following, substituting your namespace: `kubectl kots admin-console -n <YOUR_CLUSTER_NAMESPACE>` and 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 (e.g., for Sendgrid use smtp.sendgrid.net).
- ¥ Username (required) - Username to authenticate to submission server. This is commonly the same as the user's e-mail 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.

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



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

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 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 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 [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 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 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 `ci rcl eci _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/ci rcl e/ci rcl eci -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 `ci rcl eci _export/ci rcl e-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:
`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) 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 a 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.

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 explains 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 explained 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 (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 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 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 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 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

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 reduce

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:

¥ [Kubernetes 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.