# External Services – CircleCI v2.17

May 13th, 2019

# Contents

# Adding External Services to CircleCI Server v2.17

This document describes how to configure the following external services for use with CircleCI Server v2.17 installed in your private datacenter or cloud:

- MongoDB v3.6.12-xenial
- PostgreSQL v9.5.8
- Hashicorp Vault v1.1.2
- Redis v4.0.14
- RabbitMQ v3.6
- Nomad Server v0.5.6

The steps in this document assume you have an existing CircleCI v2.17 Services machine and 2.0 Builders in use.

To configure your existing installation to use externalised databases, the content of the databases currently in use on the Services machine must be exported and then imported into the external databases. The steps to do this are explained in this document.

## Prerequisites

The following prerequisites must be met before beginning configuration:

- The CircleCI license must be updated to support external services.
- The Services VM must be running CircleCI v2.17
- Planned downtime of one hour minimum or 20 minutes per service you wish to externalize.
- New instances for a MongoDB or replica set must be ready to connect in advance of beginning this procedure.

## Setting up External MongoDB

CircleCI supports MongoDB version 3.6.6 and uses WiredTiger 3.2 as the backend storage engine.

### Best Practices for your Mongo Cluster

*This section applies to the configuration and sizing of the target MongoDB Cluster. If using a SaaS you can skip this.*

Consider the following when setting up your external database hosts:

- To maximize performance, use hosts with more memory for MongoDB. Ideally size server RAM to fit all data and indexes that will be accessed regularly. For example, installs of 1000 users require at least 64 GB of RAM.
- Use mounted EBS volumes for MongoDB data, for example, provision IOPs volumes at 10k-20k IOPs adjusted for your individual load. Configure each host the same way regardless of whether it is initially the primary or a secondary to avoid degradation when roles change.
- Consider using TLS for all communication with MongoDB, for example, between clients and MongoDB and between members of the MongoDB replica set. It is also possible to use an internally generated CA to create certificates and to deploy the root certificate to all clients and MongoDB instances.
- It is best practice to use x509 certificates as described in the MongoDB documentation.
- To increase throughput and guarantee data safety, it is best practice to run three MongoDB machines as a replica set. Following are the minimum sizes of each of three machines that will form the MongoDB replica set, depending on the number of daily active users:

| Number of daily active CircleCI users | Number of MongoDB replicas | CPU | RAM | Disk space NIC speed |
|---|---|---|---|---|
| <50 | 3 | 8 cores | 32GB | 100GB |
| 50-250 | 3 | 12 cores | 64GB | 200GB |
| 250-1000 | 3 | 16 cores | 128GB | 500GB |
| 1000-5000 | 3 | 20 cores | 256GB | 1TB |
| 5000+ | 3 | 24 cores | 512GB | 2TB |

The mongodb service user should have `readWrite` access to the following databases: `circle_ghe`, `build_state_dev_ghe`, and `containers_dev_ghe`, which the service machine will attempt to create on first connection.

### Service Box Setup

Configure an external MongoDB server for use with CircleCI Server by completing the following steps:

1. Modify connection strings in the `/etc/circle-installation-customizations` file *on the Services machine* as follows:

```
# Configure the single endpoint
MONGO_BASE_URI='mongodb://circle:<password>@<hostname>:27017'
# OR for a replica set
# MONGO_BASE_URI='mongodb://circle:<password>@<hostname1>:<port>,
<hostname2>:<port>,<hostname3>:<port>
# When using replica sets, add query parameter `&replicaSet=name` to each URI
export CIRCLE_SECRETS_MONGODB_MAIN_URI="$MONGO_BASE_URI/circle_ghe?ssl=true&authSource=admin"
export CIRCLE_SECRETS_MONGODB_ACTION_LOGS_URI="$MONGO_BASE_URI/circle_ghe?ssl=true&authSource=admin"
export CIRCLE_SECRETS_MONGODB_BUILD_STATE_URI="$MONGO_BASE_URI/build_state_dev_ghe?ssl=true&authSource=admin"
export CIRCLE_SECRETS_MONGODB_CONTAINERS_URI="$MONGO_BASE_URI/containers_dev_ghe?ssl=true&authSource=admin"
export CIRCLE_SECRETS_MONGODB_REMOTE_CONTAINERS_URI="$MONGO_BASE_URI/remote_containers_dev_ghe?ssl=true&authSource=admin"
```

2. If configuring other external services, proceed to those sections. Otherwise you may now start CircleCI via the Management Console dashboard (`https://<your-circleci-hostname>.com:8800`).
3. Once started, you can check the `frontend` container logs to verify it is connected to your MongoDB server(s):

```
sudo docker logs frontend | grep <mongo_hostname>

Mar 27 14:55:16.780:+0000 INFO org.mongodb.driver.connection Opened connection \
[connectionId{localValue:36, serverValue:3757}] to YOURCLUSTER.mongodb.net:27017
```

## Backing up MongoDB

We recommend you regularly back up the MongoDB data. Here are some recommended best practices:

- Perform daily backups
- Keep at least 30 days of backups
- Use encrypted storage backups
- Perform a backup before each upgrade of CircleCI Server.

The best way to guarantee data security is to run at least three MongoDB replicas at any given time. Running less than 3 replicas may lead to undefined results.

## Disaster Recovery

The disaster recovery mechanism recommended by MongoDB is to run `mongodump` against any of the **secondary** instances to create a full dump of MongoDB data, and then use `mongorestore` to restore the previously backed up data. However, it has been observed that running `mongodump` heavily impacts disk I/O on one of the secondary MongoDB instances, which negatively impacted the performance of the CircleCI application as a whole.

Therefore, the backup mechanism for MongoDB that is best practice with CircleCI is to create snapshots of the whole disk of one of the secondary machines. Complete the following steps to preform disaster recovery:

1. On one of the secondary machines, shut down the MongoDB server by running `db.shutdownServer()`.
2. Make a snapshot of the disk on which the MongoDB data is located. If running on VMs you can use your hypervisor's built-in snapshotting capabilities. If running on bare metal servers you can use dd to write a copy of the disk's contents to a backup location.
3. Once the snapshot creation is finished and the snapshot is copied over to the backup location, start the MongoDB server on the secondary machine that you used to get a snapshot.

## Setting up External PostgreSQL

### Best Practices for your PostgreSQL

Use `circle` as the user and default database name.

Consider running at least two or more PostgreSQL replicas to enable recovery from primary failure and for backups. Following are the recommended specifications of the PostgreSQL machines:

| Number of daily active CircleCI users | Number of PostgreSQL replicas | CPU | RAM | Disk space | NIC speed |
|---|---|---|---|---|---|
| <50 | 2 | 8 cores | 16GB | 100GB | 1Gbps |
| 50-250 | 2 | 8 cores | 16GB | 200GB | 1Gbps |
| 250-1000 | 3 | 8 cores | 32GB | 500GB | 10Gbps |
| 1000-5000 | 3 | 8 cores | 128GB | 1TB | 10Gbps |
| 5000+ | 3 | 8 cores | 128GB | 1TB | 10Gbps |

### Services Configuration Steps

This section describes how to configure external PostgreSQL servers for a CircleCI Server v2.17 installation.

1. Using the text editor of your choice add the following lines to the `/etc/circleconfig/shared/postgresql` file (creating it if not present):

```
export POSTGRES_HOST="<hostname>"
export POSTGRES_PORT="5432"
export POSTGRES_PASSWORD="<password>"
export POSTGRES_USER="circle"   # We advise not changing the username
```

2. Create required databases on your PostgreSQL instance. This must be completed before CircleCI tries to start.

```
source /etc/circleconfig/shared/postgresql
apt install postgresql-client
psql postgres://${POSTGRES_USER}:${POSTGRES_PASSWORD}@${POSTGRES_HOST}:${POSTGRES_PORT}

CREATE DATABASE circle;
\connect circle
 CREATE EXTENSION IF NOT EXISTS "uuid-ossp";
 CREATE EXTENSION IF NOT EXISTS "pgcrypto";
CREATE DATABASE contexts;
\connect contexts
 CREATE EXTENSION IF NOT EXISTS "uuid-ossp";
CREATE DATABASE conductor_production;
\connect conductor_production
 CREATE EXTENSION IF NOT EXISTS "uuid-ossp";
CREATE DATABASE contexts_service_production;
\connect contexts_service_production
 CREATE EXTENSION IF NOT EXISTS "uuid-ossp";
CREATE DATABASE cron_service_production;
\connect cron_service_production
```

```
 CREATE EXTENSION IF NOT EXISTS "uuid-ossp";
CREATE DATABASE domain;
\connect domain
 CREATE EXTENSION IF NOT EXISTS "uuid-ossp";
CREATE DATABASE federations;
\connect federations
 CREATE EXTENSION IF NOT EXISTS "uuid-ossp";
CREATE DATABASE permissions;
\connect permissions
 CREATE EXTENSION IF NOT EXISTS "uuid-ossp";
CREATE DATABASE vms;
\connect vms
 CREATE EXTENSION IF NOT EXISTS "uuid-ossp";
```

CTRL+D or `\q` will exit.

3. If externalizing additional services, please move on to those sections. Otherwise move on to step 3 to start CircleCI Server

4. Start the Circle app from the Management Console dashboard (`https://<your-circleci-hostname>.com:8800`).

   After CircleCI Server has started, you can check that it is using the configured Postgres server correctly by running the following command:

   ```
   sudo docker logs $(sudo docker ps -q --filter "name=frontend") 2>&1 | grep "POSTGRES_HOST" | tail -n 1
   ```

   The result of this command should match the value for `"<hostname>"` as defined in the `/etc/circleconfig/shared/postgresql` file.

   You should also check to ensure no `org.postgresql.util.PSQLException` are printed in the frontend container.

## Backing up PostgreSQL

It is best practice to run `pg_dumpall` to write a copy of all the data stored in PostgreSQL to a single file. Please refer to the PostgreSQL documentation for specific instructions on using the `pg_dumpall` command.

The `pg_dumpall` command produces a single script file that can be used to restore the databases. If you need to restore the databases from backups at any point, please use the command:

```
psql -f db.out postgres
```

We strongly recommend the following:

- taking daily backups
- keeping at least 30 days of backup
- using encrypted storage for backups as databases might contain sensitive information
- performing a backup before each upgrade of CircleCI Server.

## Configuring External Vault

This section describes how to configure external vault for a CircleCI Server installation.

### Vault Best Practices

- Always use TLS
- Use an HA storage provider (i.e Consul or S3)

### Prerequisites

A running Vault Cluster with: - The "Root Token" created at init - Transit secrets engine enabled

**Note:** See the Troubleshooting section for example config.

### Creating the Policy and Client Token for CircleCI's use

This section provides the curl commands required to create policy and client token using the root token created at init. You may also use the Vault CLI depending on your level of understanding.

**Note: If the remote Vault provider is using TLS, as suggested in our best practices, add the appropriate `--ca-cert=/path/to/cert` argument to curl commands**

1. Create a context-service vault key:

```
curl --request POST \
         --header "X-Vault-Token: <vault root token>" \
         --header "Content-Type: application/json" \
         --silent --show-error --write-out "%{http_code}\n" \
         <vault-url>/v1/transit/keys/contexts-service
```

You should get a 204 response.

2. Create context-service policy:

```
curl --request POST  \
         --header "X-Vault-Token:<vault_root_token>"  \
         --header "Content-Type: application/json"  \
      --data '{"policy":"path \"transit/encrypt/*\" {\n capabilities = [\"update\"]}\npath \"transit/decrypt/*\" {\n capabi
         --fail --silent --show-error --write-out "%{http_code}\n" \
         <vault-url>/v1/sys/policy/contexts_service_policy
```

You should get a 204 response.

3. Create a *client* token:

```
curl --request POST \
         --header "X-Vault-Token:<vault_root_token>" \
         --header "Content-Type: application/json" \
         --data '{"policies":["default", "contexts_service_policy"],"period":"168h"}' \
         --fail --silent --show-error \
         --cacert <cert-file> \
         <vault-url>/v1/auth/token/create
```

This step will return a JSON object that has the `client_token` key. That token key is what you need to use for the `VAULT__TOKEN` value in the following procedure.

## Configuring an External Connection to Vault

1. Create a `/etc/circleconfig/contexts-service` directory in the Service VM if it does not exist:

   ```
   mkdir -p /etc/circleconfig/contexts-service
   ```

2. Add the following content to the `/etc/circleconfig/contexts-service/customizations` file. If you followed the directions above, the last line would be named `transit`.

   ```
   export VAULT__SCHEME="https"
   export VAULT__HOST="<vault-hostname>"
   export VAULT__PORT="<vault-port>"
   export VAULT__URL=$VAULT__SCHEME://$VAULT__HOST:$VAULT__PORT
   export VAULT__TOKEN="<vault-token>"
   export VAULT__MOUNT__TRANSIT="<vault-transit-mount>"
   ```

3. Restart CircleCI by going to the Managment Console and clicking Stop Now, waiting for it to stop, then clicking Start.

4. After CircleCI has successfully restarted, complete the following verification:

   SSH in to the Services VM to check the docker log for context-service, to confirm externalized vault has been successfully connected. You should see the following message:

   ```
   INFO  circleci.backplane.secrets.backend.vault Successfully retrieved current auth token;
   ```

## Setting up External Redis

1. Create the configuration file by running the command:

   ```
   touch /etc/circleconfig/shared/redis
   ```

2. Using the text editor of your choice add the following lines to the newly created file:

   ```
   export REDIS_HOST="<hostname>"
   export REDIS_PORT=<port>
   ```

**Note:** Authentication is not currently tested nor supported.

Proper installation can be verified by CircleCI Server starting up without error, and by following these steps:

1. Check domain-service logs for signs of redis hostname:

   ```
   sudo docker logs -f domain-service | grep "<host>:<port>"
   ```

   You should see the following in the output every minute or so: `Reconnected to <host>:<port>`.

2. Check the `picard-output-processor` container by executing the following command after starting a bash session in the container:

   ```
   sudo docker exec -it picard-output-processor /bin/bash
   source /.circlerc
   echo $OUTPUT_PROCESSOR_SIGNALS_REDIS
   redis://<host>:<port>/10
   ```

3. Finally check `slanger` container logs. The final log of the Redis environment variables should match your external Redis server:

   ```
   sudo docker exec -it slanger /bin/bash
   source /.circlerc
   echo $REDIS_PORT_6379_TCP_ADDR
   echo $REDIS_PORT_6379_TCP_PORT
   ```

### Slanger Requirements

CircleCI Server includes a replacement for Pusher called Slanger. While you do not need to externalize Slanger itself, it is highly recommended that you provide another externalized Redis for use by Slanger exclusively. You would follow the instructions above, but with a few changes.

1. Create a configuration file on the Services box:

   ```
   touch /etc/circleconfig/slanger/customizations
   ```

2. Add the exported environment variables used above to `/etc/circleconfig/slanger/customizations` to use a separate Redis server:

   ```
   export REDIS_HOST=<redis-host-identifier>
   REDIS_HOST=<redis-host-identifier>
   export REDIS_PORT=6379
   REDIS_PORT=6379
   ```

## Setting up External RabbitMQ

1. Create the configuration file by running the command:

   ```
   touch /etc/circleconfig/shared/rabbitmq
   ```

2. Using the text editor of your choice, add the following lines to the newly created file:

   ```
   export RABBITMQ_HOST="<hostname>"
   export RABBITMQ_PORT=<port>
   export RABBITMQ_USERNAME="<username>"
   export RABBITMQ_PASSWORD="<password>"
   ```

3. Verify proper installation by CircleCI starting up without error.

4. Use SSH to log in to the Services box and verify external RabbitMQ connectivity by searching the following logs:

   ```
   $ sudo docker logs workflows-conductor | grep <hostname>

   Configuring workflow-messages queue on host <rabbitmq-host-identifier>
   Configuring workflow-notifications queue on host <rabbitmq-host-identifier>

   $ sudo docker logs frontend | grep <hostname>

   Configuring usage-queue-events queue on host <rabbitmq-host-identifier>
   Configuring exchange picard-build-signals on host <rabbitmq-host-identifier>
   ```

## Setting up an External Nomad Server

This section describes the steps to configure your CircleCI instance to talk to an existing Nomad Server instance.

### Configuring your Services instance

With a Nomad Server running you can instruct CircleCI to connect to it. CircleCI supports Nomad Server v0.5.6.

1. Create the following file on your CircleCI Services box:

   `/etc/circleconfig/shared/nomad`

2. Edit the file and add the following line, specifying the host and port for your Nomad Server:

   `export NOMAD_API_ENDPOINT="https://<NOMAD_HOST>:<NOMAD_API_PORT>"`

   **Note:** Confirm the services box can reach the API port (usually :4646) before proceeding. You may need to update security groups or firewalls.

**Note:** If specifying https above, ensure you have configured your Nomad Server cluster with SSL certificates.

3. Start your CircleCI instance via the Management Console (`https://<your-circleci-hostname>.com:8800`).

### Nomad Clients

You will have to configure your nomad client configuration to point to the external Nomad Server. This procedure assumes you are using the Nomad client ASG created with the `enterprise-setup` script.

#### AWS Installations

Our provided AWS installations uses Auto Scaling Groups and Launch Configurations to automatically spawn Nomad clients based on pre-defined configuration. We'll need to update this to point to your external Nomad server.

1. Copy existing Launch Config created by your Terraform installation
2. Update the `user data`, either providing the new Nomad Server IP to point to the new server rather than the services box. Under the section headed *Creating config.hcl*, find and update the following lines, updating the `servers` as a list of Nomad Servers in your external cluster:

```
client {
    enabled = true
    # Expecting to have DNS record for nomad server(s)
    servers = ["10.0.1.150:4647"]
    node_class = "linux-64bit"
    options = {"driver.raw_exec.enable" = "1"}
}
```

3. Edit the existing ASG, point to your updated Launch Configuration using the new Nomad Server IP and ports.
4. Terminate any existing Nomad clients, this will force the ASG to spawn new Nomad clients with the proper config.

### Static Installations

Your exact procedure will vary by infrastructure. Be sure that your Nomad clients use the new IP of your Nomad server every time they start.

1. Update the startup/provisioning scripts on your Nomad clients as described in step #2 above.
2. Ensure this is applied across the fleet of Nomad clients currently installed.

## Nomad Server Development Installation

For testing purposes you can setup a single Nomad Server.

```
export NOMAD_VERSION=0.5.6
wget https://releases.hashicorp.com/nomad/${NOMAD_VERSION}/nomad_${NOMAD_VERSION}_linux_amd64.zip
unzip nomad_${NOMAD_VERSION}_linux_amd64.zip
sudo ./nomad agent -config nomad.hcl
```

**NOTE: this uses file storage on a single instance and is may risk loss of jobs. Please run Nomad Server in a 3 or 5 instance cluster for fault tolerance**

## Troubleshooting

### Mongo

#### General Troubleshooting

If your CircleCI Server instance is not starting up after configuring externalised databases, please check the URIs for MongoDB or PostgreSQL for correctness. Some examples of common issues related to the database connection URIs include:

- A network issue between one of your database server(s) and the Services host
- A malformed URI
- Invalid credentials to access the database(s)
- Improper configuration settings for the database
- The database refusing connections from outside sources and restricted to localhost

Additionally, run the following on the Services machine to display the main CircleCI app logs which may contain information related to the issue:

```
docker logs -f frontend
```

#### Timeouts

No max idle time is set in the MongoDB client by default. If the MongoDB server is behind a proxy with such a timeout, you may need to specify `maxIdleTimeMS`. Refer to the MongoDB documentation for available parameters.

#### SSL Errors

Verify that you have set up certificates correctly by running the following command on the services box:

```
openssl s_client -connect <mongo_hostname>:27017
```

If you aren't returned a 0 exit code then it is likely CircleCI will not be able to connect. A copy of the CA certificate must be saved on Services machine in `/usr/local/share/ca-certificates` with file extension `.crt`.

To bypass potential connection issues caused by reverse DNS lookups add the following line to the `/etc/environment` file on the Services box:

```
export JVM_OPTS='-Djdk.tls.trustNameService=true'
```

*This instructs the CircleCI services to trust the reverseDNS lookup resolving your mongo hostnames. See JDK Release Notes for more information.*

## PostgreSQL

### Connecting to the Postgres instance

You can manually connect using a configuration as follows:

```
psql  --host=<postgres_hostname>  --port=5432  --username=<your_user>  --password  --
dbname=<your_db_name>
```

### FATAL: database "user" does not exist

```
FATAL: database "user" does not exist
```

By default PostgreSQL installations create a database that matches the root user's name. To fix this issue, create a DB that matches the username. This may be referred to as `circle` in the included documentation, but can be a custom user as well.

```
# Source the variables we set previouslt
source /etc/circleconfig/shared/postgresql
# create a DB that matches the user.
/home/ubuntu# createdb -h $POSTGRES_HOST -U $POSTGRES_USER --password $POSTGRES_USER
```

**After creating the database with the command below, proceed to steps on verify full list of databases required.**

## Confirming Tables

You may run the `\list` command to ensure that the databases that are required for CircleCI Server to function have been created:

- circle
- contexts
- conductor_production
- contexts_service_production
- cron_service_production
- domain
- federations
- permissions
- vms

Note: Each database in the list above needs the `uuid-ossp` extension to be enabled. The `circle` database also needs the `pgcrypto` extension to be enabled.

If you cannot see the databases, run the following commands to create them manually:

```
CREATE DATABASE vms;
\connect vms
 CREATE EXTENSION IF NOT EXISTS "uuid-ossp";
```

### Proxy timeouts

In cases were there is a proxy in front of PostgreSQL, it is best practice to enable keepalive messages with a duration longer than your proxy provider's threshhold using:

```
keepalives_idle = 30
```

## Vault

To view issues with vault, run:

```
docker logs -f context_services
```

### Sample Vault Configuration & Setup

1. Install Vault binary. HashiCorp recommends use of consul for HA replication, but specific implementation is beyond the scope of this document.
2. Setup Vault's configuration file, i.e. `vault.hcl`:

   ```
   storage "file" {
     address = "127.0.0.1:8500"
     path    = "/vault"
   }

   listener "tcp" {
     address     = "0.0.0.0:8200"
     tls_disable = 1
   }
   ```

   **NOTE: The above example is not suitable for production**

   This trivial example uses file storage and an insecure TCP connection. In production we strongly encourage the use of a HA backend, such as consul, and using SSL/TLS for all connections. See the Vault docs.
3. Configure, serve and unseal Vault. Until vault is "unsealed" it does not have the ability to encrypt or decrypt secrets. For more information, see seal concepts.

   ```
   vault server -config=/path/to/config.hcl
   export VAULT_ADDR=http://127.0.0.1:8200
   Vault operator init
   # Note the root token and 5 unseal tokens - These can not be recovered !!!
   vault operator unseal
   vault operator unseal
   vault operator unseal
   ```
4. Enable Transit Secret Engine

   ```
   vault secrets enable transit
   ```