What is Cassandra?

Apache Cassandra is an open source distributed database management system designed to handle large amounts of data across many commodity servers, providing high availability with no single point of failure. Cassandra offers robust support for clusters spanning multiple datacenters, with asynchronous masterless replication allowing low latency operations for all clients.

[wikipedia.org/wiki/Apache\_Cassandra⁠](https://en.wikipedia.org/wiki/Apache_Cassandra)



How to use this image

Start a cassandra server instance

Starting a Cassandra instance is simple:

$ docker run --name some-cassandra --network some-network -d cassandra:tag

... where some-cassandra is the name you want to assign to your container and tag is the tag specifying the Cassandra version you want. See the list above for relevant tags.

Make a cluster

Using the environment variables documented below, there are two cluster scenarios: instances on the same machine and instances on separate machines. For the same machine, start the instance as described above. To start other instances, just tell each new node where the first is.

$ docker run --name some-cassandra2 -d --network some-network -e CASSANDRA\_SEEDS=some-cassandra cassandra:tag

For separate machines (ie, two VMs on a cloud provider), you need to tell Cassandra what IP address to advertise to the other nodes (since the address of the container is behind the docker bridge).

Assuming the first machine's IP address is 10.42.42.42 and the second's is 10.43.43.43, start the first with exposed gossip port:

$ docker run --name some-cassandra -d -e CASSANDRA\_BROADCAST\_ADDRESS=10.42.42.42 -p 7000:7000 cassandra:tag

Then start a Cassandra container on the second machine, with the exposed gossip port and seed pointing to the first machine:

$ docker run --name some-cassandra -d -e CASSANDRA\_BROADCAST\_ADDRESS=10.43.43.43 -p 7000:7000 -e CASSANDRA\_SEEDS=10.42.42.42 cassandra:tag

Connect to Cassandra from cqlsh

The following command starts another Cassandra container instance and runs cqlsh (Cassandra Query Language Shell) against your original Cassandra container, allowing you to execute CQL statements against your database instance:

$ docker run -it --network some-network --rm cassandra cqlsh some-cassandra

More information about the CQL can be found in the [Cassandra documentation⁠](https://cassandra.apache.org/doc/latest/cql/index.html).

Container shell access and viewing Cassandra logs

The docker exec command allows you to run commands inside a Docker container. The following command line will give you a bash shell inside your cassandra container:

$ docker exec -it some-cassandra bash

The Cassandra Server log is available through Docker's container log:

$ docker logs some-cassandra

Configuring Cassandra

The best way to provide configuration to the cassandra image is to provide a custom /etc/cassandra/cassandra.yaml file. There are many ways to provide this file to the container (via short Dockerfile with FROM + COPY, via [Docker Configs⁠](https://docs.docker.com/engine/swarm/configs/), via runtime bind-mount, etc), the details of which are left as an exercise for the reader.

To use a different file name (for example, to avoid all image-provided configuration behavior), use -Dcassandra.config=/path/to/cassandra.yaml as an argument to the image (as in, docker run ... cassandra -Dcassandra.config=/path/to/cassandra.yaml).

There are a small number of environment variables supported by the image which will modify /etc/cassandra/cassandra.yaml in some way (but the script is modifying YAML, so is naturally fragile):

* CASSANDRA\_LISTEN\_ADDRESS: This variable is for controlling which IP address to listen for incoming connections on. The default value is auto, which will set the [listen\_address⁠](http://docs.datastax.com/en/cassandra/3.0/cassandra/configuration/configCassandra_yaml.html?scroll=configCassandra_yaml__listen_address" \t "_blank) option in cassandra.yaml to the IP address of the container as it starts. This default should work in most use cases.
* CASSANDRA\_BROADCAST\_ADDRESS: This variable is for controlling which IP address to advertise to other nodes. The default value is the value of CASSANDRA\_LISTEN\_ADDRESS. It will set the [broadcast\_address⁠](http://docs.datastax.com/en/cassandra/3.0/cassandra/configuration/configCassandra_yaml.html?scroll=configCassandra_yaml__broadcast_address" \t "_blank) and [broadcast\_rpc\_address⁠](http://docs.datastax.com/en/cassandra/3.0/cassandra/configuration/configCassandra_yaml.html?scroll=configCassandra_yaml__broadcast_rpc_address" \t "_blank) options in cassandra.yaml.
* CASSANDRA\_RPC\_ADDRESS: This variable is for controlling which address to bind the thrift rpc server to. If you do not specify an address, the wildcard address (0.0.0.0) will be used. It will set the [rpc\_address⁠](http://docs.datastax.com/en/cassandra/3.0/cassandra/configuration/configCassandra_yaml.html?scroll=configCassandra_yaml__rpc_address" \t "_blank) option in cassandra.yaml.
* CASSANDRA\_START\_RPC: This variable is for controlling if the thrift rpc server is started. It will set the [start\_rpc⁠](http://docs.datastax.com/en/cassandra/3.0/cassandra/configuration/configCassandra_yaml.html?scroll=configCassandra_yaml__start_rpc" \t "_blank) option in cassandra.yaml.
* CASSANDRA\_SEEDS: This variable is the comma-separated list of IP addresses used by gossip for bootstrapping new nodes joining a cluster. It will set the seeds value of the [seed\_provider⁠](http://docs.datastax.com/en/cassandra/3.0/cassandra/configuration/configCassandra_yaml.html?scroll=configCassandra_yaml__seed_provider" \t "_blank) option in cassandra.yaml. The CASSANDRA\_BROADCAST\_ADDRESS will be added the seeds passed in so that the server will talk to itself as well.
* CASSANDRA\_CLUSTER\_NAME: This variable sets the name of the cluster and must be the same for all nodes in the cluster. It will set the [cluster\_name⁠](http://docs.datastax.com/en/cassandra/3.0/cassandra/configuration/configCassandra_yaml.html?scroll=configCassandra_yaml__cluster_name" \t "_blank) option of cassandra.yaml.
* CASSANDRA\_NUM\_TOKENS: This variable sets number of tokens for this node. It will set the [num\_tokens⁠](http://docs.datastax.com/en/cassandra/3.0/cassandra/configuration/configCassandra_yaml.html?scroll=configCassandra_yaml__num_tokens" \t "_blank) option of cassandra.yaml.
* CASSANDRA\_DC: This variable sets the datacenter name of this node. It will set the [dc⁠](http://docs.datastax.com/en/cassandra/3.0/cassandra/architecture/archsnitchGossipPF.html" \t "_blank) option of cassandra-rackdc.properties. You must set CASSANDRA\_ENDPOINT\_SNITCH to use the ["GossipingPropertyFileSnitch"⁠](https://docs.datastax.com/en/cassandra/3.0/cassandra/architecture/archsnitchGossipPF.html" \t "_blank) in order for Cassandra to apply cassandra-rackdc.properties, otherwise this variable will have no effect.
* CASSANDRA\_RACK: This variable sets the rack name of this node. It will set the [rack⁠](http://docs.datastax.com/en/cassandra/3.0/cassandra/architecture/archsnitchGossipPF.html" \t "_blank) option of cassandra-rackdc.properties. You must set CASSANDRA\_ENDPOINT\_SNITCH to use the ["GossipingPropertyFileSnitch"⁠](https://docs.datastax.com/en/cassandra/3.0/cassandra/architecture/archsnitchGossipPF.html" \t "_blank) in order for Cassandra to apply cassandra-rackdc.properties, otherwise this variable will have no effect.
* CASSANDRA\_ENDPOINT\_SNITCH: This variable sets the snitch implementation this node will use. It will set the [endpoint\_snitch⁠](http://docs.datastax.com/en/cassandra/3.0/cassandra/configuration/configCassandra_yaml.html?scroll=configCassandra_yaml__endpoint_snitch" \t "_blank) option of cassandra.yml.

Caveats

Where to Store Data

Important note: There are several ways to store data used by applications that run in Docker containers. We encourage users of the cassandra images to familiarize themselves with the options available, including:

* Let Docker manage the storage of your database data [by writing the database files to disk on the host system using its own internal volume management⁠](https://docs.docker.com/storage/volumes/" \t "_blank). This is the default and is easy and fairly transparent to the user. The downside is that the files may be hard to locate for tools and applications that run directly on the host system, i.e. outside containers.
* Create a data directory on the host system (outside the container) and [mount this to a directory visible from inside the container⁠](https://docs.docker.com/storage/bind-mounts/" \t "_blank). This places the database files in a known location on the host system, and makes it easy for tools and applications on the host system to access the files. The downside is that the user needs to make sure that the directory exists, and that e.g. directory permissions and other security mechanisms on the host system are set up correctly.

The Docker documentation is a good starting point for understanding the different storage options and variations, and there are multiple blogs and forum postings that discuss and give advice in this area. We will simply show the basic procedure here for the latter option above:

1. Create a data directory on a suitable volume on your host system, e.g. /my/own/datadir.
2. Start your cassandra container like this:
3. $ docker run --name some-cassandra -v /my/own/datadir:/var/lib/cassandra -d cassandra:tag

The -v /my/own/datadir:/var/lib/cassandra part of the command mounts the /my/own/datadir directory from the underlying host system as /var/lib/cassandra inside the container, where Cassandra by default will write its data files.

No connections until Cassandra init completes

If there is no database initialized when the container starts, then a default database will be created. While this is the expected behavior, this means that it will not accept incoming connections until such initialization completes. This may cause issues when using automation tools, such as docker-compose, which start several containers simultaneously.