CockroachDB

Emanuel Kranjec & Felix Tröbinger

May 5, 2020

Contents

Structure of the Paper

This paper is structured into chapters. Each chapter states the person that authored it.

how many

The first chapter is there to give the reader an overview of CockroachDB and the company behind the service, Cockroach Labs. It will also go into detail about the licensing and pricing (and historical changes thereof) of the software product.

andere chapters beschreiben

Chapter 2 will go into detail explaining how to set up a working cluster of CockroachDB nodes using Docker containers. The last sub-chapter (chapter 2.5) will go into detail about how to secure the CockroachDB cluster using certificates.

1 What is CockroachDB?

author: Felix Tröbinger

CockroachDB is advertised as a "cloud-native SQL database for building global, scalable cloud services that survive disasters"[1].

CockroachDB is a relational and transactional database built for consistent key-value store and horizontal scalability and geo-replication as a big feature. It is built to ensure data survives failures of any kind, be that disk, machine rack or even entire datacenters. Furthermore the database system is "strongly-consistent" and guarantees the ACID properites.[1]

CockroachDB is considered a distributed database. It used geo-replication to ensure fast data access regardless of physical location. The distribution and geo-replication is presumably also what got the software its name.

Cloud native, often heard alongside *microservices*, is a term that describes a system or a network of applications where single application, services or nodes are run inside Docker containers. Another important component in cloud-native computing is a load balancer, for example Kubernetes. CockroachDB is a good example of cloud-native computing since the Database is intended to be used in multiple locations and these multiple nodes are usually run in Docker containers.

1.1 History

CockroachDB started as an open-source project in 2014. Cockroach Labs was founded by ex-Google employees in the year 2015.[2] Spencer Kimball who is now CEO is also the original creator of the GNU Image Manipulation Program (GIMP).

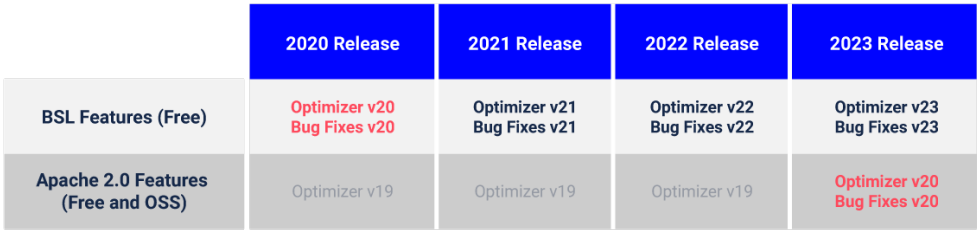
1.2 License and Pricing

As stated above, CockroachDB was original an entirely open-source project and licensed under the second version of the Apache License (APL 2.0). In 2019 however CockroachDB changed its license to a version of the Business Source License (BSL). In summary, CockroachDB can still be used with as many nodes as desired except for offering a commercial cloud database system. CockroachDB Core is no longer open-source, however the source code can still be viewed on the companies GitHub page. [3]

Another interesting change in their license change is that versions of CockroachDB that are older than three years are converted to the APL. Although

older versions (version 19.1 and downwards) are unaffected by this change and still use the Apache License as before. This is illustrated in figure 1.

Figure 1: The new CockroachDB License model[3]



In terms of pricing, there are three available "Tiers" for CockroachDB:

- *CockroachCloud*
- *CockroachEnterprise*
- *CockroachDB Core*

CockroachCloud gives customers a fully hosted and managed database platform. The price for this service is calculated per node and hour and differs depending on whether the customer chooses AWS (Amazon Web Services) or Google Cloud as a hosting platform. The price also changes depending on the CPU power and hard drive size of the node. [4]

The mayor differences to the **CockroachDB Enterprise** tier is that this option gives the customer superior support and that it is self hosted, meaning that the customer does not rely on Cockroach Labs to store their data, but will rather host it themselves. This means that the customer has to pay another company to do that for them or take it on their own to host CockroachDB nodes in a distributed manner. As for price, Cockroach Labs invite customers to contact them. [4]

On the CockroachDB page for pricing the company states that **CockroachDB Core** is open-source,[4] yet an article covering the licensing of the product explains that this version of the database is not actually open-source according to OSI's Open Source Definition (though the source code is still viewable to anyone).[3] This version however is free to download and use.

2 Setting up a CockroachDB Cluster with Docker

author: Felix Tröbinger

2.1 Creating a bridge network

Even though all created nodes – or rather containers – in this example will run on the same host, they still need to be connected via a network of some sorts. As a solution this example will use a Docker *bridge network*.

A bridge network is a software bridge, which allows containers on the same host to communicate with each other. Features of docker bridge networks are that user-defined bridges provide automatic DNS resolution between containers and that containers are isolated in a better way.[5]

The following command will create a bridge network called `roachnet`. This name will be passed to each node on startup/creation.

```
docker network create -d bridge roachnet
```

2.2 Creating Nodes

This example will create a cluster of three connected nodes. The first one is created with the following command.

```
docker run -d \
--name=roach1 \
--hostname=roach1 \
--net=roachnet \
-p 26257:26257 -p 8080:8080  \
-v "${PWD}/roach1:/cockroach/cockroach-data"  \
cockroachdb/cockroach:latest start \
--insecure \
--join=roach1,roach2,roach3
```

The node is connected to the previously created bridge network and given a name, `roach1` in this case.

Port 8080 of the host machine is mapped to port 8080 of the container which is later used for accessing the web-interface (See chapter 2.4). Port 26257 of the container is mapped to the same port on the host. This port is used for communication.

Adding a volume to the container will give the user the possibility to see what CockroachDB stores on disk and save data on container restart. To completely reset the node, the user has to delete the volume. If no volume is mapped on container start, all data will be lost when the container stops.

The Docker image in use is called `cockroachdb/cockroach` and this example uses the latest version. The command `start` is passed, which will start the CockroachDB inside the container.

For the time being this example will use the insecure mode of CockroachDB which does not require certificates. More on that in chapter 2.5.

Finally the command will use the `join` argument to connect the three nodes in the cluster by specifying their `hostname`.

For this example two more nodes will be created (nodes `roach2` and `roach3`) using a very similar command as the one above, only changing the hostname and the volume directory on the host machine. The last thing that differs for the last two nodes is that the ports will not be mapped as witch `roach1`. This is because when the web-interface in this example will be accessed, it will access the first created node. For more information about the web-interface see chapter 2.4.

As the next step the cluster has to be initialized. This is done with the following command:

```
docker exec -it roach1 ./cockroach init --insecure
```

[6]

2.3 SQL Command Line

Using the CockroachDBs SQL client can be done via the following command. In this example the SQL client is used by accessing the first node that was created, `roach1`.

```
docker exec -it roach1 ./cockroach sql --insecure
```

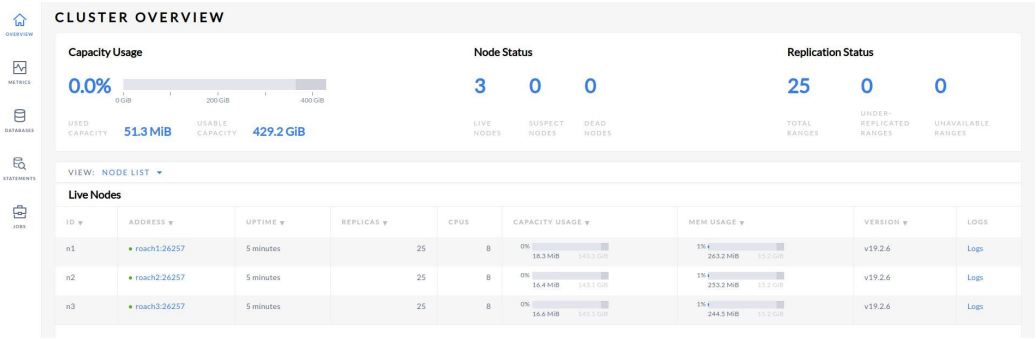
CockroachDB uses the *PostgreSQL* dialect.

2.4 Web-Interface

Accessing the admin web-interface user interface is as easy as navigating to `localhost:8080` in a browser. Port 8080 is used because the host port 8080 is mapped to port 8080 on `roach1`. If this is already in use by another application running on the host machine, port 8080 within the container can

be remapped to a different port on the host machine. This would be done by changing `-p 8080:8080` to `-p <YOUR-PORT>:8080` in the command that started the first node.

Figure 2: Cluster Overview

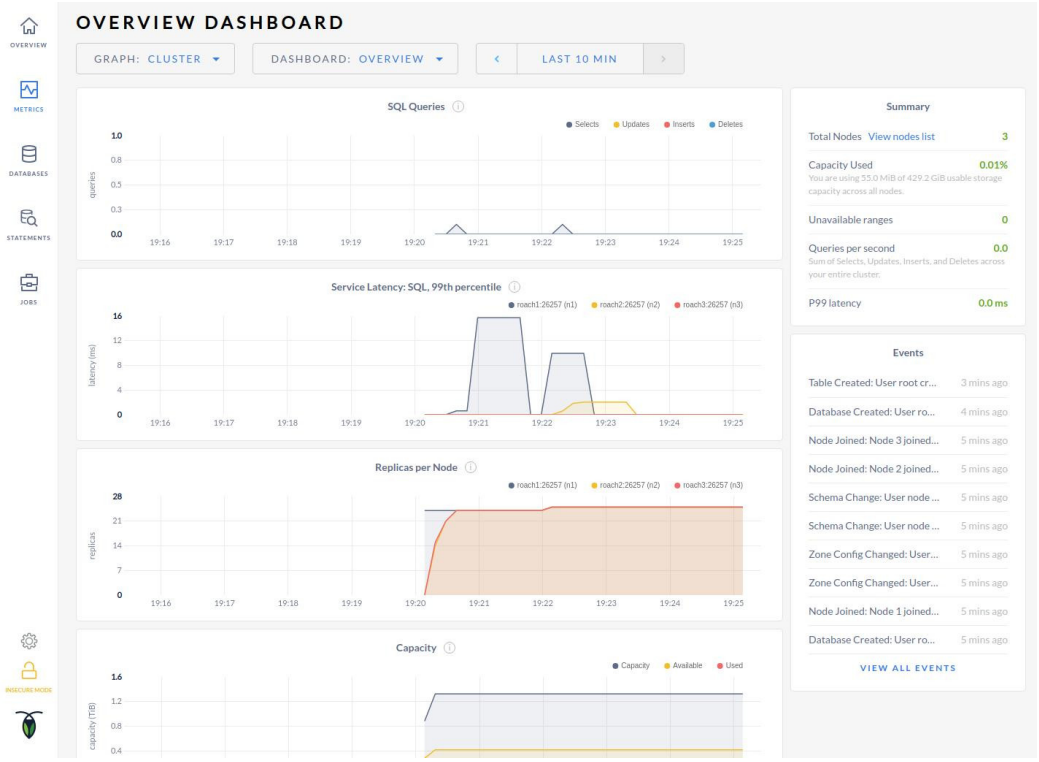


The first view the user is greeted with is the overview of the cluster which can be seen in Figure 2. This view shows all nodes connected to the cluster and the cluster capacity as well as the capacities for every single node. Also visible is the amount of failing and failed nodes. In the *Replication Status* one can see how many ranges/cells are stored and replicated across all CockroachDB nodes.

In the above figure, the three previously created nodes are visible and all running. Note that every node thinks its full capacity regarding disk and memory usage are that of the main host. So in this case all three nodes assume that they can use the entirety of remaining disk space and RAM even though they really have to share these resources with the rest of the CockroachDB nodes (or really any other container or service) that also happens to run on the host machine.

Clicking on the *Logs* button next to the node in the *Live nodes* view would allow the user to see Logs specific for each node. In this example this is unavailable because the debugging mode is set to `local` by default which disables this feature.

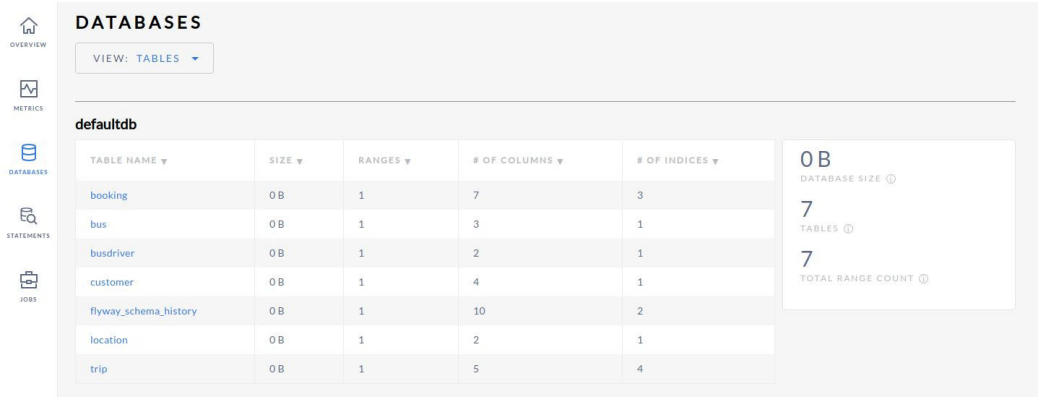
Figure 3: The CockroachDB Web-Interface Dashboard



On the Dashboard view (also labeled *Metrics* in the web UI) users can see graphs of SQL Queries ran on different nodes as well as latency and replication information. In this example we created tables and inserted data on the first node. This is visible in the *Replicas per node* graph where the line representing `roach1` immediately jumps to the value whereas the other two nodes take a little bit of time replicating the data on their system.

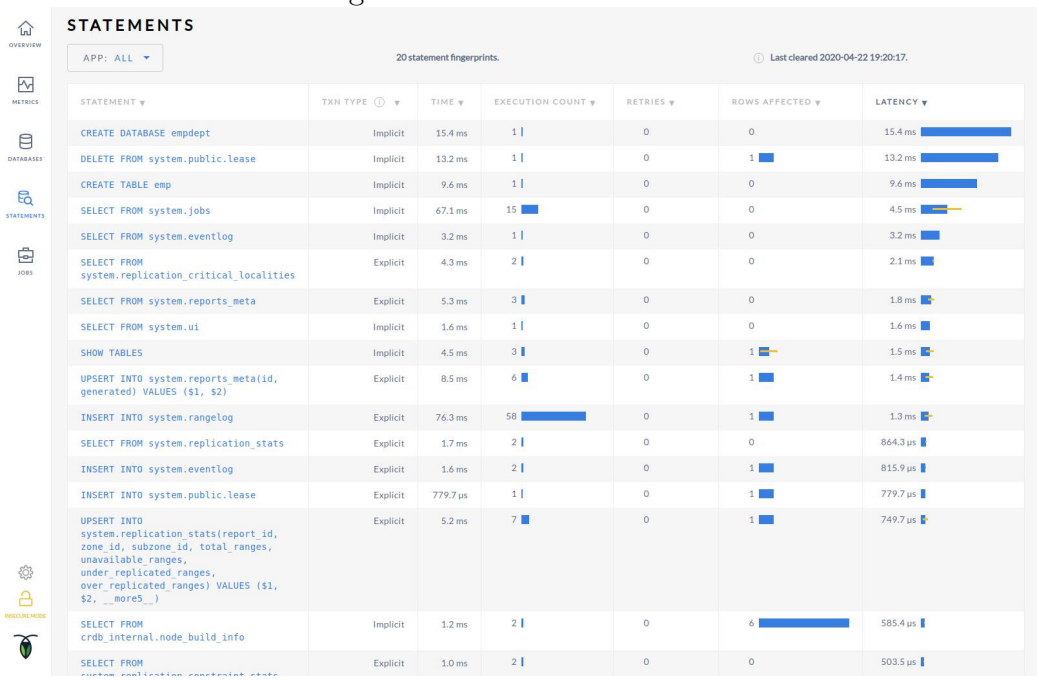
On the right-hand side a list of events that occurred in the database system is visible. In the example you can see that the latest events were the creation of the database and the tables therein.

Figure 4: View of databases



The *Databases* view show a list of all databases and lists their tables. Note that it does not display contents of the database.

Figure 5: View of statements



The *Statements* view list all executed SQL commands/statements which can be filtered and sorted.

2.5 Securing the Cluster

author: Felix Tröbinger

In the previous example all nodes have been started with the command line option `--insecure`. In the bottom of the web-interface the user is also reminded that CockroachDB is running in insecure mode by the golden lock icon. This is obviously a bad idea to be used in production or for any database, really.

Running CockroachDB in secure mode requires using certificates. These can be created by running either the `cockroach cert` command or using `opensll`. Alternatively a custom CA can be used.[7] For this example the Cockroach command will be used.

```
cockroach cert create-ca \
    --certs-dir=certs \
    --ca-key=my-safe-directory/ca.key
```

The above command generates

1. a CA (= **C**ertificate **A**uthority) certificate and clients/node certificates
2. a CA key

When creating secure nodes or client certificates the CA key is required and referenced during the creation process. In a scenario where the CA key is lost, adding nodes to a cluster created from nodes that were signed with keys is not possible anymore.

Creating certificates and key pairs for a node can be done with the following command. (Replace *hostname* with your hostname/container name.)

```
cockroach cert create-node \
    localhost $(hostname) \
    --certs-dir=certs \
    --ca-key=my-safe-directory/ca.key
```

When using a Docker container the certificates have to be in a specific directory of the container. When starting the container the `--certs-dir` is used to tell the container the specified location of the certificates. Similarly the `--ca-key` option is used to tell the container the location of the CA key when creating it. One possibility to add the certificates to the container is to use the volume command-line-parameter (similar to the one that was used

to create nodes in the beginning of chapter 2.2. A perhaps more elegant solution would be a Dockerfile that specifies options for either a single node or an entire cluster of nodes that run on a single machine.

The command will use the CA key that was created in the first command and stored in the *my-safe-directory* directory to create certificates for the new node.

In the next step a certified client is created, in this case for a user called *fred*.

```
cockroach cert create-client fred \
    --certs-dir=certs \
    --ca-key=my-safe-directory/ca.key
```

After this step a SQL user by the same name is created and granted some rights in the database. [8]

```
cockroach sql \
    --certs-dir=certs \
    --host=localhost:26257 \
    --execute="CREATE USER fred;
    GRANT SELECT ON TABLE db.example_table TO fred;"
```

A certified client can then be used in an application.

References

[1] cockroachdb. Cockroachdb on github, https://github.com/cockroachdb/cockroach.

[2] Klint Finley. Ex-googlers get millions to help you build the next google, https://www.wired.com/2015/06/cockroach-labs/.

[3] Spencer Kimball Peter Mattis, Ben Darnell. Why we're relicensing cockroachdb, https://www.cockroachlabs.com/blog/oss-relicensing-cockroachdb/.

[4] Cockroach Labs. Cockroachdb pricing, https://www.cockroachlabs.com/pricing.

[5] docker docs. Use bridge networks, https://docs.docker.com/network/bridge/.

[6] Cockroach Labs. Start a cluster in docker (insecure), https://www.cockroachlabs.com/docs/stable/start-a-local-cluster-in-docker-linux.html.

[7] Cockroach Labs. cockroach cert, https://www.cockroachlabs.com/docs/v19.2/cockroach-cert.html.

[8] Cockroach Labs. Start a local cluster (secure), https://www.cockroachlabs.com/docs/stable/secure-a-cluster.html.