[[Ch03 – Getting Started]]

# Getting Started

CockroachDB has a sophisticated and modern architecture and is designed for global scale. However, that complexity and scalability don't imply a steep learning curve or barrier to entry. In this chapter we'll help you get started with a CockroachDB installation and introduce you to the basics of working with a CockroachDB system.

## Installation

CockroachDB can be installed on virtually any flavor of desktop operating system within a few minutes. Alternatively, you can create a free CockroachCloud database or run CockroachDB within a Docker container or Kubernetes cluster.

### Installing CockroachDB software

In most scenarios, you'll want to have the CockroachDB software installed on your desktop computer, so let's start with that. You'll find a full list of CockroachDB binaries at <https://www.cockroachlabs.com/docs/releases/>. From there, you can pick your operating system and download the most recent version or pick a previous version.

The instructions below worked as of the time of writing, but installation can change with each release, so make sure you consult the CockroachDB website for the most up to date instructions. In particular, in

#### Installation on MacOS

If you have the +brew+ package manager installed, then that is probably the easiest way to get started installing CRDB on Mac. In fact, even if you don't have +brew+ installed, it’s probably easier to install it and then install +CRDB+ than to install CRDB manually.

To install +brew+, issue the following command from a terminal window:

[source,bash]

----

/bin/bash -c "$(curl -fsSL <https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh>)"

----

Once brew is installed, you can install CRDB with the following command:

[source,bash]

----

guyharrison@macos ~ % brew install cockroachdb/tap/cockroach

==> **Tapping cockroachdb/tap**

Cloning into '/usr/local/Homebrew/Library/Taps/cockroachdb/homebrew-tap'...

…

==> **Installing cockroach from cockroachdb/tap**

….

To have launchd start cockroachdb/tap/cockroach now and restart at login:

brew services start cockroachdb/tap/cockroach

Or, if you don't want/need a background service you can just run:

cockroach start-single-node --insecure

==> **Summary**

🍺 /usr/local/Cellar/cockroach/20.2.7: 134 files, 184.8MB, built in 8 seconds

[source,bash]

----

One of the great things about brew is that it sets up CRDB as a service, so you can issue +brew services start cockroach+ to start a background instance of CRDB.

However, if you don't want to use brew, then you can download the CockroachDB binary directly and copy the binary into your path. Visit <https://www.cockroachlabs.com/docs/releases/?filters=mac> to determine the path for the release you want, then use +curl+ or +wget+ to copy and decompress that binary:

[source,bash]

----

curl https://binaries.cockroachdb.com/cockroach-v20.2.7.darwin-10.9-amd64.tgz | tar -xJ

----

You can then copy the binary into your PATH so you can execute cockroach commands from any shell:

[source,bash]

----

$ sudo cp -i cockroach-v20.2.7.darwin-10.9-amd64/cockroach /usr/local/bin/

----

Note that copying the binary directly into your path may not install some of the ancillary libraries which support geospatial functionality. Consult the CockroachDB web site (https://www.cockroachlabs.com/docs/stable/install-cockroachdb-mac.html) for further details.

Once you've installed CRDB either manually, or via brew, run the +cockroach demo+ command to start a demo instance and confirm that it is running:

[source,bash]

----

guyharrison@macos ~ % cockroach demo

#

# Welcome to the CockroachDB demo database!

#

# You are connected to a temporary, in-memory CockroachDB cluster of 1 node.

#

# Enter \? for a brief introduction.

#

root@127.0.0.1:49418/movr> show databases;

database\_name | owner

----------------+--------

defaultdb | root

movr | root

postgres | root

system | node

(4 rows)

Time: 1ms total (execution 1ms / network 0ms)

root@127.0.0.1:49418/movr>

----

#### Installation on Linux

To perform a basic installation on Linux, visit <https://www.cockroachlabs.com/docs/releases/> To obtain the latest release (or a specific version you are interested in), download and unpack it. Of course, you can use +curl+ or +wget+ to obtain the tarball once you have determined its path:

[source,bash]

----

$ wget https://binaries.cockroachdb.com/cockroach-v20.2.7.linux-amd64.tgz

2021-04-17 16:10:35 (1.83 MB/s) - 'cockroach-v20.2.7.linux-amd64.tgz’ saved [94338926/94338926]

$ sudo cp -i cockroach-v20.2.7.linux-amd64/cockroach

----

Once installed, run the +cockroach demo+ command to start a temporary local instance of CRDB and verify the installation.

[source,bash]

----

$ cockroach demo

#

# Welcome to the CockroachDB demo database!

#

# You are connected to a temporary, in-memory CockroachDB cluster of 1 node.

#

…

#

# Enter \? for a brief introduction.

#

root@127.0.0.1:44913/movr> show databases;

database\_name | owner

----------------+--------

defaultdb | root

movr | root

postgres | root

system | node

(4 rows)

Time: 1ms total (execution 1ms / network 0ms)

[root@127.0.0.1:44913/movr](mailto:root@127.0.0.1:44913/movr)>

----

#### Installation on Microsoft Windows

Microsoft Windows is not a fully supported platform for running a CockroachDB server; certain features - such as spatial - is not available. However, Windows is completely supported for CockroachDB clients and the server runs well enough for experimentation and most development.

From <https://www.cockroachlabs.com/docs/releases/?filters=windows>, click on the link for the release you'd like to download. Once downloaded, unzip the archive into a directory and add the subdirectory containing the +cockroach.exe+ to your PATH.

Alternatively, you can download the file directly from a PowerShell prompt. Here, we download the zip fie for version 20.2.7, unzip it to a directory `c:\tools\cockroachdb`:

[source,powershell]

----

PS > wget https://binaries.cockroachdb.com/cockroach-v20.2.7.windows-6.2-amd64.zip

-OutFile crdb.zip

PS > mkdir c:\tools\cockroachdb

PS > Expand-Archive -Path crdb.zip -DestinationPath C:\tools\cockroachdb\

----

At the time of writing, it was necessary to download the GOLANG timezone zipfile and add an environment variable ZONEINFO pointing to that file.

To add a Windows environment variable, hold down the Windows and “R” keys simultaneously, then type control in the Run dialogue that appears. From the control panel, Select “System and Security” then “System”. Scroll down to “Advanced System Settings”, select that option, then click the “Environment variables” button. From there add a ZONEINFO environment variable that points to the directory containing the zoneinfo.zip file.

In the example below we download the zoneinfo.zip file and check that the ZONEINFO variable is set correctly.

[source,powershell]

----

PS > wget [https://github.com/golang/go/raw/master/lib/time/zoneinfo.zip -OutFile zoneinfo.zip](https://github.com/golang/go/raw/master/lib/time/zoneinfo.zip%20-OutFile%20zoneinfo.zip)

PS > Get-Item -Path Env:\ZONEINFO

Name Value

---- -----

ZONEINFO C:\tools\cockroachdb\cockroach-v20.2.7.windows-6.2-amd64\zoneinfo.zip

----

Once that is done you can test your installation by issuing the +cockroach demo+ command:

[source,powershell]

----

PS > cockroach demo

#

# Welcome to the CockroachDB demo database!

#

# You are connected to a temporary, in-memory CockroachDB cluster of 1 node.

#

#

# Enter \? for a brief introduction.

#

root@127.0.0.1:57574/movr> show databases;

database\_name | owner

----------------+--------

defaultdb | root

movr | root

postgres | root

system | node

(4 rows)

Time: 2ms total (execution 1ms / network 0ms)

----

.Connection URLs

\*\*\*\*

When connecting to a CockroachDB cluster, we need to identify the location and credentials with which we wish to connect. When connecting to a local server using +cockroach demo+ or +cockroach sql+, the CockroachDB client will default to a local server on the default port, but as we will see, more complex installations require quite a bit more information.

The most common way to connect is to use a PostgreSQL compatible URL. This URL is of the following format:

[source,bash]

----

postgresql://[user[:passwd]@]host[:port]/[db][?parameters...]

----

The simplest possible URL – for a local cluster running without authentication would look something like this:

[source, bash]

----

$ cockroach sql --url 'postgres://root@localhost:26257?sslmode=disable'

#

# Welcome to the CockroachDB SQL shell.

# All statements must be terminated by a semicolon.

# To exit, type: \q.

#

# Server version: CockroachDB CCL v20.2.4 (x86\_64-apple-darwin14, built 2021/01/21 00:12:56, go1.13.14) (same version as client)

# Cluster ID: 072189bb-3970-4f37-afe4-55bc37cdf76e

#

# Enter \? for a brief introduction.

#

root@localhost:26257/defaultdb>

----

This is equivalent to running the command +cockroach sql –insecure+.

The beauty of the URL is that it can be accepted by most PostgreSQL-compatible programs or drivers. For instance, if we have the PostgreSQL client installed, we can use it to connect to CockroachDB:

[source, bash]

----

✗ psql 'postgres://root@localhost:26257?sslmode=disable'

psql (13.2, server 9.5.0)

Type "help" for help.

root=#

----

\*\*\*\*

### Creating a CockroachCloud cluster

The +cockroach demo+ command is a handy way for playing with the CockroachDB server, but the easiest way to get a fully functional CockroachDB server with persistent storage is to take advantage of the CockroachCloud free cloud database service. This service grants you access to a fully functional multi-tenant cloud service with 5GB of storage.   
  
The CockroachCloud has a number of advantages compared with a desktop deployment:

* It's automatically configured for high availability and backup. You don't have to worry about losing your data in the event of a hard drive failure on your desktop.
* It's fully secured using encryption at rest and in transit
* It's available from anywhere, so it can be used for team development purposes.

To create a CockroachCloud server, navigate to <https://www.cockroachlabs.com/get-started-cockroachdb/> and select the CockroachCloud option. Enter your email as shown in <<Figure03-01>>.

[[Figure03-01]]   
.Signing up for CockroachCloud

image::images/Fig03-01SigningUp.png[Signing up for CockroachCloud]

Graphical user interface, application, Teams

Description automatically generated

After entering your details and validating your email address, you'll be given the option to create your free cluster as shown in <<Figure03-02>>.

[[Figure03-02]]   
.Creating a free CockroachCloud Database

image::images/Fig03-02CreatingCloudDB.png[Creating a free CockroachCloud Database]

Graphical user interface, application, Teams

Description automatically generated

Once created, the \*Connection info\* dialogue should appear, with information on how to connect to your new cluster – see <<Figure03-03>>. Download the CA certificate +cc-ca.crt++ by clicking on the first link in the dialogue and store it on your desktop.

[[Figure03-03]]   
.Connecting to CockroachCloud

image::images/Fig03-03ConnectingCockroachCloud.png[Connecting to CockroachCloud]

Graphical user interface, application, Teams

Description automatically generated

Once the certificate is stored on the desktop, you can use the connection string provided to establish a connection. Below we copy the +cc-ca.crt+ file into a +`~/.cockroach-certs/ca.crt + folder, and then connect to the newly created cloud database.

[source, bash]  
----

$ mkdir $HOME/.cockroach-certs

$ cp $HOME/Downloads/cc-ca.crt $HOME/.cockroach-certs

$ cockroach sql --url 'postgres://guy:b4\_jPoEYw4\_Ixsj7@free-tier6.gcp-asia-southeast1.cockroachlabs.cloud:26257/defaultdb?sslmode=verify-full&sslrootcert=/Users/guyharrison/.cockroach-certs/cc-ca.crt&options=--cluster=grumpy-orca-56'

#

# Welcome to the CockroachDB SQL shell.

# All statements must be terminated by a semicolon.

# To exit, type: \q.

#

# Client version: CockroachDB CCL v20.2.4 (x86\_64-apple-darwin14, built 2021/01/21 00:12:56, go1.13.14)

# Server version: CockroachDB CCL v20.2.8 (x86\_64-unknown-linux-gnu, built 2021/04/23 13:54:57, go1.13.14)

# Cluster ID: 45851b67-5277-4795-aab9-390c70a78786

#

# Enter \? for a brief introduction.

#

guy@free-tier6.gcp-asia-southeast1.cockroachlabs.cloud:26257/defaultdb>   
 show databases;

database\_name | owner

----------------+--------

defaultdb | root

postgres | root

system | node

(3 rows)

Time: 106ms total (execution 7ms / network 100ms)  
----

.CockroachCloud passwords

\*\*\*\*

Note that the password in the connection string is \*not\* the password you provided to connect to your CockroachCloud account. Your CockroachCloud account might be associated with many databases, each of which has its own password.

The password shown in the connection dialogue in <<Figure03-03>> will be shown only if you hover over the +REVEAL\_PASSWORD+ link and will only be shown at this point in the database creation. It's up to you to save that password and keep it safe.

\*\*\*\*

### Starting a local single-node server

As we've seen above, you can use the +cockroach demo+ command to start a temporary demo cluster, and we can quickly create a free CockroachCloud server. But if you want to start a single-node CockroachDB with persistent storage on your own hardware, you can use the +start=single-node+ option:

[source,bash]

----

$ cockroach start-single-node --insecure

\*

\* WARNING: ALL SECURITY CONTROLS HAVE BEEN DISABLED!

\*

\* This mode is intended for non-production testing only.

\*

\* In this mode:

\* - Your cluster is open to any client that can access any of your IP addresses.

\* - Intruders with access to your machine or network can observe client-server traffic.

\* - Intruders can log in without password and read or write any data in the cluster.

\* - Intruders can consume all your server's resources and cause unavailability.

\*

\*

\* INFO: To start a secure server without mandating TLS for clients,

\* consider --accept-sql-without-tls instead. For other options, see:

\*

\* - https://go.crdb.dev/issue-v/53404/v20.2

\* - https://www.cockroachlabs.com/docs/v20.2/secure-a-cluster.html

\*

\*

\* WARNING: neither --listen-addr nor --advertise-addr was specified.

\* The server will advertise "mubuntu" to other nodes, is this routable?

\*

\* Consider using:

\* - for local-only servers: --listen-addr=localhost

\* - for multi-node clusters: --advertise-addr=<host/IP addr>

----

This will start a single node CockroachDB cluster with no security. To connect to this server we can use the +cockroach sql+ command with the default connection string:

[source,bash]

----

$ cockroach sql --insecure

#

# Welcome to the CockroachDB SQL shell.

# All statements must be terminated by a semicolon.

# To exit, type: \q.

#

# Server version: CockroachDB CCL v20.2.4 (x86\_64-unknown-linux-gnu, built 2021/01/21 00:08:24, go1.13.14) (same version as client)

# Cluster ID: 848d8b85-4000-484a-b4ad-8f2c76c68221

#

# Enter \? for a brief introduction.

#

root@:26257/defaultdb> show databases;

database\_name | owner

----------------+--------

defaultdb | root

postgres | root

system | node

(3 rows)

Time: 3ms total (execution 2ms / network 0ms)

root@:26257/defaultdb>

----

.Insecure mode

\*\*\*\*

The use of the +insecure+ flag when starting a CockroachDB server is convenient for quickly starting a CockroachDB server, but it is absolutely not appropriate for a production system. Please see Chapter ?? for instructions on setting up a properly secured production system.

\*\*\*\*

### Starting up CockroachDB in a docker container

If you have docker, you can quickly start a CockroachDB single node instance inside a docker container.

You'll need a persistent volume for data, so let's create that first:

[source,bash]

----

$ docker volume create crdb1

----

Then, we invoke +docker run+ to pull and start the latest CockroachDB docker image and start the server in single-node, insecure mode:

[source,bash]

----

$ docker run -d \

> --name=crdb1 \

> --hostname=crdb1 \

> -p 26257:26257 -p 8080:8080 \

> -v "crdb1:/cockroach/cockroach-data" \

> cockroachdb/cockroach:latest start-single-node \

> --insecure \

>

Unable to find image 'cockroachdb/cockroach:latest' locally

latest: Pulling from cockroachdb/cockroach

a591faa84ab0: Pull complete

…

6913e7a5719b8cb705c32540523885f6592270cf091ac1013cca66914b1aafe8

----

The output of the docker run command is the container identifier for the CockroachDB container. Using that containerId, we can connect to that container using the +cockroach sql+ command.

[source,bash]

----

$ docker exec -it 6913e7a5719b8cb705c32540523885f6592270cf091ac1013cca66914b1aafe8 cockroach sql --insecure

#

# Welcome to the CockroachDB SQL shell.

# All statements must be terminated by a semicolon.

# To exit, type: \q.

#

# Server version: CockroachDB CCL v20.2.7 (x86\_64-unknown-linux-gnu, built 2021/03/29 17:52:00, go1.13.14) (same version as client)

# Cluster ID: 8fcbb9bb-ec7c-40dc-afe0-90306c87f5d7

#

# Enter \? for a brief introduction.

#

root@:26257/defaultdb> show databases;

database\_name | owner

----------------+--------

defaultdb | root

postgres | root

system | node

(3 rows)

Time: 3ms total (execution 3ms / network 0ms)

----

We don't need to have the CockroachDB software installed on our local host to connect using the above method, since we are using the cockroachdb client installed within the docker container. However, since we've forwarded port 26257 from the docker container, we can attach from the desktop using the default connection:

[source,bash]  
----

$ ~ cockroach sql --insecure

#

# Welcome to the CockroachDB SQL shell.

# All statements must be terminated by a semicolon.

# To exit, type: \q.

#

# Client version: CockroachDB CCL v20.2.4 (x86\_64-apple-darwin14, built 2021/01/21 00:12:56, go1.13.14)

# Server version: CockroachDB CCL v20.2.7 (x86\_64-unknown-linux-gnu, built 2021/03/29 17:52:00, go1.13.14)

# Cluster ID: d070609f-58a7-4aea-aa27-92bc4a1e5406

#

# Enter \? for a brief introduction.

#

root@:26257/defaultdb>

----

Note that this port forwarding can only work if there's not already a CockroachDB server listening on that port.

### Starting up a secure server

In the previous examples, we've used the +--insecure+ mode to start the server without needing to configure secure communications. This is a quick way to set up a test server but is catastrophically dangerous for anything that contains valuable data.

We'll cover CockroachDB security in-depth within Chapter 12, but for now, to set up a secure server, we need to create security certificates to encrypt the communications channel and authenticate the client and server.

The following commands create the certificates. The Certificate Authority key will be held in +my-safe-directory+; the certificates themselves will be held in the +certs+ directory:

[source,bash]

----

$ mkdir certs my-safe-directory

$ # CA certificate and keypair

$ cockroach cert create-ca \

> --certs-dir=certs \

> --ca-key=my-safe-directory/ca.key

$ # certificate and keypair for localhost

$ cockroach cert create-node localhost `hostname` --certs-dir=certs \

> --ca-key=my-safe-directory/ca.key

$ # certificate for the root user

$ cockroach cert create-client root \

> --certs-dir=certs \

> --ca-key=my-safe-directory/ca.key

----

We can now start the server and specify the directory containing the certificates:

[source,bash]

----

$ # start single node

$ cockroach start-single-node --certs-dir=certs --background

\*

\* WARNING: neither --listen-addr nor --advertise-addr was specified.

\* The server will advertise "mubuntu" to other nodes, is this routable?

\*

\* Consider using:

\* - for local-only servers: --listen-addr=localhost

\* - for multi-node clusters: --advertise-addr=<host/IP addr>

\*

\*

$ \*

\* INFO: Replication was disabled for this cluster.

\* When/if adding nodes in the future, update zone configurations to increase the replication factor.

\*

----

Now when connecting, we must specify the certificates directory. If we are connecting from a remote host then we would need to copy the certificates to that host.

[source,bash]

----

$ cockroach sql --certs-dir=certs

#

# Welcome to the CockroachDB SQL shell.

# All statements must be terminated by a semicolon.

# To exit, type: \q.

#

# Server version: CockroachDB CCL v20.2.4 (x86\_64-unknown-linux-gnu, built 2021/01/21 00:08:24, go1.13.14) (same version as client)

# Cluster ID: f908d29e-1fb6-40b8-9e1f-a2a0a3763603

#

# Enter \? for a brief introduction.

#

root@:26257/defaultdb>

----

.Certificates directory

\*\*\*\*

On Linux or MacOS systems, CockroachDB will look for certificates in the `~/.cockroach-certs directory. So if your certificates are placed there, then you won’t need to specify the – -certs-dir argument. However, if you have multiple CockroachDB servers then you may need to maintain distinct certificates for each, possibly in their own directories.

\*\*\*\*

### Remote connection

In the previous examples, we've connected to a server running on the same host as our client. This is pretty unusual in the real world, where we would normally be connecting to a server on another machine. Typically, we'd specify the URL parameter to identify the server concerned. For instance, to connect to a server on the +mubuntu+ server on the default port, we could issue the following command:

[source,bash]

----

$ cockroach sql --certs-dir=certs --url postgresql://root@mubuntu:26257/defaultdb

#

# Welcome to the CockroachDB SQL shell.

# All statements must be terminated by a semicolon.

# To exit, type: \q.

#

# Server version: CockroachDB CCL v20.2.4 (x86\_64-unknown-linux-gnu, built 2021/01/21 00:08:24, go1.13.14) (same version as client)

# Cluster ID: f908d29e-1fb6-40b8-9e1f-a2a0a3763603

#

# Enter \? for a brief introduction.

#

root@mubuntu:26257/defaultdb>

----

### Creating a Kubernetes cluster

In the above examples, we've created single-node clusters and connected to a free CockroachCloud database which is a shared region of a multi-tenant cluster. If you want to start with a dedicated multi-node cluster, then the easiest way is to install a CockroachDB cluster in a Kubernetes environment using the [CockroachDB Kubernetes operator](https://github.com/cockroachdb/cockroach-operator).

Kubernetes is an increasingly ubiquitous framework that coordinates – orchestrates – the management of the components of a distributed system. The [CockroachDB Kubernetes operator](https://github.com/cockroachdb/cockroach-operator) contains the configuration and utilities that allow CockroachDB to be deployed in Kubernetes.

We'll come back to production deployment options for Kubernetes later in the book. For now, we will deploy CockroachDB in a Kubernetes [Minikube](https://github.com/kubernetes/minikube) cluster, which implements a local Kubernetes cluster on a desktop system.

For this example, we are using a minikube cluster running on macOS with 6 CPUs and 12GB of memory.

The first step is to deploy the operator, and it's manifest:

[source,bash]  
----

$ kubectl apply -f https://raw.githubusercontent.com/cockroachdb/cockroach-operator/master/config/crd/bases/crdb.cockroachlabs.com\_crdbclusters.yaml

customresourcedefinition.apiextensions.k8s.io/crdbclusters.crdb.cockroachlabs.com created

$

$ kubectl apply -f https://raw.githubusercontent.com/cockroachdb/cockroach-operator/master/manifests/operator.yaml

clusterrole.rbac.authorization.k8s.io/cockroach-database-role created

serviceaccount/cockroach-database-sa created

clusterrolebinding.rbac.authorization.k8s.io/cockroach-database-rolebinding created

role.rbac.authorization.k8s.io/cockroach-operator-role created

clusterrolebinding.rbac.authorization.k8s.io/cockroach-operator-rolebinding created

clusterrole.rbac.authorization.k8s.io/cockroach-operator-role created

serviceaccount/cockroach-operator-sa created

rolebinding.rbac.authorization.k8s.io/cockroach-operator-default created

deployment.apps/cockroach-operator created

----

Once this is done, a +kubectl get pods+ command should show the CockroachDB Kubernetes operator running inside the cluster:

[source,bash]  
----

$ kubectl get pods

NAME READY STATUS RESTARTS AGE

cockroach-operator-84bf588dbb-65m8k 0/1 ContainerCreating 0 9s

----

We then retrieve the example configuration file that is included in the operators repository.

[source,bash]  
----

$ curl -O https://raw.githubusercontent.com/cockroachdb/cockroach-operator/master/examples/example.yaml

% Total % Received % Xferd Average Speed Time Time Time Current

Dload Upload Total Spent Left Speed

100 1098 100 1098 0 0 3399 0 --:--:-- --:--:-- --:--:-- 3399

----

This file contains definitions for the cluster to be configured, such as the number of nodes to be created and the memory and CPU required by each node. The configuration is tilted towards a production deployment, so you might want to trim down the requirements. For instance, below we see that the default configuration file specifies a 60GB storage requirement. We might want to change this to a lower value for a simple test system (or increase it for a bigger deployment):

[source,yaml]  
----

apiVersion: crdb.cockroachlabs.com/v1alpha1

kind: CrdbCluster

metadata:

name: cockroachdb

spec:

dataStore:

pvc:

spec:

accessModes:

- ReadWriteOnce

resources:

requests:

storage: "60Gi"

volumeMode: Filesystem

----

You could edit other elements of the configuration file, such as the number of nodes to be created or the version of CockroachDB to be used.

We now apply the configuration file to the operator, which will perform the necessary tasks to create the cluster:

[source,bash]  
----

$ kubectl apply -f example.yaml

crdbcluster.crdb.cockroachlabs.com/cockroachdb created

----

The cluster creation process can take some time. We'll know it's complete when a +kubectl get pods+ command shows all nodes in Running state:

[source, bash]

----

$ kubectl get pods

NAME READY STATUS RESTARTS AGE

cockroach-operator-84bf588dbb-65m8k 1/1 Running 0 6m59s

cockroachdb-0 1/1 Running 0 87s

cockroachdb-1 1/1 Running 0 71s

cockroachdb-2 1/1 Running 0 57s

----

We can connect to the cluster by invoking the +cockroach sql + command from within any of the CockroachDB nodes. For instance, here we connect to +cockroachdb-2+ and connect to the cluster:

[source, bash]

----

$ kubectl exec -it cockroachdb-2 -- ./cockroach sql --certs-dir cockroach-certs

#

# Welcome to the CockroachDB SQL shell.

# All statements must be terminated by a semicolon.

# To exit, type: \q.

#

# Server version: CockroachDB CCL v20.2.7 (x86\_64-unknown-linux-gnu, built 2021/03/29 17:52:00, go1.13.14) (same version as client)

# Cluster ID: cb78255b-befa-4447-9fa8-c06b7a353564

#

# Enter \? for a brief introduction.

#

root@:26257/defaultdb> show databases;

database\_name | owner

----------------+--------

defaultdb | root

postgres | root

system | node

(3 rows)

Time: 7ms total (execution 6ms / network 1ms)

----

Connecting to the cluster using the method above requires a very high level of access to the cluster. In a production environment, we would setup a load balancer in to securely handle incoming requests to the cluster. We’ll look at these sorts of configurations in chapter ??.

Meanwhile, to connect to the simple cluster we just created from outside the cluster, we need to first retrieve the client certificates that the operator created when the cluster was established:

[source, bash]

----

mkdir certs

kubectl exec cockroachdb-0 -it -- cat cockroach-certs/ca.crt >certs/ca.crt

kubectl exec cockroachdb-0 -it -- cat cockroach-certs/client.root.key >certs/client.root.key

kubectl exec cockroachdb-0 -it -- cat cockroach-certs/client.root.crt >certs/client.root.crt

chmod 600 certs/\*

----

Now we can forward one of the CockroachDB nodes ports to our local machine and connect using the +cockroach sql+ command:

[source, bash]

----

$ kubectl port-forward services/cockroachdb-public 26257:26257 -n default &

[1] 22643

$ Forwarding from [::1]:26257 -> 26257

$ cockroach sql --port 26257 --certs-dir certs

#

# Welcome to the CockroachDB SQL shell.

# All statements must be terminated by a semicolon.

# To exit, type: \q.

#

Handling connection for 26257

# Client version: CockroachDB CCL v20.2.4 (x86\_64-apple-darwin14, built 2021/01/21 00:12:56, go1.13.14)

# Server version: CockroachDB CCL v20.2.7 (x86\_64-unknown-linux-gnu, built 2021/03/29 17:52:00, go1.13.14)

# Cluster ID: cb78255b-befa-4447-9fa8-c06b7a353564

#

# Enter \? for a brief introduction.

#

root@:26257/defaultdb>

----

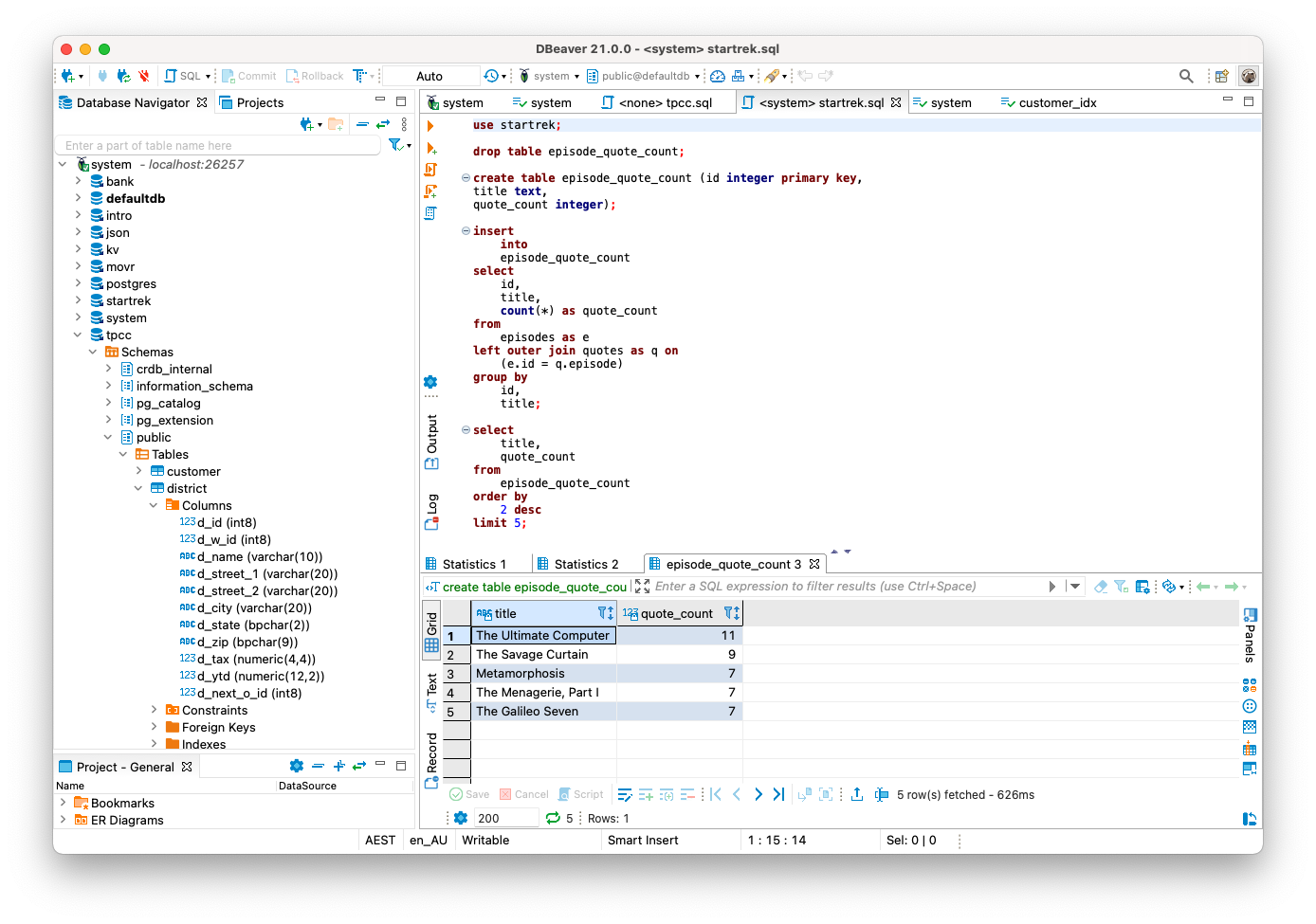
### Using a GUI client

While some are more than happy to use only a command-line client to interact with a database, some of us prefer a Graphical User Interface (GUI). Many GUI applications for PostgreSQL exist and most of these will work with CockroachDB. However, DBeaver Community edition is a free database GUI that has dedicated support for CockroachDB.

You can get DBeaver from <https://dbeaver.io>. [[Figure03-04]] shows the DBeaver GUI client.

[[Figure03-04]]   
.The DBeaver GUI

image::images/Fig03-04DBEaver.png[DBeaver GUI]



## Exploring CockroachDB

Now that we've got access to a CockroachDB cluster and have the client ready to connect let's take CockroachDB for a drive!

### Adding some data

As we say in Australia, "A database without data is like a Pub with no Beer!". Let's get some data into the database so that we have something to look at.

The CockroachDB software includes a number of demonstration databases that you can quickly add to your CockroachDB installation. In some cases, these databases are pre-populated with data; in other cases, you create the schemas then add data afterward.

To initialize the schemas, we use the +cockroach workload init [schema]+ command. To run a workload against the schema, we use the +cockroach workload run [schema]+ command.

The schemas include:

* **Bank**, which models a set of accounts with currency balances. After the initializing the schema, use +workload run+ to generate a workload against the database.
* **Intro**, a simple single-table database.
* **kv**, a simple key-value schema. After the initializing the schema, use run to generate a workload that will be evenly distributed across the cluster.
* **Movr**, a schema for the MovR example application. This schema can be used with the +workload run+ command to generate load against the databases.
* **Startrek**, A startrek database, with two tables, +episodes+ and +quotes+.
* **Tpcc**, a transaction processing schema for the TPCC standard benchmark. This schema can be used with the +workload run+ command to generate load against the databases.
* **Ycsb**, the Yahoo Cloud Serving Benchmark schema. This schema can be used with the +workload run+ command to generate load against the databases.

For the +intro+ and +startrek+ databases, we create the tables and data using the +workload init+ command. For instance, in the following example, we create the +startrek+ schema and look at some data:

[source,sql]

----

[root@crdb1 cockroach]# cockroach workload init startrek postgres://localhost:26257?sslmode=disable

I210501 04:29:29.694340 1 workload/workloadsql/dataload.go:140 imported episodes (0s, 79 rows)

I210501 04:29:29.898945 1 workload/workloadsql/dataload.go:140 imported quotes (0s, 200 rows)

[root@crdb1 cockroach]# cockroach sql --insecure

#

# Welcome to the CockroachDB SQL shell.

# All statements must be terminated by a semicolon.

# To exit, type: \q.

#

# Server version: CockroachDB CCL v20.2.7 (x86\_64-unknown-linux-gnu, built 2021/03/29 17:52:00, go1.13.14) (same version as client)

# Cluster ID: d070609f-58a7-4aea-aa27-92bc4a1e5406

#

# Enter \? for a brief introduction.

#

root@:26257/defaultdb> show databases;

database\_name | owner

----------------+--------

defaultdb | root

postgres | root

startrek | root

system | node

(4 rows)

Time: 2ms total (execution 2ms / network 0ms)

root@:26257/defaultdb> use startrek;

SET

Time: 1ms total (execution 0ms / network 0ms)

root@:26257/startrek> show tables;

schema\_name | table\_name | type | owner | estimated\_row\_count

--------------+------------+-------+-------+----------------------

public | episodes | table | root | 0

public | quotes | table | root | 0

(2 rows)

Time: 56ms total (execution 56ms / network 0ms)

root@:26257/startrek> select \* from episodes limit 1;

id | season | num | title | stardate

-----+--------+-----+--------------+-----------

1 | 1 | 1 | The Man Trap | 1531.1

(1 row)

Time: 1ms total (execution 1ms / network 0ms)

----

In this example, we create the bank schema:

[source,bash]

----

[root@crdb1 cockroach]# cockroach workload init bank postgres://localhost:26257?sslmode=disable

I210501 04:31:41.214008 1 workload/workloadsql/dataload.go:140 imported bank (0s, 1000 rows)

I210501 04:31:41.221478 1 workload/workloadsql/workloadsql.go:113 starting 9 splits

----

And then run a workload simulation for 60 seconds:

[source,bash]

----

[root@crdb1 cockroach]# cockroach workload run bank postgres://localhost:26257?sslmode=disable --duration 60s

I210501 04:33:52.340852 1 workload/cli/run.go:356 creating load generator...

I210501 04:33:52.344074 1 workload/cli/run.go:387 creating load generator... done (took 3.220303ms)

\_elapsed\_\_\_errors\_\_ops/sec(inst)\_\_\_ops/sec(cum)\_\_p50(ms)\_\_p95(ms)\_\_p99(ms)\_pMax(ms)

1.0s 0 187.3 187.9 16.8 48.2 65.0 121.6 transfer

2.0s 0 295.0 241.5 11.0 31.5 52.4 79.7 transfer

3.0s 0 260.9 248.0 13.1 37.7 54.5 83.9 transfer

4.0s 0 203.1 236.7 17.8 39.8 54.5 79.7 <snip>

\_elapsed\_\_\_errors\_\_\_\_\_ops(total)\_\_\_ops/sec(cum)\_\_avg(ms)\_\_p50(ms)\_\_p95(ms)\_\_p99(ms)\_pMax(ms)\_\_result

60.0s 0 14230 237.2 16.9 13.6 41.9 65.0 192.9

----

The run command is primarily meant to generated data for load testing purposes but is useful to generate data for query purposes as well.

### Databases and tables

As we've seen already, data in a CockroachDB deployment is organized into specific namespaces called databases. Database is a fairly loosely used and overloaded term – it's quite common for a CockroachDB cluster to be referred to as a database or for a database within a cluster to be referred to as a schema. However, in CockroachDB, as in most other SQL databases, a database cluster contains one or more databases. Within a database, one or more schemas may be defined, though it's common for each database to contain only one schema.

We can list the databases in the cluster using the +SHOW DATABASES+ command:

[source, sql]

----

root@:26257/defaultdb> show databases;

database\_name | owner

----------------+--------

bank | root

defaultdb | root

postgres | root

startrek | root

system | node

(5 rows)

----

We can set our current database with the +use+ command:

[source, sql]

----

root@:26257/defaultdb> use startrek;

SET

Time: 1ms total (execution 0ms / network 0ms)

----

We list tables within a database with the +show tables+ command:

[source, sql]

----

root@:26257/startrek> show tables;

schema\_name | table\_name | type | owner | estimated\_row\_count

--------------+------------+-------+-------+----------------------

public | episodes | table | root | 79

public | quotes | table | root | 200

(2 rows)

Time: 16ms total (execution 16ms / network 0ms)

----

We can describe a table using the +\d+ command:

[source, sql]

----

root@:26257/startrek> \d quotes;

column\_name | data\_type | is\_nullable | column\_default | generation\_expression | indices | is\_hidden

--------------+-----------+-------------+----------------+-----------------------+------------------------------+------------

quote | STRING | true | NULL | | {} | false

characters | STRING | true | NULL | | {} | false

stardate | DECIMAL | true | NULL | | {} | false

episode | INT8 | true | NULL | | {quotes\_episode\_idx} | false

rowid | INT8 | false | unique\_rowid() | | {primary,quotes\_episode\_idx} | true

(5 rows)

Time: 13ms total (execution 12ms / network 1ms)

----

### Issuing SQL

From the CockroachDB client, we can issue any SQL commands for which we are authorized.

Here we create a table within the startrek database, load it with some derived data and issue a query:

[source, sql]

----

root@:26257/startrek> create table episode\_quote\_count (id integer primary key,

title text,

quote\_count integer);

CREATE TABLE

Time: 23ms total (execution 23ms / network 0ms)

root@:26257/startrek> insert

into

episode\_quote\_count

select

id,

title,

count(\*) as quote\_count

from

episodes as e

left outer join quotes as q on

(e.id = q.episode)

group by

id,

title;

INSERT 79

Time: 25ms total (execution 25ms / network 0ms)

root@:26257/startrek> select

title,

quote\_count

from

episode\_quote\_count

order by

2 desc

limit 5;

title | quote\_count

------------------------+--------------

The Ultimate Computer | 11

The Savage Curtain | 9

Metamorphosis | 7

The Menagerie, Part I | 7

The Galileo Seven | 7

(5 rows)

Time: 2ms total (execution 2ms / network 0ms)

----

### The console

The CockroachDB server exposes a web-based client that shows the status of the cluster and useful performance metrics. The webserver is usually exposed on port 8080, though this can be changed using the +--http-addr+ setting when starting the server. [[Figure03-04]] shows an example of the console, in this case from the Kubernetes cluster that we started earlier in this chapter (we forwarded port 8080 from one of the pods in the cluster).

[[Figure03-05]]   
.The CockroachDB console

image::images/Fig03-05Console.png[CockroachDB console]

Graphical user interface, application, Teams

Description automatically generated

## Working with programming languages

Working with the CockroachDB shell is useful for experimentation, but eventually, most databases interact with application code written in languages such as Javascript, Java, Go or Python.

Because CockroachDB is wire compatible with Postgres, most Postgres compatible drivers will work with CockroachDB. Indeed, there are no CockroachDB-specific drivers on the market because the Postgres drivers work so well. In this section, we'll get you up to speed with "hello world" programs in Java, GoLang, Python and JavaScript that connect to and queries a CockroachDB cluster.

### Connecting to CockroachDB from NodeJS

Server-side Javascript using the NodeJS platform is an increasingly popular choice for application development because it allows the same Javascript language to be used for both front-end web presentation code and server-side application logic.

Assuming that you have nodeJS and the Node Package Manager (npm) installed, we'll use the +node-postgres+ driver to connect to CockroachDB. We can install this driver with the following command:

[source, shell]

----

npm install pg

----

Once pg is installed, then the following example should connect to any CockroachDB database using a connection URI:

[source, JavaScript]

----

// Example of connecting to CockroachDB using NodeJS

const CrClient = require('pg').Client; //load pg client

async function main() {

try {

// Check parameters

if (process.argv.length != 3) {

console.log('Usage: node helloWorld.js CONNECTION\_URI');

process.exit(1);

}

// Establish a connection using the command line URI

const connectionString = process.argv[2];

const crClient = new CrClient(connectionString);

await crClient.connect();

// Issue a SELECT

const data = await crClient.query(

`SELECT CONCAT('Hello from CockroachDB at ',

CAST (NOW() as STRING)) as hello`

);

// Print out the error message

console.log(data.rows[0].hello);

} catch (error) {

console.log(error.stack);

}

// Exit

process.exit(0);

}

main();

----

This program expects the connection string to be provided as the first argument to the program. The process.argv array contains the full command line including "node" and "helloWorld.js", so the URI actually shows up as the third element in the array.

We then attempt to establish a connection using that connection string, then issue a SELECT statement that retrieves the time as known to the server.

Here we connect to the CockroachCloud server that we setup earlier in this chapter:

[source, shell]

----

$ node helloWorld.js "postgres://guy:b4\_jPoEYw4\_Ixsj7@free-tier6.gcp-asia-southeast1.cockroachlabs.cloud:26257/defaultdb?sslmode=verify-full&sslrootcert=$HOME/CRDBKeys/cc-ca.crt&options=--cluster=grumpy-orca-56"

Hello from CockroachDB at 2021-05-02 00:17:40.835834+00:00

----

And here we connect to a local CockroachDB running in insecure mode:

[source, shell]

----

$ node helloWorld.js 'postgres://root@localhost:26257?sslmode=disable'

Hello from CockroachDB at 2021-05-02 00:32:39.125419+00:00

----

### Connecting to CockroachDB from Java

Java is the workhorse of millions of applications across all industries and contexts.

In this example, we will use the official PostgresSQL JDBC driver to connect to a CockroachDB server.

Download the JDBC driver from here: <https://jdbc.postgresql.org/download.html> and place it in your CLASSPATH or configure it as a dependency in your IDE.

The following program accepts a URL, username and password as arguments on the command line and connects to the CockroachDB cluster concerned, and issues a SELECT statement:

[source, java]

----

package helloCRDB;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.ResultSet;

import java.sql.Statement;

public class HelloCRDB {

public static void main(String[] args) {

Connection cdb = null;

try {

Class.forName("org.postgresql.Driver");

String connectionURL="jdbc:"+args[0];

String userName=args[1];

String passWord=args[2];

cdb = DriverManager.getConnection(connectionURL,userName,passWord);

Statement stmt = cdb.createStatement();

ResultSet rs = stmt

.executeQuery("SELECT CONCAT('Hello from CockroachDB at ',"

+ "CAST (NOW() as STRING)) AS hello");

rs.next();

System.out.println(rs.getString("hello"));

} catch (Exception e) {

e.printStackTrace();

System.err.println(e.getClass().getName() + ": " + e.getMessage());

System.exit(0);

}

}

}

----

If we wanted to connect to the CockroachCloud server we created earlier, we'd issue the following command:

[source,bash]

----

$ java -m helloCRDB/helloCRDB.HelloCRDB postgresql://free-tier6.gcp-asia-southeast1.cockroachlabs.cloud:26257/defaultdb?sslmode=verify-full&sslrootcert=/Users/guyharrison/CRDBKeys/cc-ca.crt&options=--cluster=grumpy-orca-56 guy b4\_jPoEYw4\_Ixsj7

Hello from CockroachDB at 2021-05-05 15:39:07.667438+10:00

----

And here we connect to a local CockroachDB cluster in insecure mode:

[source,bash]

----

$ java -m helloCRDB/helloCRDB.HelloCRDB postgresql://localhost:26257/?sslmode=disable root ‘’

Hello from CockroachDB at 2021-05-05 15:38:56.691009+10:00

----

### Connecting to CockroachDB from Python

Python is a widely used scripting language as well as the tool of choice for many data scientists and data wranglers. In this example we'll use the +psycopg+ python-postgresql package to connect to CockroachDB.

To install the +psycopg+ package, issue the following command:

[source, bash]

----

$ pip3 install psycopg2

Collecting psycopg2

Using cached psycopg2-2.8.6.tar.gz (383 kB)

Building wheels for collected packages: psycopg2

Building wheel for psycopg2 (setup.py) ... done

Created wheel for psycopg2: filename=psycopg2-2.8.6-cp39-cp39-macosx\_11\_0\_x86\_64.whl size=136267 sha256=0c386372a9a001b321ab7b9a1d895d46ecb4b89dceb4af5cc3d641e8e62fc361

Stored in directory: /Users/guyharrison/Library/Caches/pip/wheels/a2/07/10/a9a82e72d50feb8d646acde6a88000bbf2ca0f82e41aea438a

Successfully built psycopg2

Installing collected packages: psycopg2

Successfully installed psycopg2-2.8.6

----

Now the following short program will connect to CockroachDB using a URL provided on the command line and issue a SELECT statement:

[source,python]

----

#!/usr/bin/env python3

import psycopg2

import sys

def main():

if ((len(sys.argv)) !=2):

sys.exit("Error:No URL provided on command line")

uri=sys.argv[1]

conn = psycopg2.connect(uri)

with conn.cursor() as cur:

cur.execute("""SELECT CONCAT('Hello from CockroachDB at ',

CAST (NOW() as STRING))""")

data=cur.fetchone()

print("%s" % data[0])

main()

----

Here we connect to a local CockroachDB cluster running in insecure mode:

[source, bash]

----

$ python helloCRDB.py 'postgres://root@localhost:26257?sslmode=disable'

Hello from CockroachDB at 2021-05-02 02:33:00.755359+00:00

----

And here we connect to the CockroachCloud database we established earlier in the chapter[[1]](#footnote-2):

[source, bash]

----

$ python helloCRDB.py 'postgres://guy:b4\_jPoEYw4\_Ixsj7@free-tier6.gcp-asia-southeast1.cockroachlabs.cloud:26257/defaultdb?sslmode=verify-full&sslrootcert=/Users/guyharrison/CRDBKeys/cc-ca.crt&options=--cluster%3dgrumpy-orca-56'

Hello from CockroachDB at 2021-05-02 02:39:55.859734+00:00

----

### Connecting to CockroachDB from Go

The GO language is one of the fastest-growing programming languages, which offers high performance, modern programming paradigms and a low footprint. Much of the CockroachDB database platform is written in Go, so Go is a great choice for CockroachDB development.

In this example, we are going to use the pgx PostgreSQL driver for Go to connect to a CockroachDB cluster. First, we need to install the driver:

[source, bash]

----

$ go env -w GO111MODULE=auto

$ go get github.com/jackc/pgx

go: downloading github.com/jackc/pgx v3.6.2+incompatible

go: downloading golang.org/x/text v0.3.6

go: downloading golang.org/x/crypto v0.0.0-20210421170649-83a5a9bb288b

----

This short program connects to CockroachDB using the URL provided on the command line, and issues a SELECT statement:

[source, go]

----

package main

import (

"context"

"fmt"

"os"

"github.com/jackc/pgx"

)

func main() {

if len(os.Args) < 2 {

fmt.Fprintln(os.Stderr, "Missing URL argument")

os.Exit(1)

}

uri := os.Args[1]

conn, err := pgx.Connect(context.Background(), uri)

if err != nil {

fmt.Fprintf(os.Stderr, "Unable to connect to database: %v\n", err)

os.Exit(1)

}

var text string

err = conn.QueryRow(context.Background(),

"SELECT CONCAT('Hello from CockroachDB at ',

CAST (NOW() as STRING))").Scan(&text)

if err != nil {

fmt.Fprintf(os.Stderr, "QueryRow failed: %v\n", err)

os.Exit(1)

}

fmt.Println(text)

}

----

Here, we connect to the CockroachCloud cluster we created earlier in the chapter:

[source, bash]

----

✗ go run helloCRDB.go "postgres://guy:b4\_jPoEYw4\_Ixsj7@free-tier6.gcp-asia-southeast1.cockroachlabs.cloud:26257/defaultdb?sslmode=verify-full&sslrootcert=$HOME/CRDBKeys/cc-ca.crt&options=--cluster=grumpy-orca-56"

Hello from CockroachDB at 2021-05-02 02:24:13.930662+00:00

----

And here we run the program to connect to a local CockroachDB cluster in insecure mode:

[source, bash]

----

✗ go run helloCRDB.go 'postgres://root@localhost:26257?sslmode=disable'

Hello from CockroachDB at 2021-05-02 02:21:59.179171+00:00

----

## Summary

In this chapter, we've shown you how to install CockroachDB software on a local computer, how to create a CockroachDB cluster in a variety of configurations and how to work with CockroachDB from the command line or a programming language.

It's easy to install CockroachDB software on a desktop and, in most cases, necessary if you want to work with a CockroachDB server from the command line. You can also install CockroachDB software using Docker or Kubernetes.

While a single-node test server can be a useful tool for learning CockroachDB, the CockroachCloud offers a free 5GB server that provides backup and security. You can also install CockroachDB in a Kubernetes cluster to experiment with a full cluster in a local environment.

Because CockroachDB is PostgreSQL compatible, you can use any Postgres compatible driver to connect to CockroachDB. We also provided simple examples of connecting to CockroachDB using the PostgreSQL drivers for Java, Python, GoLang and NodeJS.

1. Note that because of limitations in the psycopg2 driver, we need to replace the final “=” in the URL with “%3d. Instead of + cluster=grumpy-orca-56+ we use + cluster%3dgrumpy-orca-56+ [↑](#footnote-ref-2)