Implementing and testing CockroachDB on Kubernetes

*Albert Kasperi Iho*
Supervisors: Ignacio Coterillo Coz, Ricardo Brito Da Rocha

**Abstract**
This report will describe implementing CockroachDB on Kubernetes, a summer student project done in 2019. The report addresses the implementation, monitoring, testing and results of the project. In the implementation and monitoring parts, a closer look to the full pipeline will be provided, with the introduction of all the frameworks used and the biggest problems that were faced during the project. After that, the report describes the testing process of CockroachDB, what tools were used in the testing and why. The results of testing will be presented after testing with the conclusion of results.

**Keywords**
CockroachDB; Kubernetes; benchmark.
1 Project specification

The core goal of the project was taking a look into implementing a database framework on top of Kubernetes, the challenges in implementation and automation possibilities. Final project pipeline included a CockroachDB cluster running in Kubernetes with Prometheus monitoring both of them. Prometheus was implemented as a deployment within the same Kubernetes cluster as CockroachDB, and the scraped metrics were forwarded through an InfluxDB instance to Grafana for data visualization.

2 Implementation

The implementation phase of the project started by learning how to use OpenStack, which is the CERN virtual machine provider. After understanding OpenStack, the next step was to initialize a Kubernetes cluster and start playing with it, going through the tutorials given in CERN clouddocs and finally trying to set up a CockroachDB database running inside the cluster.

2.1 Kubernetes

Kubernetes is an open source system for managing containerized applications across multiple hosts; providing basic mechanisms for deployment, maintenance, and scaling of applications. It builds upon a decade and a half of experience at Google running production workloads at scale. [1] Kubernetes works with a range of container tools, with Docker as the default one. [2]
2.2 CockroachDB

CockroachDB is a distributed SQL database built on a transactional and strongly-consistent key-value store. It scales horizontally, survives disk, machine, rack and even datacentre failures with minimal latency disruption and no manual intervention. [3]

2.3 Setting up Kubernetes in CERN environment

After following the provided CERN tutorials on Kubernetes, the first big problem was faced on the second week of the project. The project had moved on to the database initialization phase, up to the point on initializing and starting pods, which would be the hosts for CockroachDB database. The pod initialization kept on failing and with weak understanding of Kubernetes and `kubectl events` as the only debugging tool, the conclusion was that the failing was due to problems in dynamic memory allocation. The project was stuck eight days on the suspected memory allocation problem, during which many solutions were investigated without any progress. One of the possible solutions that was looked into, was creating new Kubernetes StorageClasses from manually created volumes, which in theory could enable dynamic memory allocation in a personal project which is currently not supported in the CERN systems. The method involved creating a new volume in OpenStack, hardcoding the volume id into a StorageClass and setting the StorageClass as the default StorageClass. Kubernetes then would make PersistentVolumeClaims to the OpenStack volume or volumes, depending on how the StorageClass was configured.

The real problem was found after getting help to the PersistentVolumeClaim-problem, the reason behind the volumes not getting bound was that Kubernetes Certificates were not activated in the created Kubernetes cluster. In the creation of CockroachDB pods, the CockroachDB StatefulSet asks to get Certificates signed via CertificateSigningRequests, which can be approved without having Kubernetes Certificates enabled. This ends up with the Kubernetes resource ‘CertificateSigningRequests’ as approved, but no actions are taken towards enabling the functionality that the pod is asking to gain access to, which leads to the pods not being able to demand for a PersistentVolumeClaim, which means that the Kubernetes events log displays only the message ‘Unbound PersistentVolumeClaims’. Debugging of the problem was done by Ricardo Brito Da Rocha, who analysed the pod logs, which indicated that no steps towards getting memory to the pod were taken because the pods were not allowed to make the claims.

Solution to the first major problem was moving to a test environment from my personal project to gain access to CERN automatic dynamic memory allocation with the help of Manila, and changing from using the Kubernetes 1.14 image to enabling manually all the needed labels with the Kubernetes 1.13-3.2 image. This helped to understand everything that would be enabled in the cluster. After initializing the cluster with the proper image and labels, the activation of CockroachDB was very fast and the earlier problem was understood thoroughly.

3 Monitoring

The monitoring part of the project was added to the core parts after successful implementation. At first I didn’t understand why it was crucial, but after having the full information pipeline working the importance of easy monitoring was clear. With one glance to the Grafana dashboard I was able to see the state of the cluster, how much computing power my current actions were using, how much memory was still free in the database and much more. In the monitoring pipeline Prometheus was used to scrape information from CockroachDB and Kubernetes. The decision on using Prometheus was done based on it being already in use at CERN and also because CockroachDB automatically projects its metrics in Prometheus’ form. After scraping the data was written into an InfluxDB instance, which was used as a Data Source for Grafana. Prometheus could have also been used as a Data Source, but in this case most of the queries were already done in InfluxQL which would have also been used in production.
3.1 Prometheus
Prometheus is an open source systems monitoring and alerting toolkit originally built at SoundCloud. Since its inception in 2012, many companies and organizations have adopted Prometheus, and the project has a very active developer and user community. It scrapes samples from specified target jobs and the collected data can be easily visualized using an API consumer like Grafana. Prometheus joined the Cloud Native Computing Foundation in 2016 as the second hosted project after Kubernetes. [4]

3.2 Prometheus Operator
Prometheus Operator is an open source Operator from CoreOS. An Operator builds upon the basic Kubernetes resource and controller concepts but includes application domain knowledge to take care of common tasks. The Prometheus Operator should enable users to configure and manage instances of Prometheus using simple declarative configuration that will, in response, create, configure, and manage Prometheus monitoring instances. [5]

3.3 InfluxDB
InfluxDB is an open source time series database which is designed to handle high write and query loads and provides a SQL-like query language called InfluxQL for interacting with the data. [6] It allows for high throughput ingest, compression and real-time querying. InfluxQL supports regular expressions, arithmetic expressions, and time series-specific functions to speed up data processing. [7]

3.4 Grafana
Grafana is an open source metric analytics and visualization suite. It is most commonly used for visualizing time series data for infrastructure and application analytics. [8] Grafana supports many different storage backends for time series data, which are called Data Sources. Each Data Source has a specific Query Editor that is customized for the features and capabilities that the particular Data Source exposes. The query language and capabilities for each Data Source obviously differ a lot. [9]

3.5 Scraping and forwarding metrics from the Kubernetes cluster
On paper the reconfiguring of Prometheus is a really easy task, but it turned out to be almost an impossible task with the recommended installations. In the CERN cloud documents there is a guide on how to activate Prometheus during the initialization of a Kubernetes cluster, but that installation uses the Prometheus Operator, which is a different thing than a pure Prometheus. Enabling the Prometheus-label also made the cluster creation phase in OpenStack take much longer than normally, and almost half of the clusters created with the label failed. Realizing the difference between Prometheus and Prometheus Operator took longer than I would like to confess, even after my supervisor Ignacio explained it to me. A lot of time could have been saved if the documentation page would be named Prometheus Operator. During the rest of this report PO will be used to represent Prometheus Operator for clearer differentiation between Prometheus and Prometheus Operator.

The second big problem in the project was reconfiguring a Prometheus instance to scrape CockroachDB targets and to forward the data to the InfluxDB instance. Because of the use of PO at CERN, PO and Prometheus were understood to be the same thing. PO can be reconfigured by using their API, which can be found on GitHub. Unluckily any API commands didn’t have any effect on the configuration of PO, which might be due to the CERN installation of PO having prometheusConfigReloader enabled. Normally Prometheus is configured by a YAML-file inside the Prometheus container, which is mounted as a volume during initialization with the help of Docker. Mounting the volume turned out to be impossible, with the returning error being ’access denied’. The error was being thrown because the configuration file inside the Prometheus container was being written as root because Docker writes files
with root rights. This meant that there was no way of accessing the file inside the container, because
the root inside a container is actually user number 1000, which is just renamed to root in Docker source
code.

The problem actually ended coming from CentOS 7 settings and due to the work being done in
the CERN server, where Docker was not allowed to write. The virtual machines CentOS was enforcing
SELinux, which did not allow the overwriting of things inside docker. The solution into mounting a
volume as a configuration file to reconfigure Prometheus was to move into a temporary folder out of
the CERN server, and set SELinux from Enforcing to Permissive. After successfully mounting the
volume with the help of Borja Aparicio Cotarelo, I realized the difference between Prometheus and PO,
which lead to first looking into re-configuring PO and as a final option manually deploying Prometheus.
Re-configuring PO was started by manually adding ConfigMap-resources to try to mount the desired
configuration, which ended up with the configuration file being inside the Prometheus container but
not active. The only changes that went through on the existing ConfigMap resources were changes on
the global scraping timer, no changes via additional value files or manual additions to scrape targets
were successful. The changes were implemented by editing the POâ€™s Prometheus-operator-config
ConfigMap inside Kubernetes.

After giving up on trying to reconfigure PO and just implementing as bare Prometheus as possible,
my understanding of creating deployments and extensions on Kubernetes grew a lot. The final imple-
mentation was very light, and configuration was done by mounting a ConfigMap as a volume inside the
Prometheus container. This time the ConfigMap was easy to overwrite and redeploy, due to me under-
standing everything that was going on and the amount of pods being minimal. Initially the Prometheus
was set to scrape targets based on their IP inside the cluster, which ended up as a bad choice, because the
CockroachDB database nodes get a new IP every time they are restarted. The final configuration scrapes
the pods according to name, which makes every node their own job in Grafana, making the monitoring
of differences really easy. The full documents on the implementation and configuration can be found
at [10].

4 Testing CockroachDB on Kubernetes
After the full pipeline was implemented and working, the testing of CockroachDB’s capabilities was
started. The aim of the tests was to benchmark the performance of CockroachDB with comparable
results. Testing was done with three different tools. The tools were python using the SQLAlchemy
toolkit for database access, pgbench and CockroachDB’s inbuilt workload-function.

4.1 SQLAlchemy
SQLAlchemy is a Python SQL toolkit and Object Relational Mapper that provides a full suite of well-
known enterprise-level persistence patterns, designed for efficient and high-performing database access,
adapted into a simple and Pythonic domain language. [11] To find the fastest insertion method from
SQLAlchemy, four different methods of insertion were compared before starting the real tests on Cock-
roachDB. The fastest method was the bulk_save_objects-function from SQLAlchemy. [12] SQLAlchemy
was used because CockroachDB supports and has guides on developing applications with it. [13]

4.2 Pgbench
Pgbench is a simple program for running benchmark tests on PostgreSQL. It runs the same sequence of
SQL commands over and over, possibly in multiple concurrent database sessions, and then calculates
the average transaction rate (transactions per second). By default, pgbench tests a scenario that is loosely
based on TPC-B, involving five SELECT, UPDATE, and INSERT commands per transaction. [14] The
reason behind using pgbench is to gain easily comparable results on the transaction speeds of tested
databases.
4.3 CockroachDB workload

CockroachDB comes with built-in load generators for simulating different types of client workloads, printing out per-operation statistics every second and totals after a specific duration or max number of operations. There are six different workloads that can be simulated. [15] The CockroachDB workload was used because it was believed to provide benchmarking data from the database.

4.4 Benchmarking CockroachDB

The first tests were done by using a simple SQLAlchemy script to create users and insert them into the database. Three different insertion tests were done, first with only one concurrent connection, then with two concurrent connections, and the final was performed with four concurrent connections. All tests were ran for 30 minutes and 10 times to gain a large enough set to calculate results from. The second tests were done with pgbench, the goal was to find out how many concurrent connections could exist before the frameworks transaction per second rate starts decreasing. To enable pgbench, the tables inside the database had to be manually created, due to CockroachDB’s SQL dialect differing from the PostgreSQL’s one, which meant that the initialization method from pgbench could not be used. The solution was to create a SQLAlchemy script for database initialization and deletion to keep the test results comparable. Finally the third tests were done using the Bank-workload from the Cockroach client. Broader documentation including the code used in testing can be found at [12].

5 Test results

5.1 SQLAlchemy

From the first tests done on SQLAlchemy the following results were gained:

<table>
<thead>
<tr>
<th>Concurrent connections</th>
<th>Mean of insertions/second</th>
<th>Variance of insertions/second</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>450.0</td>
<td>44.092</td>
</tr>
<tr>
<td>2</td>
<td>484.375</td>
<td>3202.546</td>
</tr>
<tr>
<td>4</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 1: Results from testing insertion speeds on CockroachDB

Where both mean and variance are averages over 10 runs. The test with four concurrent connections was never able to finish due to the testing script not having error checking. The error checking should have been done on the testing client because CockroachDB performs concurrency control optimistically [16], which means that the testing client will crash if transactions with retries are not used [12].

5.2 Pgbench

The optimistic concurrency control also hindered pgbench from being able to be used with more than one concurrent connection. The following results were gained from one test run of pgbench:

<table>
<thead>
<tr>
<th>Database</th>
<th>Number of transactions</th>
<th>Transactions per second</th>
</tr>
</thead>
<tbody>
<tr>
<td>PostgreSQL</td>
<td>100000</td>
<td>744.245</td>
</tr>
<tr>
<td>CockroachDB</td>
<td>1000</td>
<td>26.230</td>
</tr>
</tbody>
</table>

Table 2: Results from testing transaction speeds with pgbench CockroachDB and PostgreSQL.

The transactions per second includes connection establishing. It is understood that during testing number of transactions should be the same for comparable results, but in this case it can be seen that the classic PostgreSQL database was almost 30 times faster than CockroachDB, with hundred times larger workload. It was also seen that the speed of transactions per second would decrease when the number of
transactions increased, which means that PostgreSQL with a similar workload as CockroachDB, would be over 30 times faster.

5.3 CockroachDB workload
The third test was an attempt at running the inbuilt workload-function of the CockroachDB client. The CockroachDB cluster that was implemented on Kubernetes was set to secure state, which meant that connection-strings needed passwords, and using certificates to connect as a root user was not possible. To run the Bank-workload, first it needed to be initialized, which was not possible in a secure cluster due to the initialization script trying to use CREATE DATABASE- command, which is only restricted to super-users in PostgreSQL. Unluckily the GRANT ROLE-command is an enterprise feature in CockroachDB, which meant that the only superuser available was the root user, which could not be used to connect into the database, so the only option was to manually initialize the database. The database schema was found from the CockroachDB GitHub repository [17], so the manual initialization was possible. After the manual initialization, the bank-workload was coded to use SHOW CLUSTER SETTING-command, which unluckily is also only restricted to superusers.

5.4 Conclusion
In conclusion to perform broad and comparative benchmarking of CockroachDB was not possible with the current tools used. This has already been conversed on the CockroachDB GitHub page [18], where the answer was "...you should check out our workload tool." [19], which can only be ran with an active enterprise licence.

6 Acknowledgements
Big thank you to my supervisor Ignacio Coterillo Coz for help on multiple problems, exemplary leadership and granting me a lot of freedom while working.
Thank you to the team, IT-DB-IA, for the great work environment and support.
Thank you to my supervisor Ricardo Brito Da Rocha for help on the Kubernetes image and labels problem.
Bibliography

[1] Kubernetes GitHub repository,
https://github.com/kubernetes/kubernetes/

[2] Kubernetes development blog,
https://kubernetes.io/blog/2018/10/10/kubernetes-v1.12-introducing-runtimeclass/

[3] CockroachDB frequently asked questions,

[4] Prometheus documentation,
https://prometheus.io/docs/introduction/overview/

[5] Prometheus Operator documentation,
https://coreos.com/blog/the-prometheus-operator.html

[6] InfluxDB overview,
https://www.influxdata.com/products/influxdb-overview/

[7] InfluxQL overview,
https://www.influxdata.com/time-series-platform/

[8] Grafana overview,
https://grafana.com/docs/v4.3/

[9] Grafana basic concepts,
https://grafana.com/docs/v4.3/guides/basic_concepts/

[10] Documentation of Implementing CockroachDB on Kubernetes,
https://codimd.web.cern.ch/s/HybblSufH

[11] SQLAlchemy homepage,
https://www.sqlalchemy.org/

[12] Documentation of Testing CockroachDB on Kubernetes,
https://codimd.web.cern.ch/s/BkSShRAGr

[13] Tutorial on building an application with SQLAlchemy,
https://www.cockroachlabs.com/blog/building-application-cockroachdb-sqlalchemy-2/

[14] PostgreSQL pgbench documentation,
https://www.postgresql.org/docs/9.1/pgbench.html

[15] CockroachDB workload documentation,

[16] CockroachDB forum post on transactions,
https://forum.cockroachlabs.com/t/insert-failed-on-transaction/831/7

[17] CockroachDB bank workload source code,

[18] CockroachDB comparative benchmarking thread,
https://github.com/cockroachdb/cockroach/issues/24217

[19] CockroachDB on comparative benchmarking,
https://github.com/cockroachdb/cockroach/issues/24217#issuecomment-376386052