Waterford Institute *of* Technology

Cloud Computing 1 - Term Paper

Dimitri Saridakis

Work submitted
within the
BSc in Applied Computing
Cloud & Networks

Lecturer: Richard Frisby

October 31, 2021

Contents

List of Figures

1 Initial Exploratory Investigation

1.1 System Goals and Technologies

For this term paper I have decided to try and come up with a system that is both highly available and is highly scalable. A system that is capable of reacting - in real time - to changes i. e any of the CRUD operations in a database. Upon these change events occurring, the changes to the data in the system which triggered the events should be made available to other service components of the system to allow for the processing and the business goals of the system to be met. To aid with accomplishing these requirements the system I have chosen to design is an event driven, microservice architectured design with Debezium looking after the change events whilst the deployment orchestration managed by Kubernetes, in this case I'll be using a single Kubernetes node on my machine by utilizing Minikube(*Minikube Start* n.d.). These are the primary technologies around which this term paper will focus.

1.1.1 Non-Goals and Wants

Some non-goals/ nice to have goals if time goes my way, I would like to write a Kafka consumer to consume the changes from the event stream(Kafka topics). To further aid in hitting the system business goals I would also like to incorporate as much monitoring of the system as I can (time permitting) as detailed monitoring of the system is crucial in keeping the system working in an optimized fashion which then means that the system is more easily managed and hence scalable and highly available. For this I intend on exploring the deployment of a service mesh, probably using Istio into my Kubernetes environment. Whilst this may seem like a lot, and I probably won't get to the implementation in time. I just wanted to outline my thought process in the design of a system and how I can leverage the areas covered in this module to have a real world application use.

1.1.2 Debezium

Debezium is an open source Change Data Capture (CDC) technology which is applied to continuously monitor your database via its connectors. It lets any of your applications/ services stream every row-level change in the same order that the changes were committed to the database (Debezium Community n.d.) This will take the form of Apache Kafka streams to Kafka topics. These Kafka topics can then be 'consumed'

by any number of services that simply subscribe to the topics. These consumers (services) will then carry out the business logic on the changes required to meet the systems business goals.

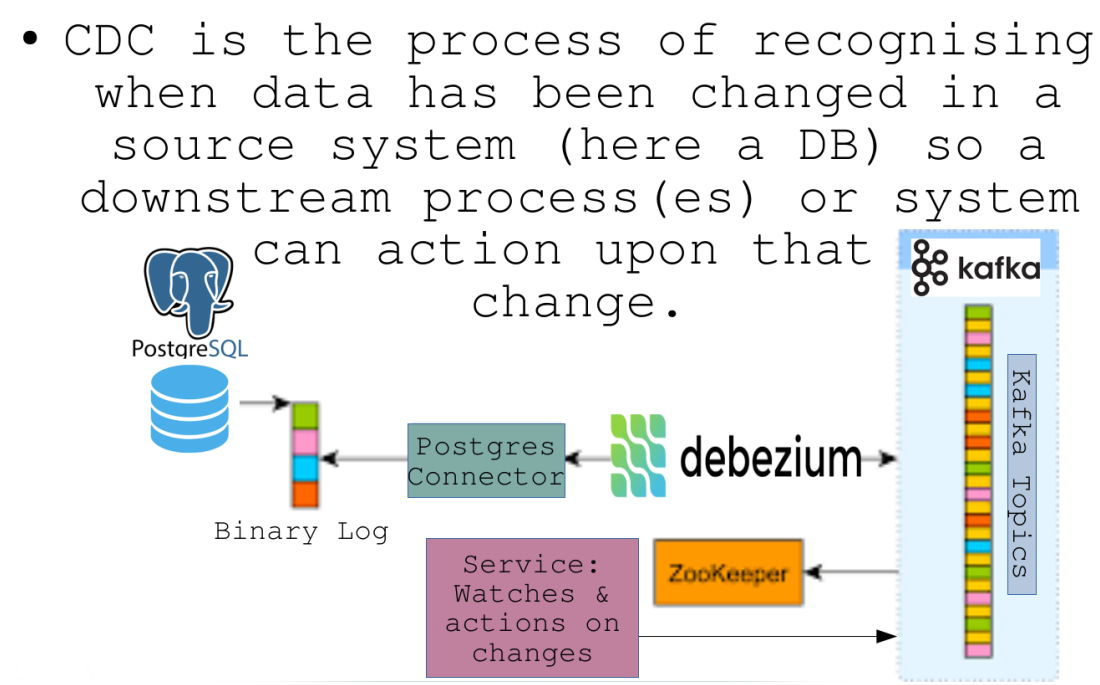

Figure 1.1: Architecture of the system, taken from the proposal slide

1.2 Interim Report

I've left in the preceding section from the initial proposal for context. This section is written on the 31/10/21. It is just a succinct summary as to what I have done so far.

1.2.1 Minikube

As this is pretty much all Kubernetes based I have configured a Minikube single node cluster for the work that I hope to complete.

1.2.2 Data

Since this report will focus on capturing changes from a database I will have to have a source of data to save to the database. For this I am going to use an API that I am writing. It is essentially a Transaction API, it exposes an endpoint and which creates a fake transaction. This transaction is a typical transaction you'd find in a pub/ bar. It has Transaction owner, this is the entity for which the transaction has been possessed, in this case a pub/ bar. The transaction contains a random number of beers with some randomized information for things like name, ABV, price etc. I may have to tune down the amount of randomness in the API like only having N amount of transaction owners or selection of beers to choose from. The API is written in GoLang and uses Gorilla Mux(a high performance HTTP router package).
I've 'dockerized' the API and can now deploy it on Minikube with successfully being able to expose the application to external(from K8s env) interaction. Furthermore, I haven't written up how I have done that yet, but I do have all the commands saved, along with having all the Kubernetes manifests that I wrote to get the API to deployment.

1.2.3 PostGres

I've thus far used a PostGres container for my database that is based off of the newest PostGres image from docker hub. Looking into the Debezium PostGres connector I've learned quite a bit about how it works. How the connector reads from the *write-ahead log*(WAL), "which describes the changes on a storage level, into an application-specific form such as a stream of tuples or SQL statements" (*Logical Decoding Output Plug-in Installation for PostgreSQL :: Debezium Documentation* n.d.).
The PostGres encodes to 'pgoutput' by default, a format I know very little about. Debezium does also support JSON, via the installation of a *logical decoding output* plugin, which would be my preferred format to work with. This is going to require some additional configuration to the PostGres database itself. The database is going to need altering anyway to work with the connector, so I'm going to try and reconfigure the PostGres image used in the deployment to include the `wal2json` (the specific logical decoding output plugin for json) plugin too. I've gone through some of the steps

I think are needed for it but have not yet written it out.

1.2.4 Apache Kafka

Debezium writes the changes it captures from the database to an Apache Kafka topic. This means I will obviously have to implement a Kafka cluster in my Minikube environment. I will need:

- Zookeeper
- Kafka
- Kafka Entity Operator
- Kafka Connect

There are a number of differing ways to deploy Kafka, but I've narrowed it down to two ways for now. I can deploy each component on its own, or I could go down the Kubernetes Operator route. There is an open-source Kafka Operator available called Strimzi (*Strimzi Documentation (0.16.2)* n.d.) and I have so far looked into taking this route. I have managed to successfully deploy the required Kafka components by installing the Strimzi Operator on my cluster in its own 'Kafka' namespace, but I'm unsure if I will use it. Strimzi appears to hide away some of the configuration of Kafka and I feel like I'm not in as much control over the components as I would like to be. This may change as I've had very little time to play with the system between Strimzi deployment and the writing of this interim report.

Bibliography

[1] Debezium Community. *Debezium*. https://debezium.io/.

[2] *GOCDC and Postgres*. https://dev.to/thiagosilvaf/gocdc-and-postgres-1m4m.

[3] *Gorilla/Mux*. Gorilla Web Toolkit. Oct. 2021.

[4] *How to Integrate Kafka with Istio on OpenShift*. https://labs.consol.de/development/2021/02/02/istio

[5] *How to Use Change Data Capture (CDC) with Postgres*. https://dev.to/thiagosilvaf/how-to-use-change-database-capture-cdc-in-postgres-37b8.

[6] *Lessons Learned from Running Debezium with PostgreSQL on Amazon RDS*. https://debezium.io/blog learned-running-debezium-with-postgresql-on-rds/.

[7] *Logical Decoding Output Plug-in Installation for PostgreSQL :: Debezium Documentation*. https://debezium.io/documentation/reference/postgres-plugins.html.

[8] *Minikube Start*. https://minikube.sigs.k8s.io/docs/start/.

[9] *Strimzi Documentation (0.16.2)*. https://strimzi.io/docs/0.16.2/.

[10] *The 7 Best CDC Tools (Change Data Capture) - Learn — Hevo*. https://hevodata.com/learn/7-best-cdc-tools/.

[11] *Using Strimzi*. https://strimzi.io/docs/operators/latest/full/using.html.