Scalable Real Time Taxi Application

Amith Gopal, Akriti Kapur, Tarunianand Muruganandan, Sowmya Ramakrishnan

University of Colorado, Boulder

1. Introduction

Recently, we see more and more industries shifting towards a microservice architecture. As mentioned in [2], these cloud-native architectures can facilitate migrating on-premise architectures to fully benefit from the cloud environments because non-functional attributes, like scalability, are inherent in this style. The aim of our research project is to explore this novel microservice architecture, document the challenges we face and evaluate the architecture's performance.

For small systems with fewer complexities a monolithic architecture is well suited as scalability can be achieved by a simple load balancer. However, as this system becomes large, problems tend to appear with data-intensive loads, large query time and increased time in deployments [6, 7]. A microservice architecture works better in this case as big systems could be divided into smaller services being deployed and managed independently.

We needed a domain with data complexities that required scaling, for us to use a microservice architecture. Taxi service is the perfect domain as it provides the required complexity in terms of the features like latitude, longitude used to express location and also in terms of querying the large data space to say, find all the taxis in a particular area.

In our research, we explore running queries on a Spark pod [4] on a Kubernetes cluster [3], by deploying a MongoDB server on the same Kubernetes cluster [3], a relatively unexplored combination according to our knowledge. We document the challenges we faced while converting the problem space to use a microservice architecture. We then compare the results by employing our architecture with the results obtained by querying directly from the database and querying using a standard Spark job alone. We found that running Kubernetes on Spark does increase the performance of our application as compared to other architectures we evaluated.

We go through previous work done on Kubernetes, then talk about our work and how it compares to the previous work done. We then go through the system design and implementation in depth , the results of our implementation and future work in this domain.

Most user facing applications are developed with a user interface to ensure that there is maximum user satisfaction and usability. Popular applications like that of taxi services have an extra job of ensuring that the application is scalable to their expanding user base, and additionally handling the larger and more frequent user requests. This further increases an application's monetary returns.

This project thus serves as an attempt to emulate an application of this kind, and in turn deliver scalability features similar to the real world.

2. Related Work

The popularity of containers is fairly recent. [5] describes some of the knowledge gained and lessons learned during Google's journey from Borg to Kubernetes, and explains our choice - to go with Kubernetes. It is a combination of microservices and small control loops and is open-source, highly available, conformant and configurable. It allows for higher-level versioning, validation, semantics and policy, and is in support of a more diverse array of clients. Kubernetes makes it easy to deploy and manage complex distribution systems and handle multiple requests parallely, which directly relates to our scalable application.

[10] is an interesting visualization application, of a random NYC taxi on a single day in the year 2013. This proved to be our initial inspiration and motivation for the project as well as for visualization. While this application depicted the journey of a single taxi on a busy day in New York, we wanted to make our application more scalable and real-time, and closer to emulation of an existing taxi service like Uber. The visualization looked cool and we felt, even at first glance, that this was something we wanted our application to look like. Although it was a stretch goal initially, we decided to make it a part of the project as this is the best way to demonstrate and showcase how tasks like finding taxis are executed.

Our choice of dataset was the 2014 Green Taxi Trip Data [11]. This was an obvious choice since it included the most complete data we needed and could acquire - with fields such as pick-up and drop-off dates and times, locations in terms of latitude and longitude, trip distances and passenger counts, among others.

The issue of landing the perfect query to use on the data was solved with the help of a Reddit thread (the Taxi Data BigQuery thread) by a user who called himself u/fhoffa, wherein he explains import of the Green Taxi data into Google's BigQuery. [11] The thread consists of numerous queries on the data, which helped us with the custom query we needed to pull full days of data for random taxis.

We then came to a point where the resultant queried data was in GeoJSON format (encoding of geographical data structures). This needed to be turned into an animated map, and Mike Bostock's D3 + Leaflet [13] was a godsend. This uses APIs which helped render the map, project points and translate coordinates between screen pixels and latitude/longitude, i.e., turning geoJSON into an SVG layer. Mapbox, a source of hosted tiles that enables easy stylization of a

map, helped us choose a simple subdued basemap tile (so the background does not call too much attention to itself). It also had a node package for conversion of Google-encoded polylines into raw coordinates, which was a boon. [14], a technique which involves tweening (transitioning the shape/position of SVG elements), helped with the challenge of making an animated cab marker leave a trail on the map.

3. System Design

3.1 Dataset

We chose 2014 Green Taxi Trip Data [11] for our work. This dataset consists of the columns mentioned in *Figure 1* below

Column Name	Description	Type
vendorid	A code indicating the TPEP provider that provided the reco...	Plain Text
pickup_datetime	The date and time when the meter was engaged.	Date & Time
dropoff_datetime	The date and time when the meter was disengaged.	Date & Time
Store_and_fwd_flag	This flag indicates whether the trip record was held in vehi...	Plain Text
rate_code	The final rate code in effect at the end of the trip. 1= Stand...	Number
Pickup_longitude		Number
Pickup_latitude	Latitude where the meter was engaged.	Number
Dropoff_longitude	Longitude where the meter was disengaged.	Number
Dropoff_latitude	Latitude where the meter was disengaged.	Number
Passenger_count	The number of passengers in the vehicle. This is a driver-e...	Number

Figure 1: Columns present in Green Taxi database

3.2 Architecture and System Components

The system design consists of the following components:

1. **Kafka Message Queue Unit**:
   Kafka [11] is used as a messaging service in our application. Every request received by the application for a user is added to the Kafka queue waiting to be processed by a service. The main motivation behind this approach was that the requests could be

categorized into Kafka topics and each topic could be handled parallelly by an independent service for scalability. Some examples of topics are user authentication, get nearby rides etc.

2. **Django Framework:**
   Django [12] is a framework used to build web applications. We decided to use this particular framework as it provides API, template, object-relational mapping layer (ORM) support and was perfect for the medium sized application we are trying to build with Python as the main developing language. We used Django templates to create a user interface (UI) to search for available rides. All the user requests from the user interface are fed into the Kafka queue which are then directed to the appropriate service. The results from this service are again received by Django and sent to the user. This data is also used to support visualizations as we will show later in our results demonstration.

3. **Amazon EKS:**
   We are using Amazon's Elastic Container Service for Kubernetes (Amazon EKS) to deploy, manage, and scale our applications using Kubernetes on AWS. We chose EKS as it is strongly integrated with different AWS services, is highly secure and it is a managed service that makes it easy to run Kubernetes on AWS without needing to maintain our own Kubernetes control plane. Another advantage with EKS is high availability - to eliminate a single point of failure, the Kubernetes infrastructure is is run across multiple AWS availability zones.

4. **Spark:** We are using Spark [4] to perform queries on MongoDB. We chose Spark since it is meant for fast computation (increased processing speed due to in-memory cluster computing technology) and is relatively easy to implement. Additionally, the 2.3.0 release of Spark supports a new kubernetes scheduler backend, allowing us to submit spark applications to a kubernetes cluster, which is directly in line with our methodology/goals.

5. **MongoDB**: Apart from the various advantages it offers, like being document-oriented, simple and fast, we went with MongoDB as the database because of the availability of detailed documentation and community support. (This proved helpful when we had to integrate the database with Django.) Our initial choice was Amazon's DynamoDB, but that proved to be relatively new, with not much documentation and hence difficult to implement.

Figure 2 depicts how the described components interact with each other.

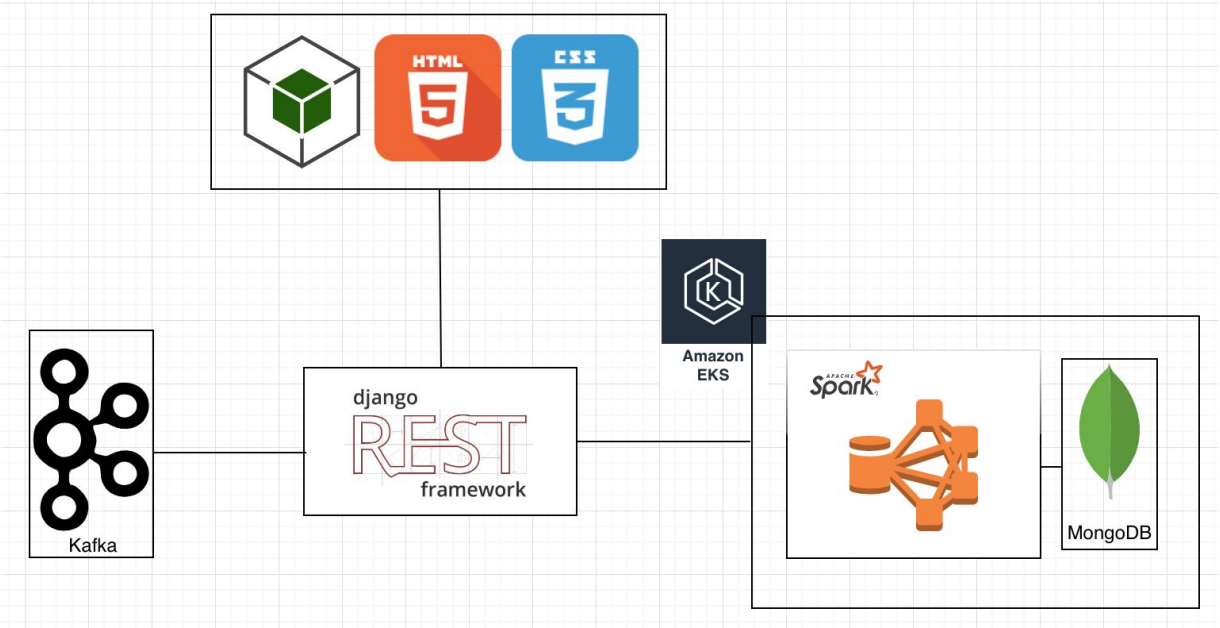

Figure 2: Microservice Architecture diagram

Interaction between the components

The requests from the Kafka messaging service is picked up by the Django Rest Framework based on the topics associated with each request. The processing is later off-loaded to a Spark job which runs on a Kubernetes cluster. The Spark service created queries the MongoDB server which is also setup in the Kubernetes cluster. The result obtained from the Spark job is later used by the Django framework for visualisation.

3.3 Challenges

I. Setting up Spark and MongoDB on EKS
   A. Setting up and running Spark and MongoDB on EKS was filled with complexities which made it a non-trivial task. This was due to the limited resources available online for debugging. We spent a considerable amount of time in the setup and process of trying to make this work.
   B. Certain features were not available on Kubernetes due to which some workarounds needed to be made. Some features like running spark jobs using pyspark on Kubernetes is not yet supported which required us to work with Java and ensure dependencies were met by adding many jar files.

II. Database Migrations

A. Django model needed to be created and a script needed to be written for copying the data from our original database to the new one. This was needed so that we could perform queries using the inbuilt Django ORM which would make things like visualizations, any data manipulations or get, post, put, delete calls easier.

4. Evaluation and Findings

Through our work, we aimed to explore and justify using a microservice architecture. We were successfully able to set up a real-time and scalable taxi service.

We base our evaluation on the following hypothesis:
Hypothesis H1: Using Spark on Kubernetes for performing queries is faster than performing queries using spark alone or querying MongoDB only.

4.1 Experimental Setup

We queried to "*find all the taxis near a pickup location*". The query prioritized taxis in order of decreasing time and distance from the pickup location.

The simplified query used was,

Select rides where,
(pickup_latitude - δ_{lat} <= ***ride_latitude*** <= pickup_latitude + δ_{lat})
and
(pickup_longitude - δ_{lng} <= ***ride_longitude*** <= pickup_longitude + δ_{lng})
and
(within $\delta_{minutes}$ away from pickup_location)

Rides ordered by time away from the pickup location

For the experiment, we had set,

δ_{lat}, latitude threshold = 0.02
$\delta_{lng,}$ longitude_threshold = 0.02
$\delta_{minutes,}$ threshold_minutes = 10 minutes

We tested for the following three conditions:

a) Use Django ORM to execute the query
b) Use Spark to query MongoDB
c) Use Spark deployed on Kubernetes to Query

Table 1 shows the figures that support this hypothesis. The average time has been found by taking the average over 10 runs.

Scenarios	Average Time (seconds)
Query MongoDB directly	~25s
Django ORM to execute the query	~82s
Use Spark to query MongoDB directly	~73s
Use Spark deployed on Kubernetes to Query - where number of executors used is 1	~77s
Use Spark deployed on Kubernetes to Query - where number of executors used is 2	~47s
Use Spark deployed on Kubernetes to Query - where number of executors used is 3	~44s

Table 1: Evaluation Results

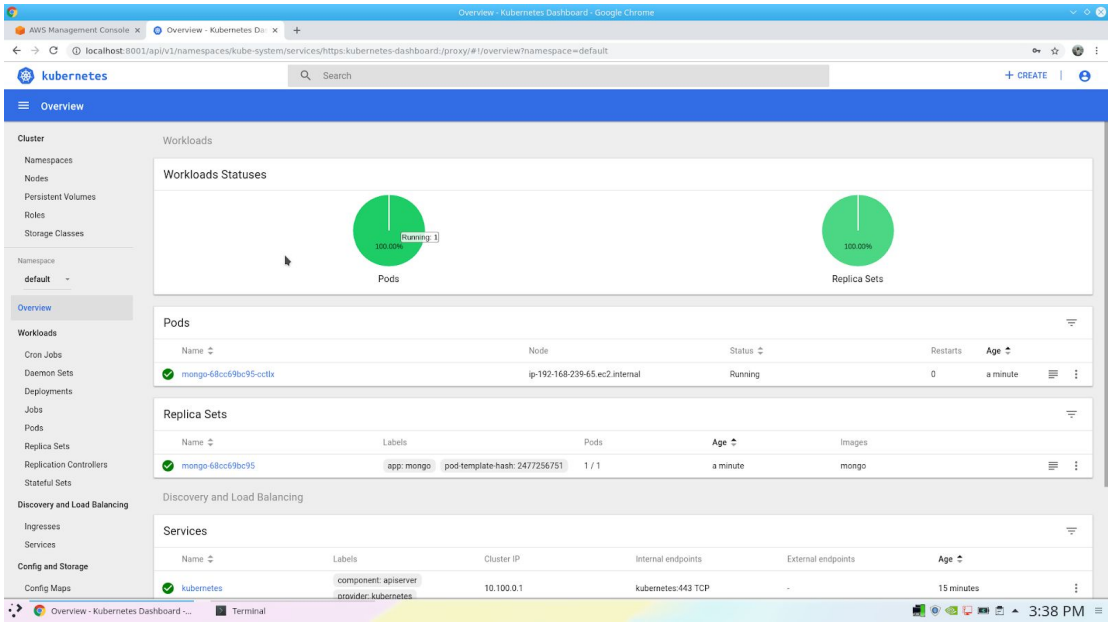

Figure 3: Kubernetes Dashboard showing running Mongo Pod

The application has been demonstrated with a working user interface here. It consists of adding a source and destination to the query, followed by clicking on the "Search for Rides" button, which results in the appearance of nearest available cars. The demo can be found on the link - demo.

Spark submit command:

```
spark-submit --master k8s://https://192.168.99.100:8443  \
--deploy-mode cluster \
 --name taxi-app \
--class TaxiApplication  \
--conf spark.executor.instances=2  \
--conf spark.kubernetes.container.image=amgodocker/spark:0.1
https://s3.amazonaws.com/out-transform/hp-1.0-SNAPSHOT.jar
```

The parameters in the above command are:

1. **TaxiApplication** : The Java file containing logic for queries on the MongoDB database
2. **k8s://https://192.168.99.100:8443** : URL of the Minikube cluster
3. **amgodocker/spark:0.1** : Spark container image used for running spark applications in a pod
4. **https://s3.amazonaws.com/out-transform/hp-1.0-SNAPSHOT.jar** : Jar file containing the TaxiApplication java file.

5. Review of Team Member Work

Team Member	Contributions	Timeline
Amith Gopal	**Worked on AWS EKS, minikube, Setting up and running Spark and mongoDB on Kubernetes, Integration of mongoDB with Spark** ● Reading up on EKS and setting it up ● Minikube setup ● Running spark on EKS/Minikube ● Lots of debugging and research on the best possible way to set up Spark and Mongodb on Kubernetes ● Running mongodb on EKS/Minikube ● Spark-Mongodb integration ● Evaluation with different spark cluster sizes	Setting goals for our project and design discussion : **Initial weeks** Reading about EKS, Reading about Spark. Installation of Spark, AWS CLI, Kubectl. Setting up Spark on Kubernetes :- **November 1st - November 15th** Spark successfully working on EKS. However the cost of EKS was a lot, so we replicated the setup with minikube. However, there were a lot of issues which required lot of debugging. Researching on different ways to deploy MongoDB on Kubernetes **:- November 16th - December 4th** Deployed MongoDB on Kubernetes. Tried lots of approaches to deploy MongoDB. After deploying MongoDB server, integration of spark with MongoDB. Evaluation with multiple cluster sizes of Spark in the Kubernetes cluster.:-**December 5th - December 13th**

Akriti Kapur	**Worked on Django ORM models, querying, visualizations and Kafka** ● Preprocessing pipeline implemented ● Project structure added - dependencies, structure, settings etc ● Integrating MongoDB with Django ● Creating Database models in Django - Django models correspond to database tables ● Migrating data to the model created through Django ● Integrating Kafka with the application, producer and consumer test code added for Kafka ● Creating a Readme on instructions on how to get the application running with installation Instructions. ● Visualization using Google Maps ● Creating/executing a query to find all nearby rides ● Assisted in evaluations	Design discussions: **Initial weeks October 5th and before** Preprocessing, Project setup, structure: **Week of November 12** Django Orm Models and Integration with MongoDB: **Weeks of November 19 and November 26** Kafka Processing Unit**: Week of November 26** Querying and Visualization for evaluation: **Week of December 3**
Tarunianand Muruganandan	**Worked on Django framework, application design and research**	Design discussions: **Initial weeks**

	● Found workable database to work on. ● Read up on Spring boot and other RESTful web APIs that can be used for the project. ● Finalized Django as the framework to be used, weighing the pros and cons with other frameworks. ● Built initial app framework ● Attempted to emulate real time data test cases ● Researched on some visualization tools and APIs to represent taxi travel on a map with Django. ● Worked on initial MapBox API, and then switched to Google maps ● Read up on Kafka-Django integration	Framework specifications and research: **Week of November 12** Framework build: **Week of November 19** Dynamic data emulation trial, Kafka integration readings**: Week of November 26** Visualization for evaluation: **Week of December 3**
Sowmya Ramakrishnan	**Worked on AWS EKS, minikube, Setting up and running Spark and mongoDB on Kubernetes** ● Learning Kubernetes and associated concepts ● Setting up Kubernetes cluster on Minikube and AWS EKS ● Running sample spark job on EKS/Minikube	Design discussions: **Initial weeks** Reading up on, learning and implementing Kubernetes concepts: **Week of November 12** Setting up Kubernetes cluster on EKS and Minikube, running sample spark application on EKS: **Week of November 19**

	• Lots of debugging and research on the best possible way to set up Spark and Mongodb on Kubernetes • Running highly-available mongodb on EKS using Portworx, a cloud native storage platform designed to run persistent workloads on engines like kubernetes. • Spark-Mongodb integration	Debugging Spark on EKS, Running sample spark application on Kubernetes cluster spun up through Minikube: **Week of November 26** MongoDB on Kubernetes (AWS EKS) and Integration of Spark and MongoDB: **Week of December 3** Deployed MongoDB on Kubernetes. Tried lots of approaches to deploy MongoDB. After deploying MongoDB server, integration of spark with MongoDB: **Week of December 13**

Table 2: Team Member Contributions

6. Conclusion

Through our work, we were successfully able to put a complete microservice architecture together with a complex taxi service domain. We were able to evaluate our design and came to the conclusion that,

Running Spark on Kubernetes led to faster response time

Our evaluations show that this was the most efficient way out of the methods we compared with.

Setting up Spark on MongoDB is non-trivial

As we mentioned in Section 5 - challenges, Setting up Spark and MongoDB was a difficult task and requires experience. We aim to contribute to the open source community by sharing our experience and implementation steps.

We were also able to put up one basic functionality of a taxi service to find rides. We hope to build on this and make it better. We were also not able to fully evaluate the scalability of our application by testing the performance with multiple user requests. This would also be a part of future work in our application.

7. References

[1] Roles, Decoupling Operations. "How Kubernetes Changes Operations." Operating Systems and Sysadmin: 36.

[2] Balalaie A., Heydarnoori A., Jamshidi P. (2016) Migrating to Cloud-Native Architectures Using Microservices: An Experience Report. In: Celesti A., Leitner P. (eds) Advances in Service-Oriented and Cloud Computing. ESOCC 2015. Communications in Computer and Information Science, vol 567. Springer, Cham

[3] https://kubernetes.io/docs/concepts/

[4] https://spark.apache.org/docs/latest/

[5] Burns, Brendan, et al. "Borg, omega, and kubernetes." Queue 14.1 (2016): 10.

[6] Balalaie, Armin, Abbas Heydarnoori, and Pooyan Jamshidi. "Migrating to cloud-native architectures using microservices: an experience report." European Conference on Service-Oriented and Cloud Computing. Springer, Cham, 2015.

[7] Richardson, C.: Microservices architecture. http://microservices.io/ (2014), [Last accessed 15-June-2015]

[8] https://kafka.apache.org/

[9]https://docs.djangoproject.com/en/2.1/

[10] http://chriswhong.github.io/nyctaxi/

[11] https://data.cityofnewyork.us/Transportation/2014-Green-Taxi-Trip-Data/2np7-5jsg

[12]https://www.reddit.com/r/bigquery/comments/28ialf/173_million_2013_nyc_taxi_rides_shared_on_bigquery/

[13] https://bost.ocks.org/mike/leaflet/

[14] http://bl.ocks.org/KoGor/8163022