29 APRIL 2018

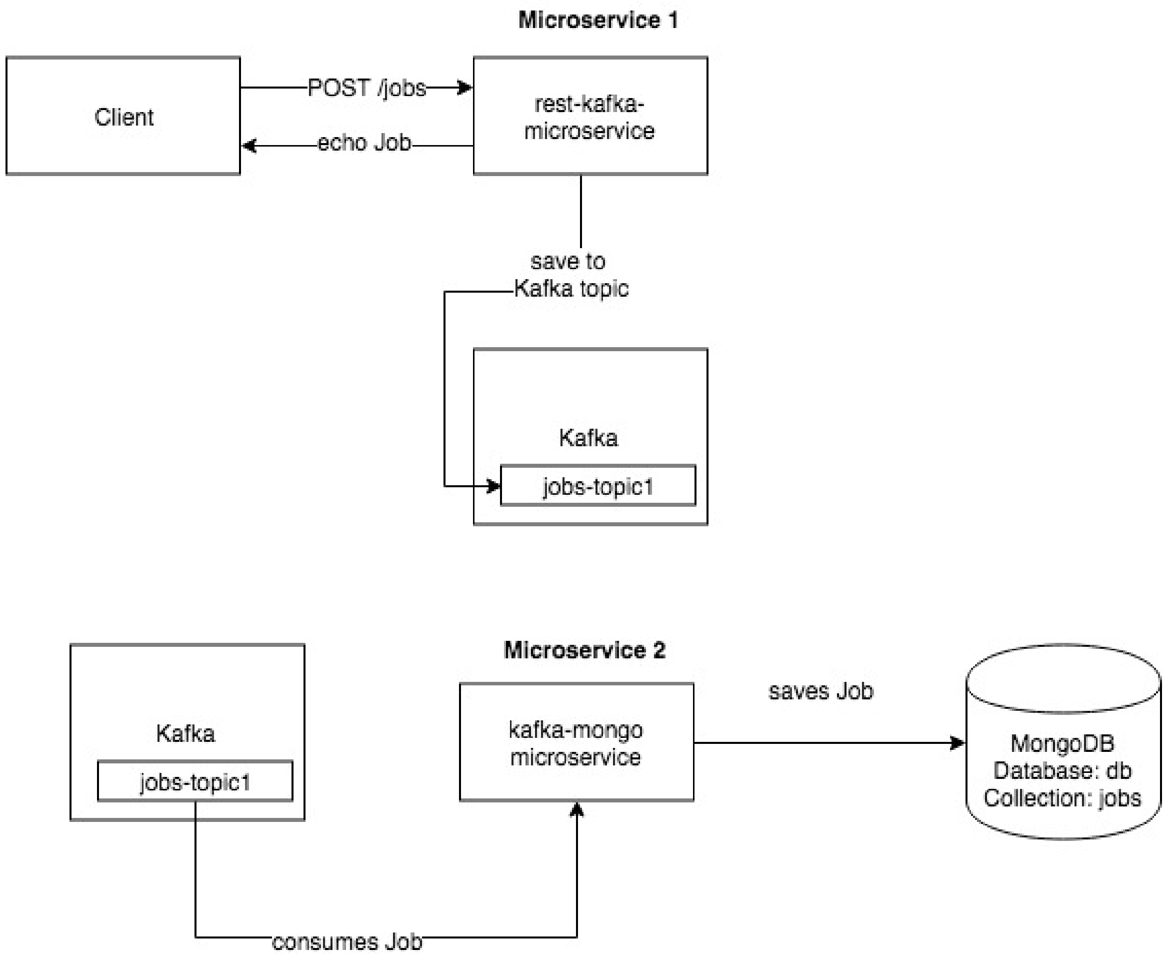
Asynchronous Processing with Go using Kafka and MongoDB

In my previous blog post "My First Go Microservice using MongoDB and Docker Multi-Stage Builds", I created a Go microservice sample which exposes a REST http endpoint and saves the data received from an HTTP POST to a MongoDB database.

In this example, I decoupled the saving of data to MongoDB and created another microservice to handle this. I also added Kafka to serve as the messaging layer so the microservices can work on its own concerns asynchronously.

> *In case you have time to watch, I recorded a walkthrough of this blog post in the video below :)*

Here is the high-level architecture of this simple asynchronous processing example wtih 2 microservices.



Microservice 1 - is a REST microservice which receives data from a /POST http call to it. After receiving the request, it retrieves the data from the http request and saves it to Kafka. After saving, it responds to the caller with the same data sent via /POST

Microservice 2 - is a microservice which subscribes to a topic in Kafka where Microservice 1 saves the data. Once a message is consumed by the microservice, it then saves the data to MongoDB.

Before you proceed, we need a few things to be able to run these microservices:

1. Download Kafka - I used version kafka_2.11-1.1.0
2. Install librdkafka - Unfortunately, this library should be present in the target system

3. Install the Kafka Go Client by Confluent
4. Run MongoDB. You can check my previous blog post about this where I used a MongoDB docker image.

Let's get rolling!

Start Kafka first, you need Zookeeper running before you run the Kafka server. Here's how

```
$ cd /<download path>/kafka_2.11-1.1.0
$ bin/zookeeper-server-start.sh config/zookeeper.properties
```

Then run Kafka - I am using port 9092 to connect to Kafka. If you need to change the port, just configure it in config/server.properties. If you are just a beginner like me, I suggest to just use default ports for now.

```
$ bin/kafka-server-start.sh config/server.properties
```

After running Kafka, we need MongoDB. To make it simple, just use this docker-compose.yml.

```
version: '3'
services:
  mongodb:
    image: mongo
    ports:
      - "27017:27017"
    volumes:
      - "mongodata:/data/db"
    networks:
      - network1
```

```
volumes:
   mongodata:

networks:
   network1:
```

Run the MongoDB docker container using Docker Compose

```
docker-compose up
```

Here is the relevant code of **Microservice 1**. I just modified my previous example to save to Kafka rather than MongoDB.

rest-to-kafka/rest-kafka-sample.go

```
func jobsPostHandler(w http.ResponseWriter, r *http.Request) {

        //Retrieve body from http request
        b, err := ioutil.ReadAll(r.Body)
        defer r.Body.Close()
        if err != nil {
                panic(err)
        }

        //Save data into Job struct
        var _job Job
        err = json.Unmarshal(b, &_job)
        if err != nil {
                http.Error(w, err.Error(), 500)
                return
        }

        saveJobToKafka(_job)

        //Convert job struct into json
        jsonString, err := json.Marshal(_job)
        if err != nil {
                http.Error(w, err.Error(), 500)
                return
        }

        //Set content-type http header
        w.Header().Set("content-type", "application/json")

        //Send back data as response
        w.Write(jsonString)

}

func saveJobToKafka(job Job) {

        fmt.Println("save to kafka")

        jsonString, err := json.Marshal(job)

        jobString := string(jsonString)
        fmt.Print(jobString)

        p, err := kafka.NewProducer(&kafka.ConfigMap{"bootstrap.servers"
        if err != nil {
                panic(err)
        }

        // Produce messages to topic (asynchronously)
        topic := "jobs-topic1"
        for _, word := range []string{string(jobString)} {
                p.Produce(&kafka.Message{
                        TopicPartition: kafka.TopicPartition{Topic: &top
                        Value:          []byte(word),
                }, nil)
        }
}
```

Here is the code of **Microservice 2**. What is important in this code is the consumption from Kafka, the saving part I already discussed in my previous blog post. Here are the important parts of the code which consumes the data from Kafka.

kafka-to-mongo/kafka-mongo-sample.go

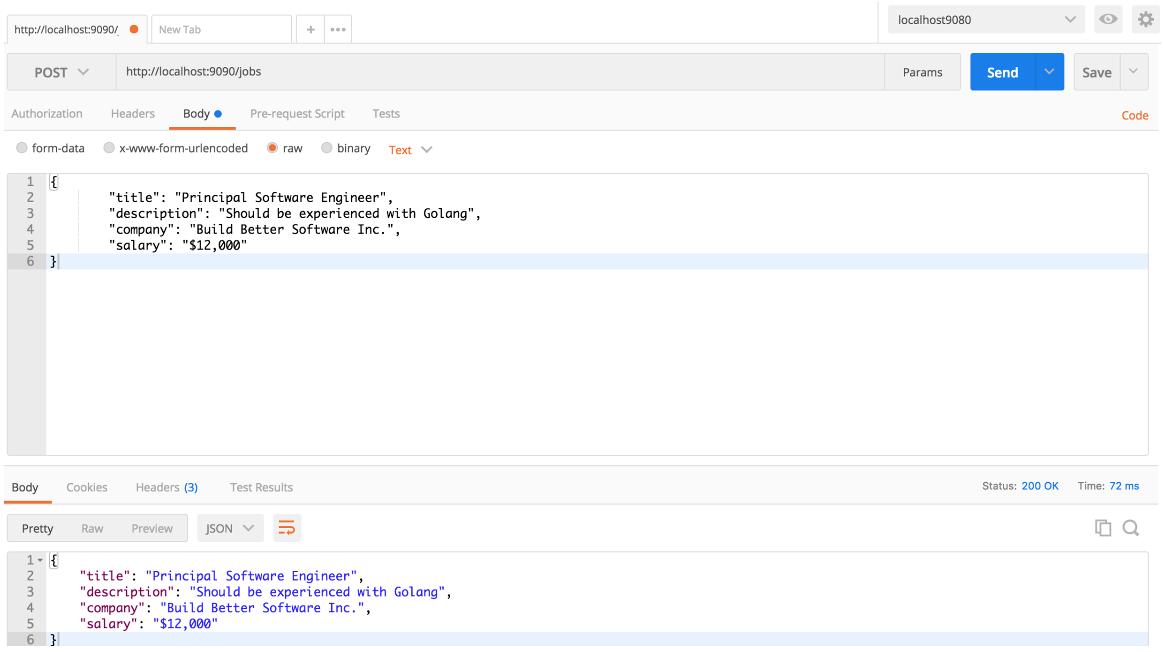
```
func main() {

        //Create MongoDB session
        session := initialiseMongo()
        mongoStore.session = session

        receiveFromKafka()

}

func receiveFromKafka() {

        fmt.Println("Start receiving from Kafka")
        c, err := kafka.NewConsumer(&kafka.ConfigMap{
                "bootstrap.servers": "localhost:9092",
                "group.id":          "group-id-1",
                "auto.offset.reset": "earliest",
        })

        if err != nil {
                panic(err)
        }

        c.SubscribeTopics([]string{"jobs-topic1"}, nil)

        for {
                msg, err := c.ReadMessage(-1)

                if err == nil {
                        fmt.Printf("Received from Kafka %s: %s\n", msg.T
                        job := string(msg.Value)
                        saveJobToMongo(job)
                } else {
                        fmt.Printf("Consumer error: %v (%v)\n", err, msg
                        break
                }
        }

        c.Close()

}
```

```
func saveJobToMongo(jobString string) {

        fmt.Println("Save to MongoDB")
        col := mongoStore.session.DB(database).C(collection)

        //Save data into Job struct
        var _job Job
        b := []byte(jobString)
        err := json.Unmarshal(b, &_job)
        if err != nil {
                panic(err)
        }

        //Insert job into MongoDB
        errMongo := col.Insert(_job)
        if errMongo != nil {
                panic(errMongo)
        }

        fmt.Printf("Saved to MongoDB : %s", jobString)

}
```

Let's get down to the demo, run Microservice 1. Make sure Kafka is running.

```
$ go run rest-kafka-sample.go
```

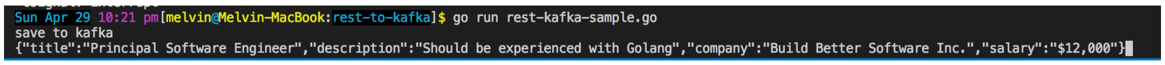
I used Postman to send data to Microservice 1



```
{
        "title": "Principal Software Engineer",
        "description": "Should be experienced with Golang",
        "company": "Build Better Software Inc.",
        "salary": "$12,000"
}
```

```
{
    "title": "Principal Software Engineer",
    "description": "Should be experienced with Golang",
    "company": "Build Better Software Inc.",
    "salary": "$12,000"
}
```

Here is the log you will see in Microservice 1. Once you see this, it means data has been received from Postman and saved to Kafka

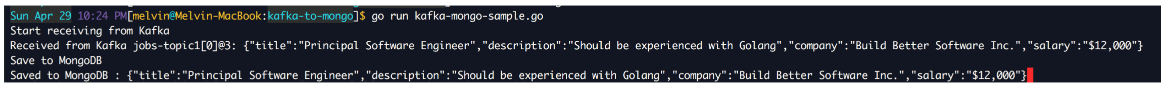


```
Sun Apr 29 10:21 pm[melvin@Melvin-MacBook:rest-to-kafka]$ go run rest-kafka-sample.go
save to kafka
{"title":"Principal Software Engineer","description":"Should be experienced with Golang","company":"Build Better Software Inc.","salary":"$12,000"}
```

Since we are not running Microservice 2 yet, the data saved by Microservice 1 will just be in Kafka. Let's consume it and save to MongoDB by running Microservice 2.

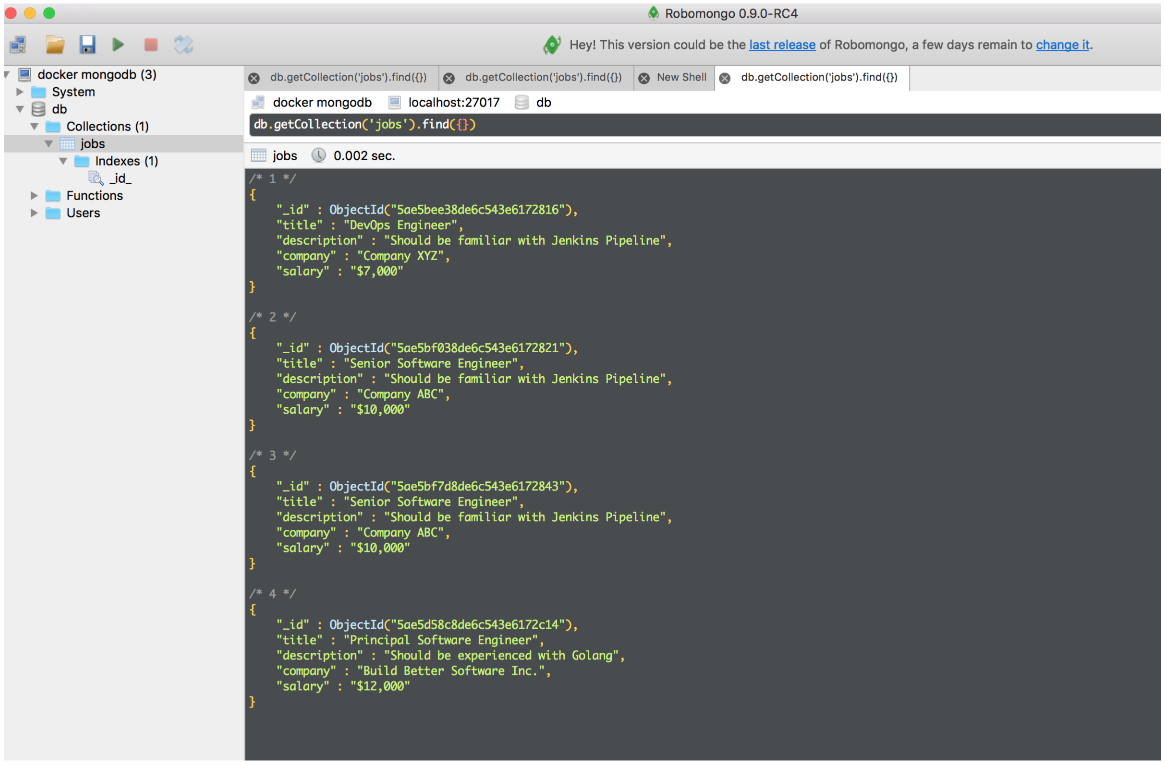
```
$ go run kafka-mongo-sample.go
```

Now you'll see that Microservice 2 consumes the data and saves it to MongoDB



```
Sun Apr 29 10:24 PM[melvin@Melvin-MacBook:kafka-to-mongo]$ go run kafka-mongo-sample.go
Start receiving from Kafka
Received from Kafka jobs-topic1[0]@3: {"title":"Principal Software Engineer","description":"Should be experienced with Golang","company":"Build Better Software Inc.","salary":"$12,000"}
Save to MongoDB
Saved to MongoDB : {"title":"Principal Software Engineer","description":"Should be experienced with Golang","company":"Build Better Software Inc.","salary":"$12,000"}
```

Check if data is saved in MongoDB. If it is there, we're good!



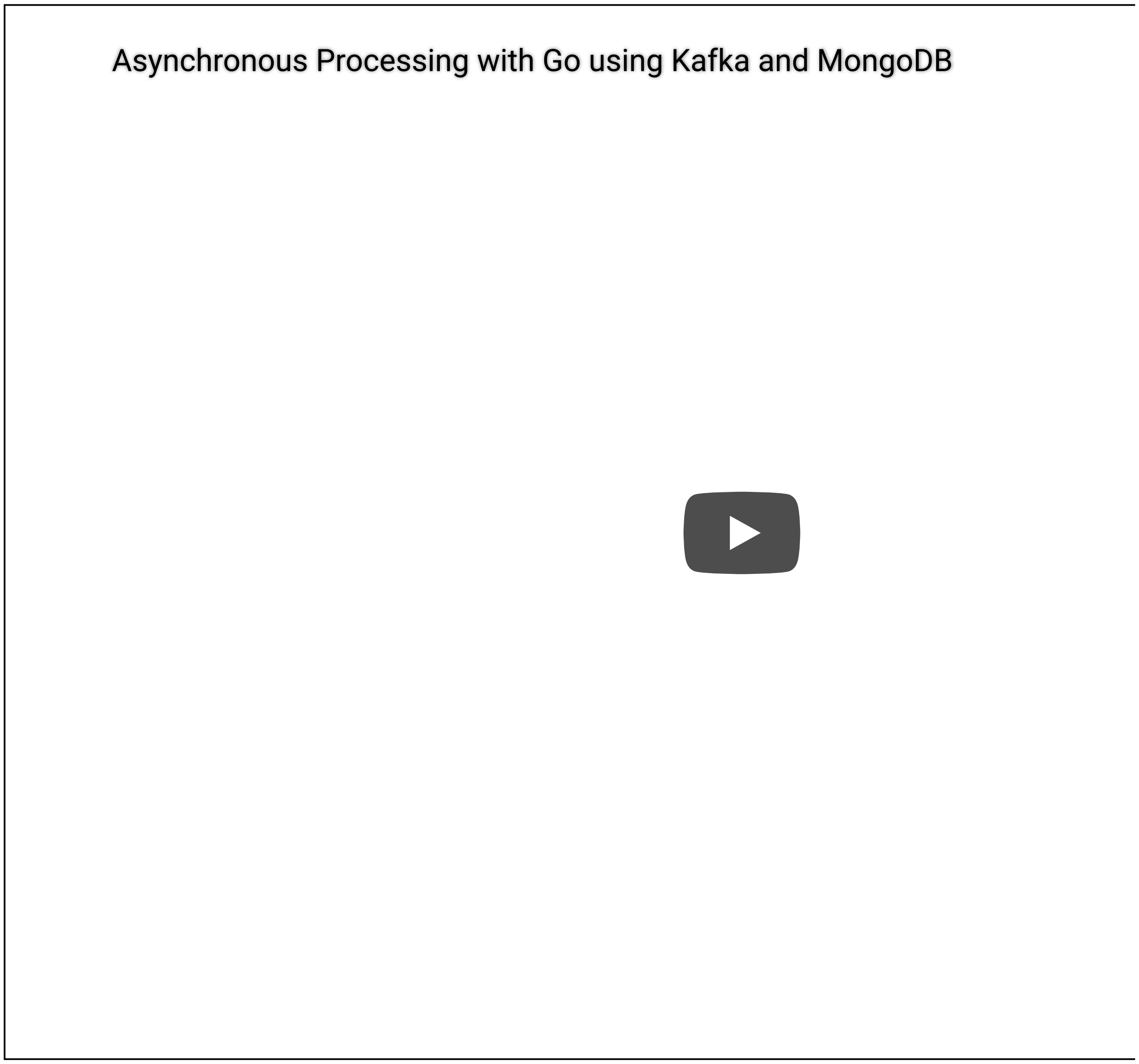
```
db.getCollection('jobs').find({})
```

```
/* 1 */
{
    "_id" : ObjectId("5ae5bee38de6c543e6172816"),
    "title" : "DevOps Engineer",
    "description" : "Should be familiar with Jenkins Pipeline",
    "company" : "Company XYZ",
    "salary" : "$7,000"
}

/* 2 */
{
    "_id" : ObjectId("5ae5bf038de6c543e6172821"),
    "title" : "Senior Software Engineer",
    "description" : "Should be familiar with Jenkins Pipeline",
    "company" : "Company ABC",
    "salary" : "$10,000"
}

/* 3 */
{
    "_id" : ObjectId("5ae5bf7d8de6c543e6172843"),
    "title" : "Senior Software Engineer",
    "description" : "Should be familiar with Jenkins Pipeline",
    "company" : "Company ABC",
    "salary" : "$10,000"
}

/* 4 */
{
    "_id" : ObjectId("5ae5d58c8de6c543e6172c14"),
    "title" : "Principal Software Engineer",
    "description" : "Should be experienced with Golang",
    "company" : "Build Better Software Inc.",
    "salary" : "$12,000"
}
```

Complete source code can be found here https://github.com/donvito/learngo/tree/master/rest-kafka-mongo-microservice

Shameless plug! If you like this blog post, please follow me in Twitter **@donvito**. I tweet about Docker, Kubernetes, GoLang, Cloud, DevOps, Agile and Startups. Would love to connect in **GitHub** and **LinkedIn**

[VIDEO]



Enjoy!

Melvin Dave Vivas
Read more posts by this author.

Read More

3 Comments - *powered by utteranc.es*

broklyngagah commented on Jun 20, 2018

Thanks.. Really nice & helpful tutorial.

donvito commented on Sep 29, 2018 Owner

Thanks @broklyngagah! Glad it helped!

andresr19 commented on Dec 13, 2018

hello im new to microservices in general and i have a question. What if the saveJobToKafka function fails?
The 200 response is still returning to the user?

Write Preview

Sign in to comment

Styling with Markdown is supported Sign in to comment

The GopherCon Singapore 2018 Experience

A few months back, I started my journey to learning Golang. I started with Tour of Go, watched videos of Todd McLeod in Youtube, and tried out examples I find in the web.

MELVIN DAVE VIVAS

Converting a MongoDB Docker Compose file to a Kubernetes Deployment

It's pretty straighforward to convert a Docker compose file to a Kubernetes deployment. I used the kompose tool to do the conversion. Kompose is an official Kubernetes project. https://github.com/kubernetes/kompose

MELVIN DAVE VIVAS