Team Paris

Pub-Sub System Using Kafka

Technologies

APACHE KAFKA

We are using a kafka go library to access all the functionalities of Apache Kafka in a go service.

GOLANG

We are using Gin-Gonic framework to establish a route to the publisher API and call the postDataToKafka function

DOCKER

We are using the docker image given in the PS, with three zookeepers and three brokers in the cluster.

POSTMAN

We are using postman to hit the publisher API through a POST request and the input is given in a JSON format.

Packages

Task	Link
Kafka	confluent-kafka-go
Configuration	Viper
Logging	Logrus, lumberjack
Mocks for testing	Go mock

Features

- Topics
  - Email
  - Phone
- Producer Service
- Consumer Services
  - Email Group (Email client)
  - Phone Group (Phone client)
- Fault Tolerance
- Logging
- Configuration
- Unit Testing (with mocks wherever needed)
  Coverage > 80%(Avg.)

1

Producer

Let's start with producer

Producer

Factory

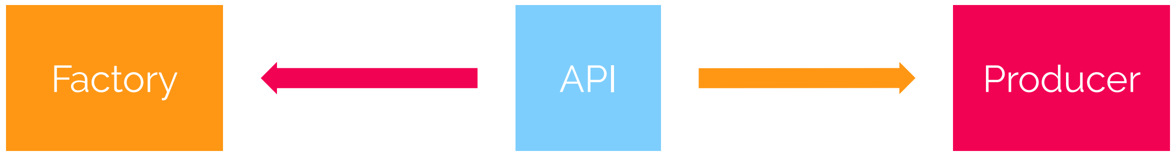
Basically a factory pattern which handles the configuration of a message being published to the Kafka Broker.

API/main

Here is the entry point for the producer API and in this we listen to an address associated with the API call and when this API is hit, we execute a handler function where we publish messages to Kafka Broker.

Producer

We handle everything regarding the producer here. There is a global producer object created once whenever there is a service bootup and the producer is initialized, we create a new producer instance and assign it to the global object.



2

Consumer

Let’s move to consumer

Consumer

Consumer/main

Here is the entry point for the consumer where we initiate a consumer object from worker package. Also we run the consume function as a goroutine here.

Worker

We handle everything regarding the consumer here. There is a global consumer object similar to that of producer and also it contains the consumer function which handles retrieving messages from kafka Broker.

Client

Every functionality regarding the client is here. Here we are mocking the Email and SMS clients. We are also mimicking a server on the client side for check of fault tolerance logic.

Consumer
Worker
Client

3

Configuration & Logging

Let’s dive into Configuration & Logging

Details

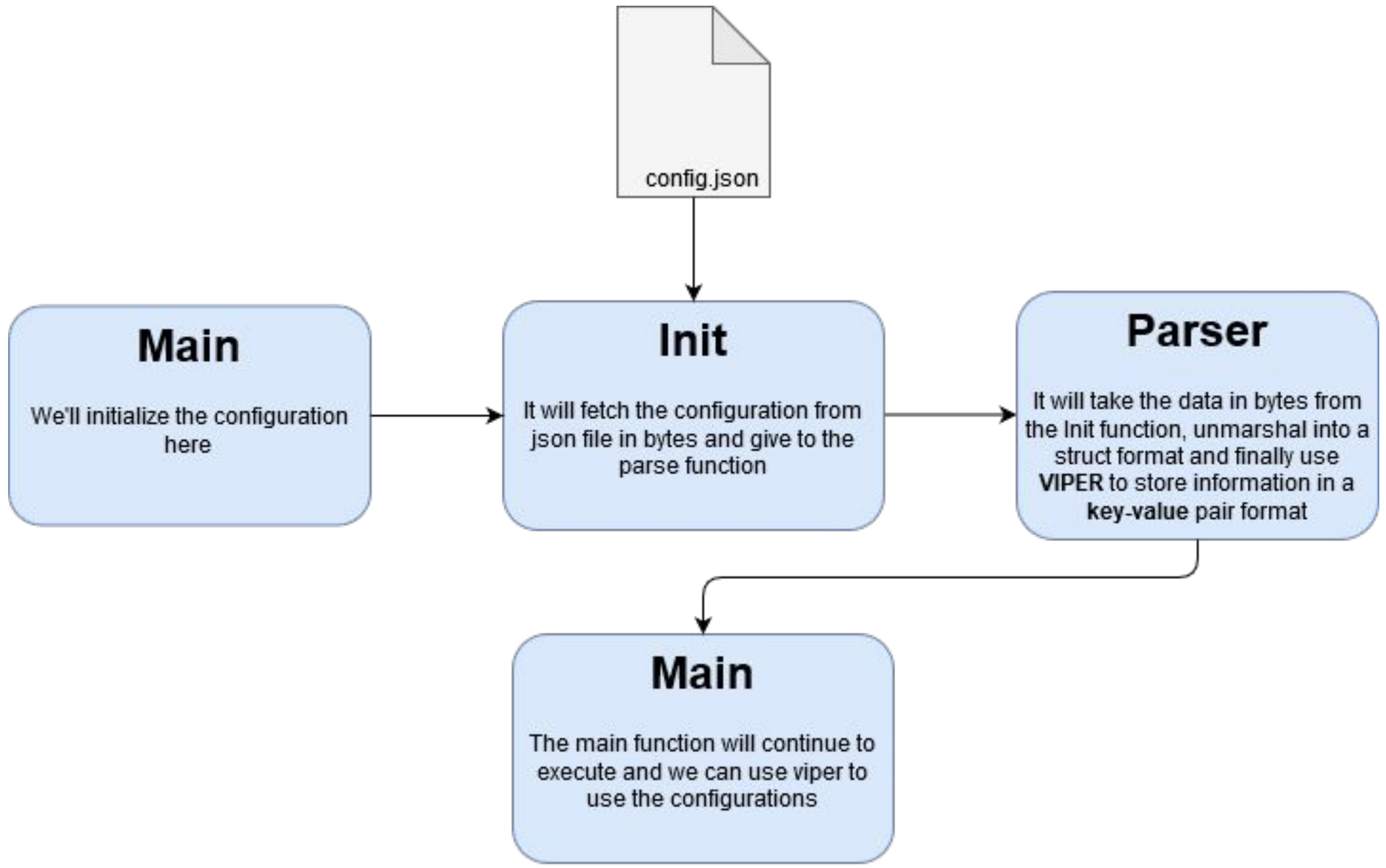
Configuration

For using variables in the code, we are using configuration along with viper to access all those configurable variables.

Logging

For logging, we are having logger package defined and using the logger object throughout the code. We have log rotation also in place.

Configuration


config.json
Main
We'll initialize the configuration here
Init
It will fetch the configuration from json file in bytes and give to the parse function
Parser
It will take the data in bytes from the Init function, unmarshal into a struct format and finally use **VIPER** to store information in a **key-value** pair format
Main
The main function will continue to execute and we can use viper to use the configurations

4

Unit Testing

Let’s end with testing

Testing

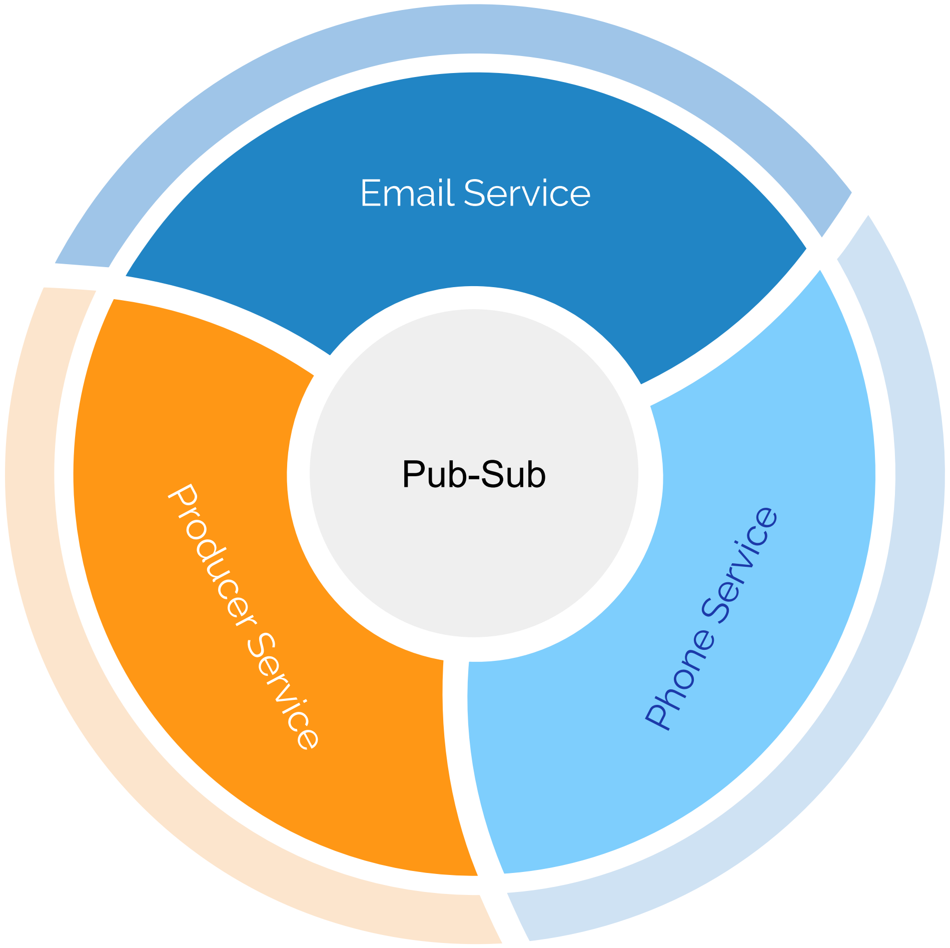
Unit testing

We wrote some unit test cases for most of the units in our packages and tried to achieve a test coverage of more than 80% in many packages.

Mocking

For packages which are dependent on other packages, we used mocks generated by mockgen tool from Go-mock package.

Future Aspect - Refactoring Our Code



Deployed!!

- GCP

Ask IP?

On a Virtual Machine and the External IP of the machine is used for calling the API from Postman

Google Cloud Platform

Thanks!

Any questions?

- Team Paris