

# KUBERNETES

Micro Services

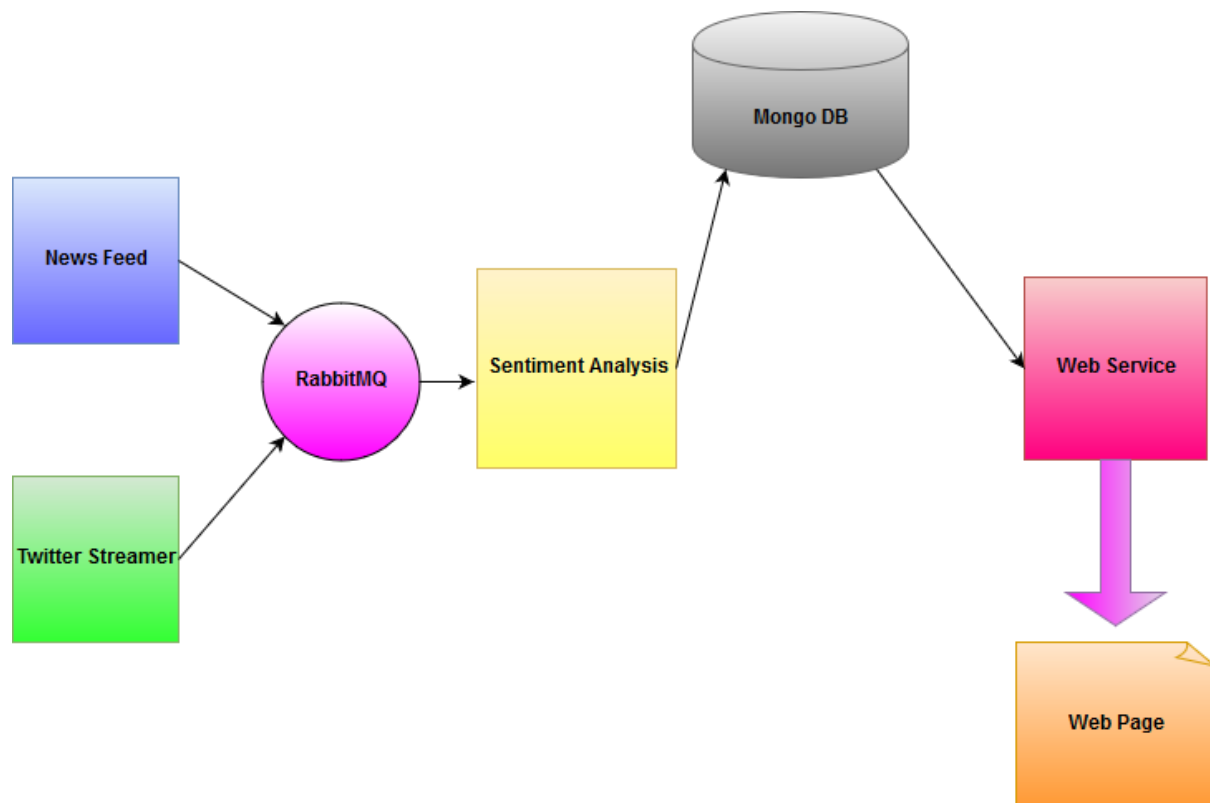


David Coughlan  
R00009964

## Contents

Architecture .....	2
Kubernetes .....	3
Testing.....	4
Automation expansion.....	6

## Architecture



### **News Feed:**

The news feed has two news sources. It has Reddit API set to the sub reddit of Politics. It also uses Python's Feed Parser to get the RSS feeds of three URL's (Yahoo News, Google News, Atom News). It adds the word news to the string just before it sends the message through RabbitMQ. The news sources are static.

### **Twitter Streamer:**

The Twitter Streamer uses a Python library called Tweepy to send tweets about Donald Trump to the RabbitMQ. The word Tweet is added to the front of each tweet before being sent. This produces a constant stream.

### **RabbitMQ:**

The RabbitMQ queue uses the same queue name for both the Twitter and the News. This allows the services to be interchangeable while keeping the Sentiment service pure. This means that when the extra services are added it will form a queue of messages allowing for processing to take place one by one allowing the Sentiment to easily handle the increased feeds.

### **Sentiment Analysis:**

The sentiment receives the news and tweets from the sources through the queue. It checks if the message has the news or tweet string appended to the start of the string. If it does it removes it before doing a sentiment analysis. It scores the sentiment as negative, positive or neutral and then uses this as a "topic" when storing it (either news or tweet). It stores the date, text of the message, its sentiment score and its topic to the MongoDB database.

## MongoDB:

This stores the data that is feed in from the Sentiment service.

## Web Service:

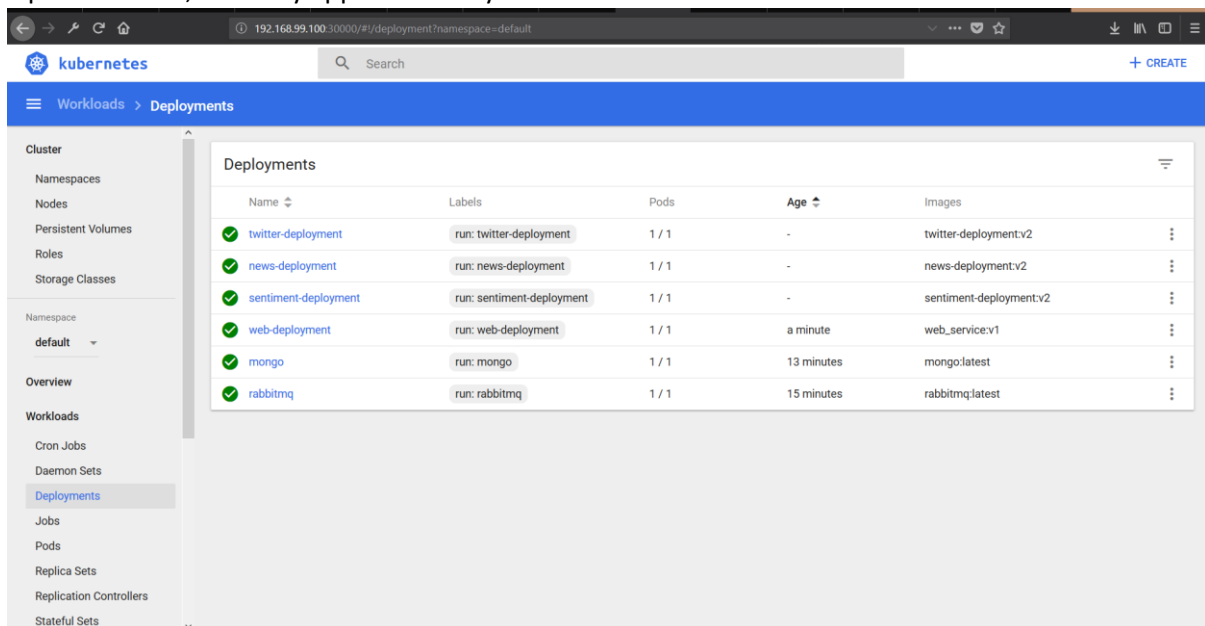
This retrieves data from the database every minute. It does a query for all entries in that time frame and returns a list. The list is then filtered based on the topic (news/tweet). The averages are calculated for both and then sent to a web page using Python's Flask. When the web page is refreshed the new data is displayed for the latest query.

## Kubernetes

I chose to use the MiniKube and work locally. I followed a tutorial to install it and worked with MiniKube and Kubectl in PowerShell. I used the Kompose executable to change my docker-compose file to the yamls for my services and deployments.

I then built each image and tagged them. I then opened the port for the web service.

I pushed them, and they appeared in my MiniKubes Dashboard.



The screenshot shows the Kubernetes Dashboard interface. The left sidebar contains navigation links for Cluster, Namespaces, Nodes, Persistent Volumes, Roles, Storage Classes, Namespace (set to default), Overview, Workloads, Cron Jobs, Daemon Sets, Deployments (selected), Jobs, Pods, Replica Sets, Replication Controllers, and Stateful Sets. The main panel displays a table of Deployments.

Name	Labels	Pods	Age	Images
twitter-deployment	run: twitter-deployment	1 / 1	-	twitter-deployment:v2
news-deployment	run: news-deployment	1 / 1	-	news-deployment:v2
sentiment-deployment	run: sentiment-deployment	1 / 1	-	sentiment-deployment:v2
web-deployment	run: web-deployment	1 / 1	a minute	web_service:v1
mongo	run: mongo	1 / 1	13 minutes	mongo:latest
rabbitmq	run: rabbitmq	1 / 1	15 minutes	rabbitmq:latest

Then the app was accessible on the IP address



The screenshot shows a web browser displaying sentiment analysis results. The address bar shows the IP address 192.168.99.100:32228. The page content shows the following data:

Positive Tweets: 37.10691823899371
Negative Tweets: 25.78616352201258
Neutral Tweets: 37.106918238993714
no news

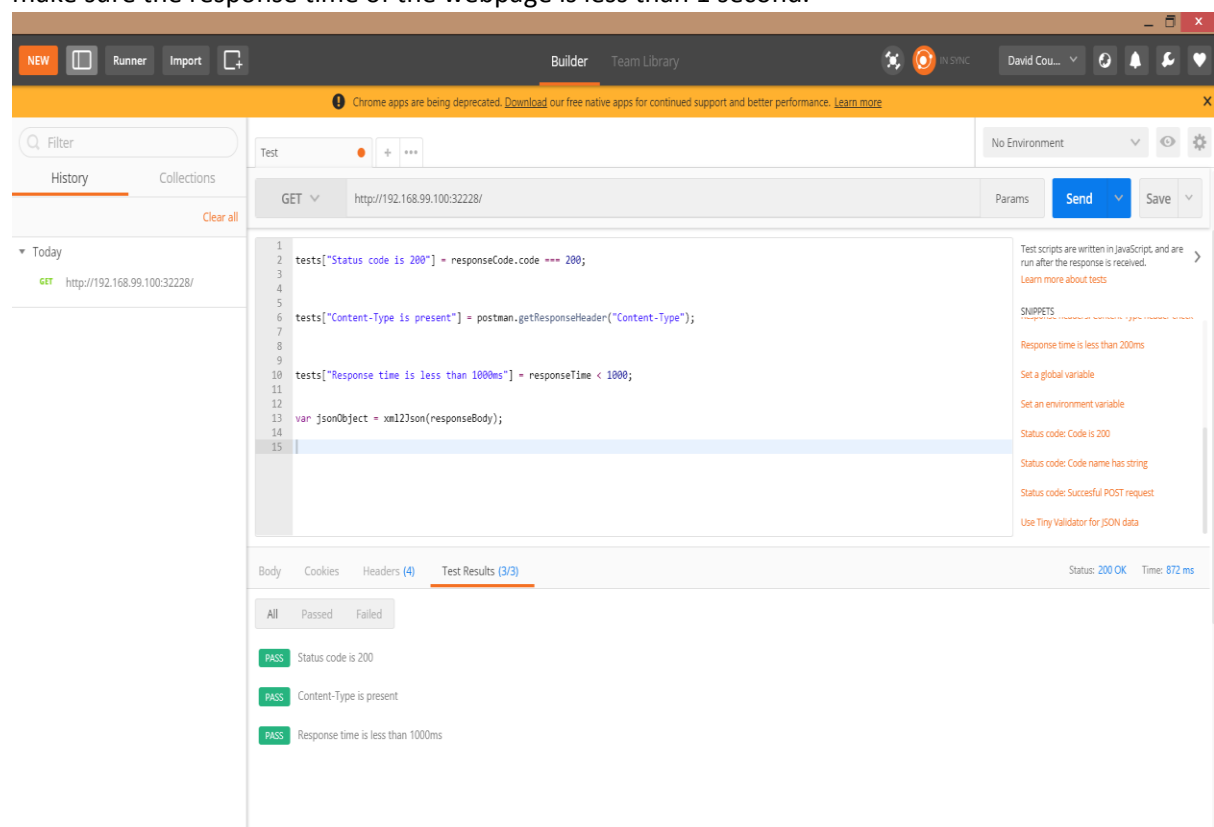
## Testing

I used Gatling for my non-functional test. (<https://gatling.io/>). It is used to help anticipate slow response times and crashes. Gatling can be added to a project to inspect load and performance testing.

I have attached a video of the process as well as the output files of the tests. There is a webpage in the results folder that displays graphs of the outcomes. It captures the traffic of the application and analyses it.

I then used Postman to test the webpage. The tests included checking the response code to make sure it isn't an error and can connect (not 404).

A test to check the content type in the header to make sure it's being displayed. A response test to make sure the response time of the webpage is less than 1 second.



I then tried to do Postman Monitoring however this was not successful as the webpage did not have a JSON collection. I have included a video of this as well.

I then moved to Prometheus as told in the IBM talk. Prometheus uses Helm to be installed.

Prometheus

Details

**Name:** prometheus-operator

**Namespace:** default

**Labels:** app: prometheus-operator chart: prometheus-operator-0.0.25 heritage: Tiller operator: prometheus release: prometheus-operator

**Annotations:** deployment.kubernetes.io/revision: 1

**Creation Time:** 2018-05-19T14:30 UTC

**Selector:** app: prometheus-operator operator: prometheus release: prometheus-operator

**Strategy:** RollingUpdate

**Min ready seconds:** 0

**Revision history limit:** 2

**Rolling update strategy:** Max surge: 25%, Max unavailable: 25%

**Status:** 1 updated, 1 total, 1 available, 0 unavailable

New Replica Set

Name	Labels	Pods	Age	Images
prometheus-operator-6d94d55b6d	app: prometheus-operator operator: prometheus pod-template-hash: 2850811628 release: prometheus-operator	1 / 1	10 minutes	quay.io/coreos/prometheus-operator: <div> </div>

This added Prometheus to my minikube dashboard as a pod.

Pods

Name	Node	Status	Restarts	Age
prometheus-operator-6d94d55b6d-qzcm7	minikube	Running	0	4 hours

Deployments

Name	Labels	Pods	Age	Images
prometheus-operator	app: prometheus-operator chart: prometheus-operator-0.0.25 heritage: Tiller operator: prometheus release: prometheus-operator	1 / 1	4 hours	quay.io/coreos/prometheus-operator: <div> </div>

I used Prometheus to monitor the CPU, memory, disk and network metrics of the pods using the Prometheus node exporter. Kubernetes also has this with cAdvisor.



Using kube-state-exporter I could see an overview of the clusters state. This tells me of the state of the running pods and scheduling queues.

## Automation expansion

All the testing can be incorporated into CLI tools such as Jenkins. A pipeline is a suite of plugins which supports implementing and integrating continuous delivery pipelines into Jenkins.

A continuous delivery (CD) pipeline is an automated expression of your progress for getting software from version control right through to the user. Every change of the software (committed to source control) goes through the complex process of development and changes as the software evolves and increments. This process allows the software to be developed and built in a responsible manner as well as progressing the build through multiple stages of testing.

A Jenkins file can be made and committed to the source control. This will automatically create a Pipeline build process for all branches and pull requests.

### *Jenkinsfile (Declarative Pipeline)*

```

pipeline { ❶
  agent any ❷
  stages {
    stage('Build') { ❸
      steps { ❹
        sh 'make' ❺
      }
    }
    stage('Test'){
      steps {
        sh 'make check'
        junit 'reports/**/*.xml' ❻
      }
    }
    stage('Deploy') {
      steps {
        sh 'make publish'
      }
    }
  }
}

```

Here the Test stage can clarify the process of testing to take place for each build/ push alerting the developers to any changes that cause an issue.