Microservice API's in container Architecture for Autonomous Sensor Network

Kalpana A, Raj Kumar V, Rajashekhar N H
Central Research Laboratory-BEL, Bangalore, India
{kalpanaa,rajkumarv,rajashekharnh}@bel.co.in

DOI: 10.29322/IJSRP.10.02.2020.p9875
http://dx.doi.org/10.29322/IJSRP.10.02.2020.p9875

Abstract

Microservice based Autonomous surveillance system connects data sources with data sinks using interfaces and message queues for more reliable and scalable performance. In autonomous surveillance systems, data acquisition, message queue, data persist, data analysis and visualize models are running as a microservices in container. Communication between Microservices is very efficient and robust since it is handled by docker-compose file. The proposed paper describes the process of decision making at real-time by integrating of multiple sensors, big data in various format, and disseminate the data from sensors to a visual mode through different level data preprocessing techniques. The preprocessing models from sensors to edge node is running as microservices in containers. The containers named as data acquisition, Data processing, data persistence, Data analyzing and Visualization services. Those processes accomplished by Kafka, mongoDB, TensorFlow, ELK stack deployed in containers.

Keywords: Microservices, Docker, Containers, ELK, Object detection, Kafka.

1. INTRODUCTION

A. Autonomous Surveillance system

Autonomous Surveillance System, capable of capturing and monitoring the sensor data from distributed ground sensors and transfer the data to the cloud. Further, the data is analyzed using artificial intelligence to perform quick actions based on user input. The proposed system supports planning and decision making in real-time on cloud infrastructure with efficient usage of resources and services. Sensors should be monitored, data should be captured and captured data should be consolidated, shared, analyzed and computed for decision making and visualizing on wired/wireless networks.

The two major functionalities of this system include Observation and Service. The observation and service parts are handled by the cloud platform from capturing the data from sensors through wired/wireless interfaces. Also, data is modeled and transferred to end-users through wired/wireless interfaces. If microservices run directly on cloud, it will be deployed on virtual machines and it has serious limitations. Virtual machines imposes a large performance loss and copy of the operating system and its must run its own execution environment. In contrast, containers are the best solution and it performs isolated execution at the operating system level. Here, multiple containers supported by a single operating system, each running within its own, and separate execution environment. It will free up processing, reduce overhead and power for application components.

In Monolithic architecture, different modules of the system are tightly coupled and even if a single service failure will affect the entire system. The proposed system is developed using microservices where all modules are decoupled and independent of each other. So, the failure of a single service will not bring down the entire system. These services can be developed using different programming languages and use different independent data storage techniques. A Container is the best infrastructure for Microservices and it is the new trend to packaging libraries and dependencies and deploying Microservices based application.

Containers provide execution isolation at the operating system level. A single operating system instance supports multiple containers, each running in its defined environments and containers are very lightweight compared to VMs. The benefit of microservices in containers for the autonomous system is elasticity, storage, security, and availability.

2. BACKGROUND

A. Microservices deployed on Container

Microservice could communicate using HTTP protocols. Microservices expose API's to interact with other Microservices and client applications. REST APIs are helping to make fast and network latency and less overhead communication between clients and microservices. Each microservice can run in single or multiple containers, and these services can be distributed among different geographical locations. All the microservices are abstracted to the client applications behind an API gateway. Client access these services through the API Gateway which combines all the responses from different microservices.

B. Docker Compose

Docker Compose is the toolkit provided by the Docker platform for build, ship and run multi-container applications. If an application is a combination of multiple components i.e., multiple containers for different purposes then Docker compose is the tool that creates the environment to define the containers from the application stack.

Docker Compose is mainly categorized into two key forms:

1) Docker Compose in the form of a command-line command that will be used for build, ship and run multi-container applications. The command is called docker-compose.

2) Docker Compose provides the Compose/Configuration file, where all the components of the application stack and their interaction with each other are defined under the services, networks, and volumes of the configuration file in the yaml (or) yml format.

Docker-compose allows the users to declare the rules within a single docker-compose.yml configuration file. Every compose file format should have at least one service, and optionally volumes and networks.

C. Real Time Object Detection

The Autonomous system, anomaly detection, and object detection are mainly used for surveillance. A camera also being part of the sensors list, objects are detected using the TensorFlow machine learning framework[5]. This framework allows for the recognition, localization, and detection of multiple objects. It provides us with a much better understanding of an image as a whole.

3. AUTONOMOUS SYSTEMS

Automatic systems refer to self-managing, inexpensive, user-friendly systems. The continuous live streaming of sensors data producing a big volume of data to Kafka distributed processing unit under various topic names. Add to this, data persisting on mongo DB parallel. When data arrives at Kafka topics, it forwards the data to Elasticsearch for searching and indexing using Logstash and the data is visualized using the Kibana dashboard. So in the underneath system, data acquisition, analyses, processing and persist functions are automated.

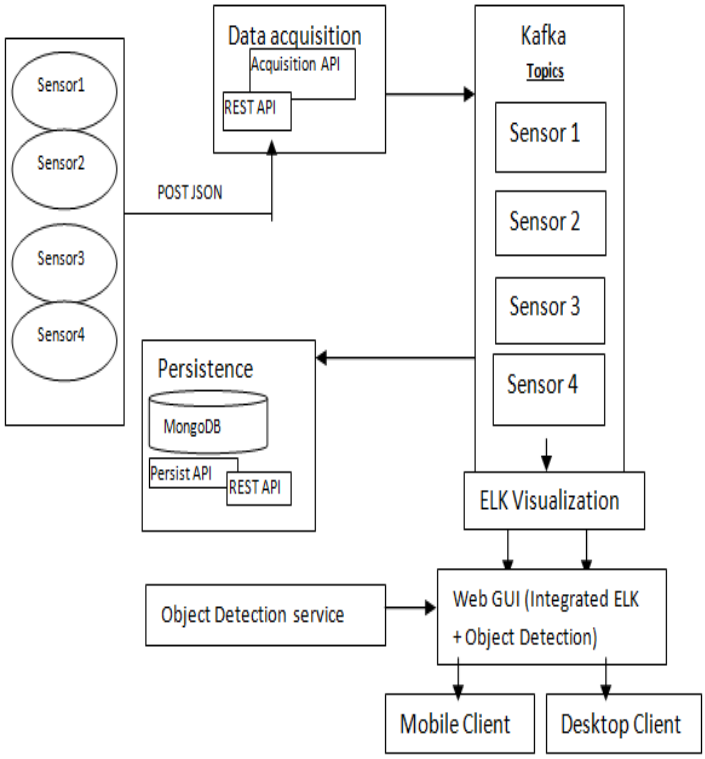

Fig. 1 Architecture of Autonomous System

Fig. 1 shows the flow of the automated system, how the acquisition, processing, data persistence, object detection, analyze and visualize models are integrated with others [1]. This automated system is self -management system, where complex technology and heterogeneous modules are integrated, which collects data from various sensors on various topics and dispatches the data for storing and analyzing. Besides, supports dynamic updating of the MicroServices at runtime. All sensors are data producers; gathered sensed data from the surrounding area, MicroService applications are data consumers and leverage the generated data for meaningful insights.

A. Data Acquisition:

Data acquisition microservice has the capabilities of read data from sensor and data can be any format such as CSV, JSON, and byte array. When sensors start to interact with applications, it makes a POST request to an API endpoint to send their streaming information. This information is transferred to Kafka microservices.

B. Messaging service

Kafka is mainly used for transfer data from one end to another end. In an autonomous system, Kafka acts as a mediator between data acquisition and Logstash containers. Kafka is distributed and highly fault-tolerant and handles high demands of Microservices very efficiently. Data acquisition Microservice will write data from various sensors to particular topics using Kafka messaging queue and Logstash will read data from Kafka topics and forward those data to Elasticsearch for indexing, searching, analyzing and visualizing by Kibana. For processing or visualizing consumers will read data from Kafka brokers [2]. Kafka broker is a per-node agent store message in different topics for later utilization. Messages are managed in Kafka in the partition by offset to identify polling position [3].

TABLE 1
Table of all services and their consumer and producer information

Services	Consume From	Produces To
DataAcquisition Service	Sensors[Temperature, Humidity, etc]	Kafka
Object detection Service	Sensor(Camera)	WebRTC Flask server
Kafka Service	DataAcquisition Service	MongoDB, Logstash
Logstash Service	Kafka	ElasticSearch
ElasticSearch	Logstash	Kibana

C. Database as a Container

The intent of database service is for smooth data flow operations. MongoDB sharding capability helps for the automation process to store data in multiple systems and ease of accessibility when data in demand. Its suit for microservice architecture and eliminates the complexity of communication over microservices API. It adds durability to the application and the scale of distributed approach help to replicate the service if a service goes down.

D. ELK Services as Containers

Three dockerized images Elasticsearch, Logstash, and Kibana running parallel by using docker-compose. Logstash is depending on Elasticsearch and Kafka services so those two services need to be deployed first. Logstash is a transformer, where input Kafka is an input provider and Elasticsearch is output reader. Elasticsearch is exposed and mapped on port number 9201. Kibana is depending on the Elasticsearch, therefore Elasticsearch should be deployed before Kibana. Kibana visualizes the data which is stored on elastic search data and Kibana services expose on port number 5601[4].

4. IMPLEMENTATION

The proposed project configuration (yaml) file consists of 7 containerized services for Kafka (zookeeper and broker), ELK (Elasticsearch, Logstash, Kibana) stack, DataAcquistion and Object Detection models.

Sensors deployed in remote locations gather different types of information from the surroundings like temperature, humidity, and live feed through cameras, etc. All this information collected by a container service named data acquisition. This service depends on the Kafka services namely Zookeeper and Broker.

```
dataacqusition:
        image: dataacquisition:latest
        container_name: dataacquisition
....
depends_on:
        - zookeeper
        - broker
```

Data Acquisition container service runs on 8081 ports and exposes the same 8081 port on to the host machine[6]. This container service uses the host volumes which maps the host machine's directory to the local directory.

```
ports:
  - “8081:8081”
volumes:
  - <hostMachine’s directory:/<container Local directory>
```

Object Detection service running as a container using TensorFlow Object Detection API and Flask web server. Javascript HTTP POST method sending frames to TensorFlow API to detect the object on every frame. This application running as a docker image and it is exposed and mapped on port number 5000.

All the services defined in this configuration file are discoverable by each other with a user-defined network.

```
networks:
 - <user_defined_network>
```

The command attribute usually takes the value of starting the application from the command line in the container.

After the data gets collected in the data acquisition container and then it is pushed to the Kafka containers namely Zookeeper and Broker.

Zookeeper service runs on port 2181 and the same exposure to the host machine and broker runs on 9092 with 9092 exposed to the host machine and depends on the zookeeper service. The environment variables for the broker are defined in the configuration file such as BROKER_ID, ADVERTISED_HOST_NAME, CREATE_TOPICS, REPLICATION_FACTOR, etc.

From the Kafka containers, the data is directed to the Logstash service running on 4000 port. From the Logstash service, data is directed to the Elasticsearch service, where it is indexed and visualized using the Kibana framework and Kibana is running on port number 5601.

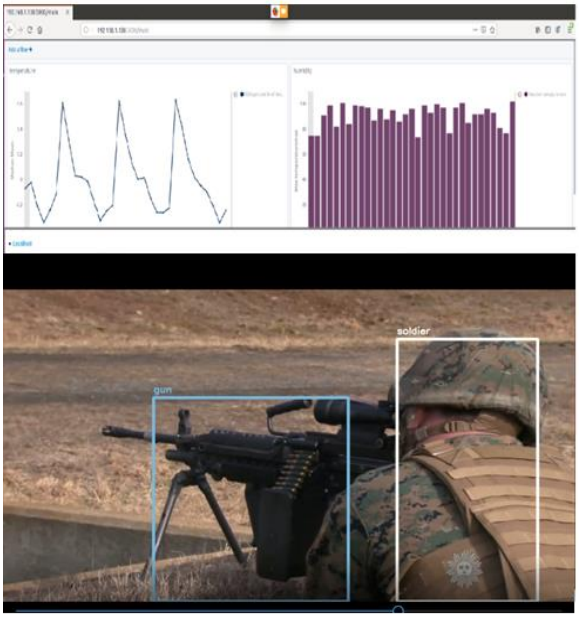

Fig. 2 Screenshot of Autonomous Sensor Network

5. CONCLUSIONS

In this paper, we described the system which is integrated with various heterogeneous sensors build an autonomous system using microservice applications in container architecture. The bid data is involving an end to end data transportation that is from various sources to sink will be automated without any human intervention. This model will deliver the reliable decision making system for surveillance. This system various complex technologies and easily handled and interconnected by docker-compose file. To further extend, the system will be is interrogated with unmanned aerial and unmanned water sensors for surveillance.

REFERENCES

[1] AdalbertoR.SampaioJr.1*,Julia Rubin2, IvanBeschastnikh3 andNelsonS.Rosa1, SampaioJr. etal. "Improving microservice-based applications with runtime placement adaptation", Journal of Internet Services and Applications (2019)
[2] Paul Le Noac'h, Alexandru Costan, Luc Bouge, INSA Rennes, " Performance Evaluation of Apache Kafka in support of Big Streaming Applications", IEEE International Conference on Big data, 2017.
[3] Cao Nguyen, Jik-Soo Kim,Soonwook Hwang, "KOHA: Building a Kafka based Distributed Queue System on the fly in a Hadoop Cluster", National Institute of supercomputing and Networking at KISTI, 2016
[4] SammerDharur, K Swaminathan, "Efficient Surveillance and Monitoring using the ELK stack for IoT Powered Smart Buildings", pp.700-705, Proceedings of Second International Conference on Inventive Systems and Control, 2018.
[5] Chad hart, **"**Computer Vision on the Web with WebRTC and TensorFlow", December 3,2017, webrtcH4cks
[6] For microservice development https://start.spring.io
[7] https://spring.io/blog/2015/07/14/microservices-with-spring