SERVERLESS SERVICES IN KNATIVE

K.Regin Bose[1], Dinesh Karakambaka[2], Belwin J Brearley[3]

[1] Professor, Department of CSE, Chennai Institute of Technology, Chennai, India

[2] Department of CSE, Chennai Institute of Technology, Chennai, India

[3] Assistant Professor, Department of EEE, B.S. Abdur Rahman Crescent Institute of Science and Technology, Vandalur, India

Email: [1]reginbosek@citchennai.net, [2]karakambakadineshcse2019@citchennai.net, [3]belwin@crescent.education

Abstract

Knative is a serverless environment for easy code deployment to kubernetes. It is an open-source to build serverless and even-driven applications. It increases the ease and functionality of running workload in kubernetes. The proposed work deals with the auto scaling benefit of open source serverless platform against proprietary serverless tools based on Kubernetes and Knative. This can reduce the service cost for customers.

Keywords: *Kubernete, Knative, Cloud, Serverless service*

1. Introduction

Knative refers to Kubernetes plus native that provides deployment for containerized apps which is native to the Kubernetes platform. It is used to manage serverless workloads with build, eventing and serving component. The build component says how the code is built and packed in a container. The serving component deploys serverless apps as knative services and it scales up and down. The eventing component is designed to address the needs of cloud-native message. In Knative Kubernetes are used for scaling, load balancing and self healing [1][2]. Istio is used for traffic management, security and observability. The characteristics of knative are serverless computing, on demand execution, per second billing.

Figure 1:Knative Structure

Kubernetes is an open source plat form used by enterprises for the past few years as public cloud, private data center and hybrid cloud environments.

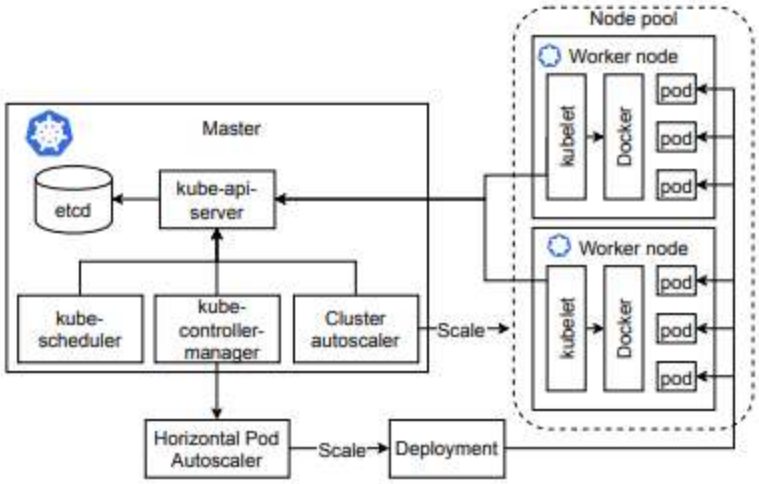

Figure 2: Components in Kubernetes

The major components in Kubernetes are master, worker node, horizontal pod autoscaler. Pod is the smallest element in Kubernetes that has one or more containers for sharing network and storage namespace. For a request by user for pods in Kubernetes, kube- api-server creates pods and kube -scheduler will select appropriate cluster to locate the pods. Kube scheduler in application level is done by two ways namely horizontal autoscaler and vertical pod scaler[3][4][5]. In horizontal pod autoscaling the number of pods is adjusted based on memory and CPU utilisation. In vertical pod autoscaling the number of pods is adjusted based on past resource utilization.

The Kubernetes alone if applied to manage serverless workloads are i) more effort is needed for managing cluster resources ii) possibility to scale down to zero is very less iii) YAML files has to be created which is too complex iv) non availability of common IaC templates

2. Proposed System

The proposed application circuitry is built over the windows operating system. Over the windows operating system the WSL(Windows Subsystem for Linux) is integrated.WSL enables to run a Linux file system over the windows. WSL brought an ability to run Kubernetes on Windows almost seamlessly since Kubernetes has been originally designed to be deployed and used in the Linux environments. Now over the WSL a Kubernetes cluster is created to run application. Kubernetes clusters allow containers to run across multiple machines and environments: cloud-based, virtual, physical and on-premises.

Kubernetes containers are not restricted to a specific operating system, unlike virtual machines. instead, they are able to share operating systems and run anywhere. There are many types of Kubernetes distributions available such as minikube, kind k3s. Among them here k3S distribution is opted.

2.1 Architecture Diagram

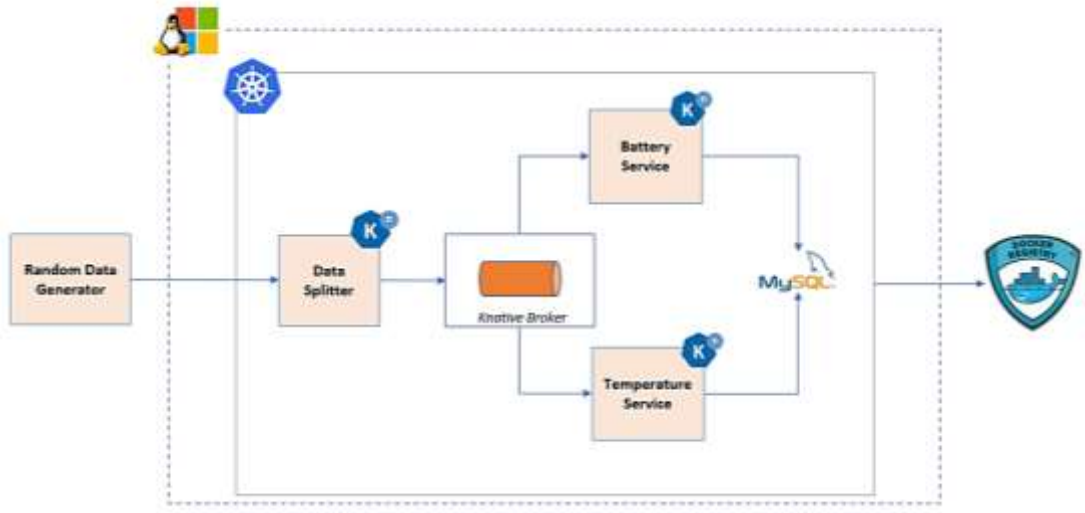

Figure 3. Architecture design of proposed system

2.1.1 Random Data Generator

Random data generator is a device that generates the data with the perception of integrated environment. The data generated includes beacon Id, date, time, battery level and temperature level. After the data is generated, it is send to the data parser.

2.1.2 Data Spliter

Data splitter receives the data from random data generator and then separates the received data into cloud events. Then the cloud events is sent to the broker, which is being integrated over the messaging channel.

2.1.3 Knative Broker

Knative Brokers forms the middle-ware which is embedded over a messaging channel. It supports filtering of events. Event filtering allows the subscribers to show interest in a specific set of messages that flows into the broker. For every broker, a knative eventing channel is created through knative eventing.

2.1.4 Battery Service

In battery service, the separated battery data is received from data parser. The received data is compared with the threshold battery voltage. If it is greater than the threshold voltage then alert message is prompted. Here MySQL database is used for data storage.

2.1.5 Temperature Service

In temperature service the temperature alone is received from data parser. If the data received is greater than the threshold then alert message is prompted.

2.1.6 Triggers

Trigger is defined as the state transition function. The trigger moves from one state to other state whenever the specified condition on the input holds. The event services actively participate in the entire process of cloud control such as event production, event detection, event logging, data analytics, service orchestration and event reaction [6][7].

2.2 Docker Images

The docker files are used to generate docker images for the flask apps. Base image forms the empty first layer which is used to build the docker images from scrape[8]. A variety of stored images can be used as the parent image and several new images can be created from the existing parent image.

2.3 Docker Registry

Docker registry is server side application used to store and distribute docker images. The docker registry runs the port 5000 locally to store docker images that are created. The registry observes port 5000 for the operations such as pulling an image, pushing an image and loading an image from the disc [9][10][11]. Several versions of same images are maintained with different index and seperate tags. There may be several versions of the same image, each with its own set of tags

3. Working principle of Proposed System

The proposed system used four different applications namely Random data generator, data splitter, temperature service and battery service. Random data generator percepts with the environment and generates data. After generating the data , it sends it to the data parser. Data splitters receive the data from beacon simulator and separates it into cloud events and sends it to the broker, which is integrated over the messaging channel. Battery service receives the separated battery data from data parser and sends alert message if it is greater than threshold value. MySQL database is used for data storage .

Temperature service receives temperature data from data parser and generates alert if it exceed the threshold value. For every application an image is built separately and stored in the docker in the local machine.

The Random data generator sends the data to the Data splitter and it splits the battery data and temperature data separately and send to the broker. The broker sends to the Triggers where the cloud event is specified. And if ce-type is battery it is send to the battery service and if ce-type is temperature it is send to the temperature service. The voltage or temperature values which exceeds the threshold value are stored in mysql database.

4. Comparison, advantages and sample output of Proposed System

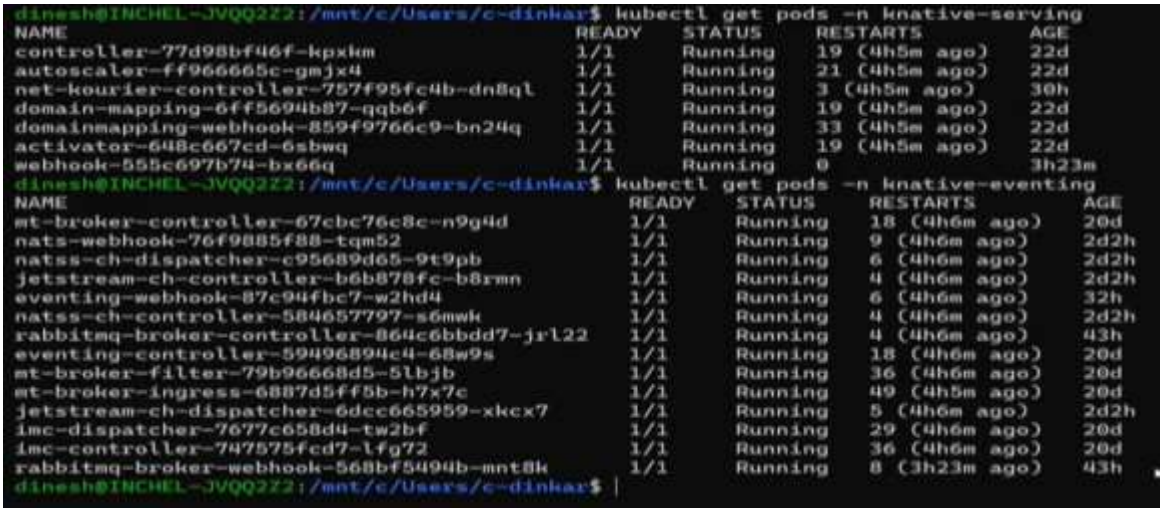

Figure.4 : Installation of knative components

The installation of the knative component knative serving and knative eventing are done successfully and is shown in Figure 4. Then the knative services and broker with the URLs for those services is created to access them using the provided URL which is shown in Figure 5. Whenever request is received by the service URL, the pods for that particular service are scaled to 1. Here it is done for single request here. Hence by default a single pod can handle upto 100 requests and it can be configured in knative which is shown in figure 6. When there is no request, the service pods are completely scaled down to 0 thereby Auto Scaling is done in Knative which is shown in figure7.

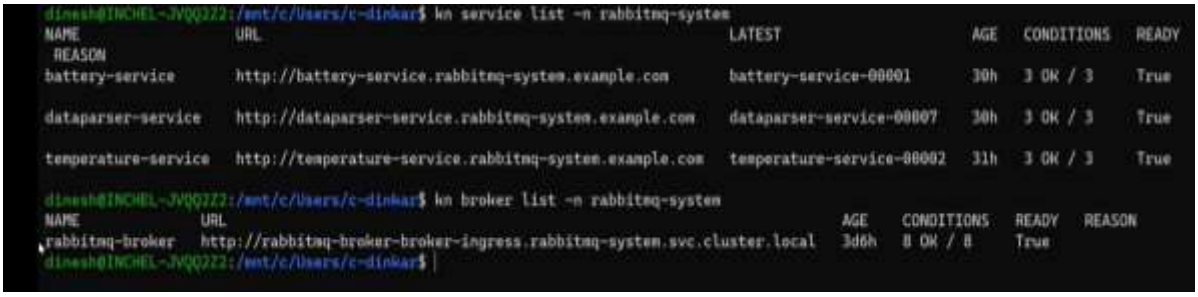

Figure.5: Deployment of knative services and broker with url

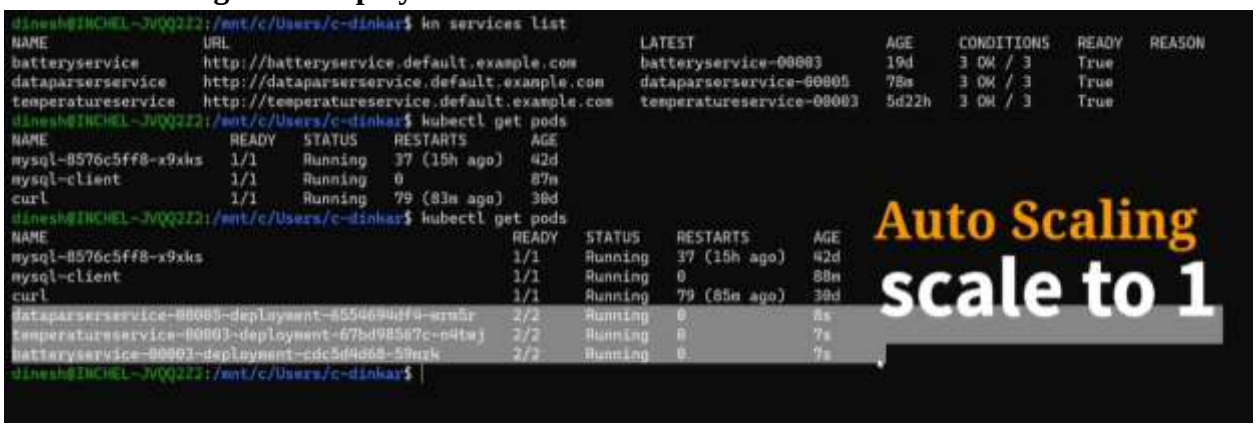

Figure.6 :Proposed system Scale to 1

Figure -7 Proposed system Scale to 0

4.1 Comparison

In Kubernetes without the Knative scaling down to zero is not possible and it is achieved in knative using autoscaling. By using simple commands the services by the service name and docker image url and the port on which it has to run is carried out.

5. Conclusion and Future work

Thus in this paper four different applications namely Random data generator, data splitter, temperature service and battery service is done using knative. With knative the autoscaling is done down to zero . It has also several other features such as Concurrency, Container freezer and Traffic Splitting which can be extended as future work.

References

1. Sarah R Nadaf, H. K. Krishnappa, "Kubernetes in Microservices", International Journal of Advanced Science and Computer Applications, https://doi.org/10.47679/ijasca.v2i1.19, Vol. 2, No. 1: March 2023 pp 7-18.
2. Aneta Poniszewska-Maranda,Ewa Czechowska,"Kubernetes Cluster For Automating Software Production Environment"Sensors 2021,21,1910.:Pages:1-24,http://doi.org/10.3390/s21051910
3. Google Cloud Platform,"An update on container support on google cloud platform",https://cloudplatform.googleblog.com/2014/06/an-update -on-container–support-on-google-cloud-platform.html,2014.
4. Kubernetes,"Borg:The predecessor to Kubernetes," http://kubernetes.io/blog/2015/04/borg-predecessor-to-kubernetes/, 2015.
5. L.Versluis,M.Neacsu and losup ,"A trace-based performance study of autoscaling workloads of work-flows in datacenters."in Proc.IEEE/ACM CCGRID,2018
6. Nima Kaviani,Dmitriy Kalinin,Michael Maximilien, "Towards Serverless as Commodity: a case of Knative", Proceedings of 5th International Workshop on Serverless Computing December 2019 WOSC '19: Pages:13–18, doi.org/10.1145/3366623.3368135
7. Sadjad Fouladi, Riad S Wahby, Brennan Shackle, Karthikeyan Vasuki Balasubramaniam, William Zeng, Rahul Bhalerao, Anirudh Sivaraman, George Porter, and Keith Winstein. 2017. Encoding, fast and slow: Low-latency video processing using thousands of tiny threads. In 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17). 363–376
8. Pedro Garcia Lopez,"Triggerflow:Trigger-based Orchestration of Serverless Workflows"june 2020,pg-1-13
9. Josep Sample,"Triggerflow:Trigger-based Orchestration of Serverless Workflows"june 2020,pg-1-13
10. Pnina Soer, Annika Hinze, Agnes Koschmider, Holger Ziekow, Claudio Di Ciccio, Boris Koldehofe, Oliver Kopp, Arno Jacobsen, Jan Surmeli, and Wei Song. 2019. From event streams to process models and back: Challenges and opportunities. Information Systems 81 (2019), 181–200.