International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 07 Issue: 06 | June 2020 www.irjet.net p-ISSN: 2395-0072

Review of Kubernetes Cost-Efficient Architecture and Federation Orchestration of Cloud based Applications

Keshav Bharat[1], Ganashree K. C.[2]

[1]B.E. Department of Computer Science and Engineering, R.V. College of Engineering
[2]Assistant Professor, B.E. Department of Computer Science and Engineering, R.V. College of Engineering

---

Abstract - *In this age of huge data processing and cloud computing with time critical applications we need a tool which can guarantee the processing capabilities. One of the cost-efficient ways is to use framework for the efficient orchestration of containers in cloud environments to run these applications. In this architectural style, small and loosely coupled modules are developed, deployed, and scaled independently to compose cloud-native applications. Kubernetes is an open source platform that defines a set of building blocks which collectively provide mechanisms for deploying, maintaining, scaling, and healing containerized microservices. Thus, Kubernetes hides the complexity of microservice orchestration while managing their availability. We also reviewed Kubernetes Federation, which allows developers to increase the responsiveness and reliability of their applications by distributing and federating container clusters to multiple service areas of cloud service providers. To improve the high availability factor, we also looked into various models with auto-scaling technology.*

Key Words: **Kubernetes, Cloud Architecture, Cloud Application, Federation Architecture, Pod scheduling, Autoscaler, Cloud Resources**

1.INTRODUCTION

The Orchestration of the containers in the manner to financially and viably make the cloud platform available is the major reason to use these infrastructure tools. Resource scheduling and rescheduling policies [1], autoscaling algorithms that enable the creation of elastic virtual clusters to improve the utilization of the available virtual resources and reduce operational cost for the provider. The major models are based on cost-efficiency (pricing), reliability (fault-tolerability), and QoS. These orchestrations can be used as the Federation models [2]. The key points cater for federation and monitoring functions across different cloud service areas. Kubernetes Federation allows developers to increase the responsiveness and reliability of their applications along with auto-scaling capability of the configured system through various scenarios making it a tough competitor for other topologies. Further along methods to automatically form and monitor Kubernetes Federation, given application topology descriptions can be employed to fully take advantage of the Orchestration.

The virtual private cloud (VPC) can be formed and function properly if the SLA is provided for the availability of the containerized application running in the Orchestration tool. Small and loosely coupled modules are developed, deployed, and scaled independently reaching the collective goal to enable high availability for microservices [3]. But in certain cases, the service outage for applications managed with Kubernetes is significantly high due to the size of the cluster, application's complexity etc. To deal with this "Microservice Based Architecture: Towards High-Availability for Stateful Applications with Kubernetes" proposes a State Controller that allows for state replication and automatic service redirection to the healthy entities through management of secondary labels, thus ensuring the high-availability.

Many of the questions based on the downtime of the Kubernetes services can be answered by answering the following research question (RQ) [4]:

- RQ1: What is the level of availability that Kubernetes can support for its managed stateful microservices solely through its repair features?
- RQ2: What is the impact of adding redundancy on the availability achievable with Kubernetes?
- RQ3: What is the availability achievable with Kubernetes under its most responsive configuration?
- RQ4: How does the availability achievable with Kubernetes compare to existing solutions?

The availability can be maintained if the cloud is scaled as per the application's requirement, scaling of the resources is very essential to maintain the cost to service ratio [5]. The paper talks about study of the existing auto-scaling strategy of Kubernetes along with various models for auto-scaling based on: Initial Container Placement, Dynamic Rescheduling, Dynamic Virtual Cluster Provisioning, Application QoS Management, Resource Consumption Estimation.

2. SYSTEM ARCHITECTURAL ELEMENTS AND CHALLENGES

2.1 System Architecture

The main problems which all the cloud architectural frameworks faces are pricing, fault-tolerability, and QoS-aware scheduling or rescheduling algorithms and scaling policies for containerized clusters in Clouds. The cluster managers interact with the following components to support cost-efficient and QoS-aware resource management:

Scheduler: Allocates the pawnable jobs to the cluster resources. This is done while considering various factors like resource pricing, application fault-tolerability, QoS of the applications, resource utilization, and cost. The scheduler is a managed service in case of some cloud providers where it can't be fine-tuned manually.

Autoscaler: If the scheduler finds that there are not enough resources available to allocate to critical jobs, it will interact with the auto-scaler to provide additional resources. The auto-scaler will then decide the type and the amount of resources to be added to the virtual cluster and then provision the new resources through calling the API of the underlying Cloud provider. The autoscaler is also responsible for migrating or terminating containers running on a specific host in order to shut down the host at the end of the host's billing period if the cluster is underutilized. The killed containers will be rescheduled by the scheduler immediately after their termination or be queued and rescheduled later.

Task Launcher: This entity is responsible for launching containers on specific machines and specifying the amount of resources that should be allocated to each container.

Resource Monitor: Monitors the real-time resource consumption such as CPU and memory on each host in the cluster and provides the information to the scheduler to make resource allocation decisions.

Task Monitor: This component is responsible for auditing a running containerized task by recording its resource consumption and monitoring its QoS metrics. This information aides in detecting faults or QoS violations and enables the system to make better scheduling or relocation decisions.

Resource Estimator: This module is used to predict and estimate the amount of resources such as memory and CPU that a container consumes at different points in time. It aims to reduce the framework's reliance on the amount of resources requested by users when submitting their applications.

Accounting: Maintains the actual usage of resources by requests to calculate real-time usage costs. Historical usage information can also be used to improve service allocation decisions.

Cloud Resources: Infrastructure as a Service Clouds provide the main compute infrastructure for the deployment of the containerized applications. VMs can be leased on demand and their prices vary based on their capabilities and their pricing model.

2.2 Architectural Elements And Challenges

In addition to these resource management tools there are other various key elements in the system architecture:

Initial Container Placement: Since container migration has to be conducted in a stop and resume manner, its working depends on the nature of the applications. For some applications the state can't be saved and hence the container can't be stopped in between and shifted to another node. For these containers, the management platform can only release the resources they use after they completed execution. Therefore, without live migration support, the initial placement algorithm needs to be aware of the application characteristics and as much as possible. Currently, there is no such algorithm is in service. Their target is to simply assign the containers onto as few VMs as possible. Besides, the placement algorithm in Cloud should also be aware of the heterogeneities of the underlying Cloud resources, including different pricing models, locations, and resource types and sizes. The paper talks about how it is required to filter unqualified resources and propose new resource affinity models to rank the resources when provisioning each framework.

Dynamic Rescheduling: The quality of the placement of the containers may degrade as the time passes with workload fluctuations, launchings of containers, and terminations of containers. Therefore, it is necessary to optimize the placement of containers during runtime through migration. The scheduler can use the resource monitor to check and place the jobs as per the VM capabilities and QoS.

Application QoS Management: Few applications have specific QoS requirements. For instance, long-running services commonly have to serve a minimum amount of request per time unit or have stringent latency requirements. Batch jobs on the other hand can have a deadline as a time constraint for their execution or may need to be completed as fast as possible. We can monitor CPU utilization, and check if a predefined threshold is exceeded, in that case another instance of the service is launched.

Resource Consumption Estimation: When we talk about long-running services, estimating the amount of resources they consume over time can aid in the efficient use of the cluster resources. Cloud providers such as Borg monitors their resource consumption over time and based on the collected data, predicts their actual resource consumption.

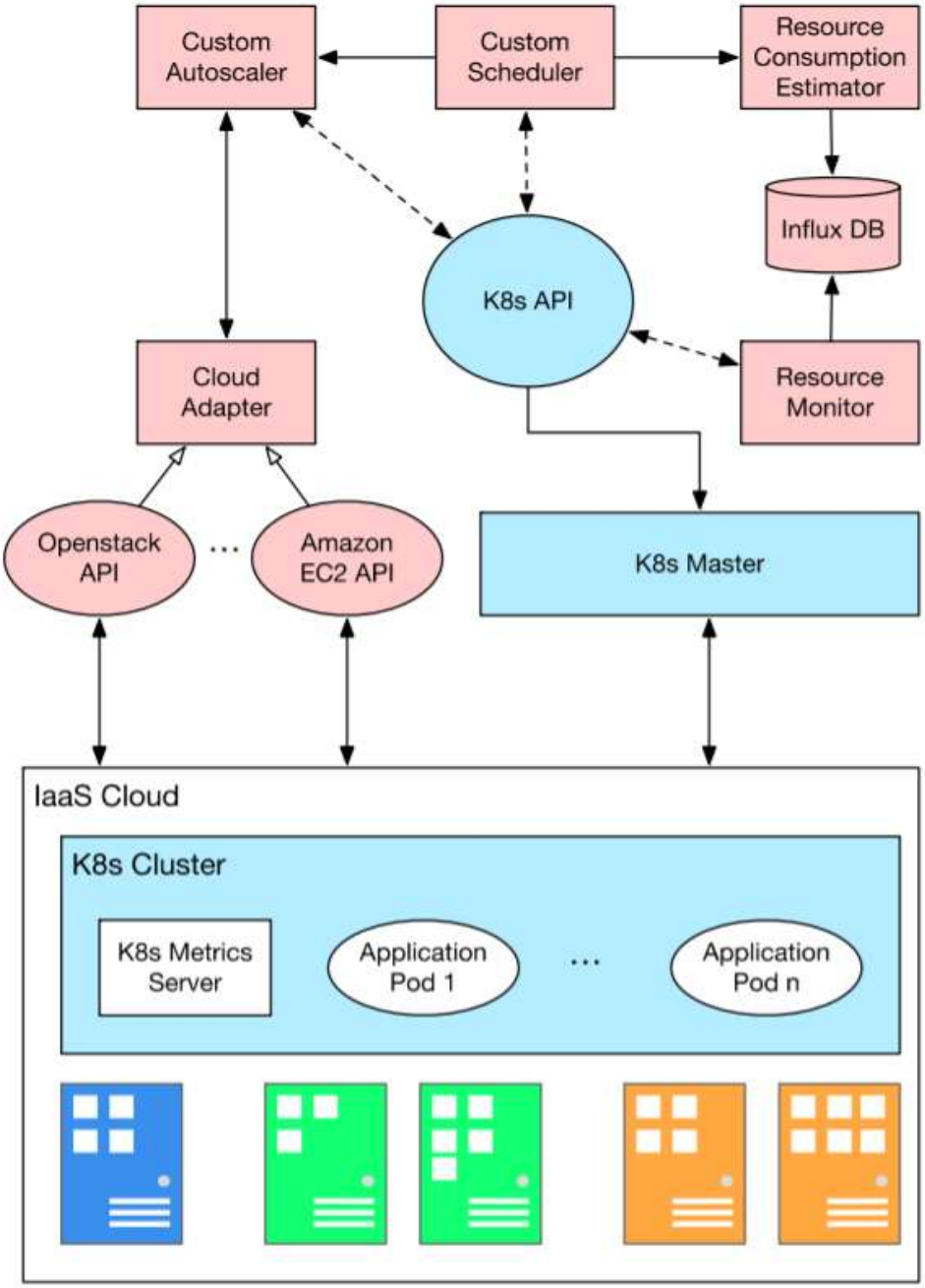

Fig -1: System prototype architeture

3. OVERVIEW OF KUBERNETES FEDERATION CONTAINERIZED SERVICE ORCHESTRATION TECHNOLOGY

Docker [6] containerizes individual processes and allows them to be run within a separated lightweight execution environment called a container. Kubernetes [7] is an open-source container orchestration tool that automatically installs and manages a cluster of Docker containers. Kubernetes includes the following elements:

- Kubernetes pod: this is an essential building block of Kubernetes, usually containing multiple Docker containers.
- Kubernetes node: this represents a VM (Virtual Machine) or physical machine where the Kubernetes pods are run.

• Kubernetes cluster: this consists of a set of worker nodes that cooperate to run applications as a single unit. Its master node coordinates all activities within the cluster.
• Kubernetes Federation: this is a cluster of clusters, i.e., viewed as a backbone cluster that combines multiple Kubernetes clusters. Figure 1 shows an example of the Kubernetes Federation architecture. Kubernetes provides a flexible, loosely coupled mechanism for service delivery. The federation application program interface (API) server interacts with the cluster through the federation controller manager. Which is responsible for exposing the API, scheduling the deployments, and overall cluster management.

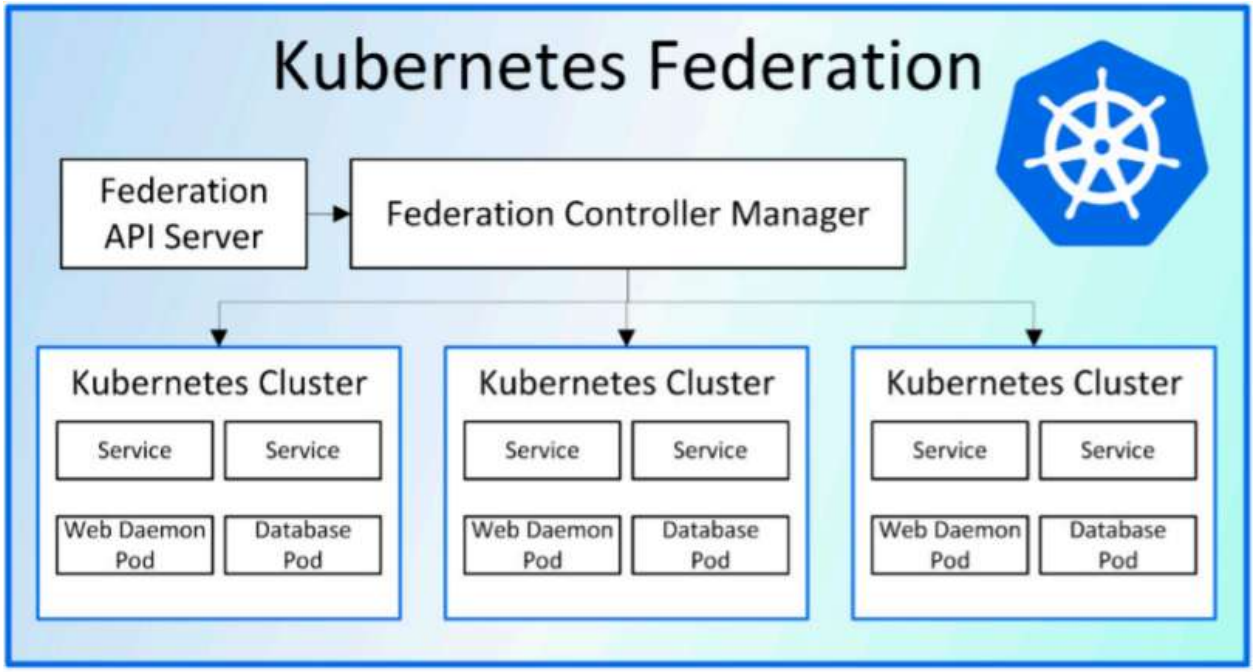

Fig -2: Example of the Kubernetes federation architecture. The federation API server automatically manages the subordinate Kubernetes clusters by providing the API that is equivalent to Kubernetes API

4. FEDERATION FRAMEWORK

4.1 Architecture

The clusters are deployed across the cloud provider's different service areas and federated to increase the reliability and responsiveness of the cloud service. The architecture aims towards enabling the following functions:

1. Automation of the distribution, assignment and federation of container clusters.
2. Automation of the service status management by defining Kubernetes horizontal pod auto-scaler (HPA).
3. Enabling the identification of the entire service topology of the application as well as monitoring by allowing the cloud orchestration tool to access the information of the Kubernetes components.

Key interactions between the Orchestration System and Kubernetes Federation are:

1) Defining a Kubernetes federation according to the certain standards makes it easy to communicate Kubernetes clusters information and join them to the federation.
2) The system also supports Kubernetes HPA, which auto-scales the number of pods across the entire Kubernetes federation. The HPA component type is defined to be associated with the K8s API Mapper so that AutoscalingV1Api requests can be sent to the Kubernetes Federation.
3) A monitoring agent allows the system to monitor the status of the Kubernetes components. A monitoring agent installed on each pod allows direct monitoring access to individual pods.

4.2 Definition of Horizontal Pod Autoscaler (HPA) Components

Using Kubernetes HPA enables an automatic upscaling and downscaling of the number of pods according to the workload of the service. Kubernetes administrators can set the minimum and maximum of the number of pods by providing an HPA option.

HPA can be extended to generate the pods for Kubernetes Federation as well in the same way. The number of pods to increase or decrease is communicated to each Kubernetes clusters. To use an HPA, the proposed system used a YAML-based definition.

4.3 Monitoring the Information of Kubernetes Components

Some information of the Kubernetes components cannot be determined when their creation is requested and can only be obtained after some time has elapsed. For example, a Kubernetes service component that provides Kubernetes components has the IP value of null at the time of its creation. After sometime, they may be given an IP address that can be read from the service component. A new and efficient method for receiving the information has been devised for the situations that require an active retrieval of information from Kubernetes components.

5. CONCLUSION

This paper review mainly talks about the architecture of the Kubernetes orchestration which ensures the high cost-efficiency based on various scheduling algorithms based on number of criteria and resource tools by constantly monitoring them and keeping a check on pricing, fault-tolerance, and QoS. Along with this a federation architecture is also reviewed which aims towards providing a fail-safe multi region support service with automated tools.

ACKNOWLEDGEMENT

I would like to extend my gratitude towards my guide Prof. Ganashree K. C. for her continued support and guidance on my work in this paper. I would also like to show my gratitude to Department of Computer Science and Engineering, RV College of Engineering for giving me the opportunity to write this paper with complete support.

REFERENCES

[1] Buyya, R., Rodriguez, M. A., Toosi, A. N., & Park, J. "Cost-efficient orchestration of containers in clouds: a vision, architectural elements, and future directions" In Y. Sri Rahayu, R. Ekawati, N. Suprapto, E. Asep Bayu Dani Nandiyanto, S. Chendra Wibawa, S. Fiangga, & M. Jakfar (Eds.), 2nd Mathematics, Informatics, Science and Education International Conference (MISEIC), July 2018 M. Young, The Technical Writer's Handbook. Mill Valley, CA: University Science, 1989.

[2] Kim, Dongmin & Muhammad, Hanif & Kim, Eunsam & Helal, Sumi & Lee, Choonhwa. "TOSCA-Based and Federation-Aware Cloud Orchestration for Kubernetes Container Platform." *Applied Sciences*. Vol 9, 2019 K. Elissa, "Title of paper if known," unpublished.

[3] Vayghan, Leila & Saied, Mohamed & Toeroe, Maria & Khendek, Ferhat. "Kubernetes as an Availability Manager for Microservice Applications.". 2019

[4] L. Abdollahi Vayghan, M. A. Saied, M. Toeroe and F. Khendek, "Microservice Based Architecture: Towards High-Availability for Stateful Applications with Kubernetes," 2019 IEEE 19th International Conference on Software Quality, Reliability and Security (QRS), Sofia, Bulgaria, 2019, pp. 176-185, doi: 10.1109/QRS.2019.00034.

[5] Zhao, Anqi & Huang, Qiang & Huang, Yiting & Zou, Lin & Chen, Zhengxi & Song, Jianghang. "Research on Resource Prediction Model Based on Kubernetes Container Auto-scaling Technology." *IOP Conference Series: Materials Science and Engineering*. 2019

[6] Docker—Build, Ship, and Run any App, Anywhere. Available online: https://www.docker.com (accessed on 6 January 2019).

[7] Kubernetes. Available online: https://kubernetes.io (accessed on 6 January 2019)