Volume: 9 Issue: VII Month of publication: July 2021

DOI: https://doi.org/10.22214/ijraset.2021.36648

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.429
Volume 9 Issue VII July 2021- Available at www.ijraset.com

Centralised Cleanup Service for Kubernetes and Cloud Resources

Pranava Bhat[1], Darshan Narayana Murthy[2], Nagaraj G Cholli[3]
[1, 3]R.V. College of Engineering, Bengaluru, India
[4]Cloudera, Bengaluru, India

Abstract: The architectural style of developing a software application using loosely coupled and highly cohesive services can be termed as microservices architecture. The microservices allow agile software development and enable businesses to build and deliver applications quickly. To achieve the benefits of microservices, an underlying infrastructure that supports them must exist. This includes CI/CD pipelines, execution environments like virtual machines and containers, logging and monitoring, communication mechanisms, and so on.

Containers are lightweight, enable multiple execution environments to exist on a single operating system instance, and provide isolation. Container Orchestration Engines such as Docker swarm or Kubernetes automate deployment, scaling, fault tolerance, and container networking. Many organizations use containers to spawn resources in public or private clouds. Different engineering teams perform various kinds of tests by bundling the test code and dependencies into containers. However, cleaning up these containers is necessary for the efficient utilization of hardware resources. This paper discusses the need and benefits of a centralized cleanup service for Kubernetes and cloud resources. It analyzes the value additions this service can bring to the software development process of large organizations.

Keywords: Containers, Microservices, Kubernetes, Namespaces, Time to Live, Docker, REST API

I. INTRODUCTION

Migration towards the cloud has been the trend in the industry in the past decade. In this context, the microservice architecture style of software development has received significant attention from the industry. Microservice architecture involves developing small, loosely coupled services that can be deployed and scaled independently of other services. In modern industry practices, an application is a collection of microservices that handle separate business functionalities. Each microservice runs in its own process and communicates through lightweight mechanisms, often using APIs. Monolith architecture makes the application too large and complex to understand thoroughly to make any changes. The entire application must be redeployed after each update, and it will slow down the startup time. It is difficult to scale such applications when different modules have conflicting dependencies. Microservices solve these drawbacks of monolith application. The failure in a single function is not contagious in microservices as compared to monolith applications, which helps in faster debugging and troubleshooting [1].

Containerization technology has contributed to building microservices. Google's Borg is a pioneer of this technology. Google's Borg system is a cluster manager. It runs thousands of jobs from different applications across clusters residing in thousands of machines [2]. Google Borg popularized containers and paved the way for the development of many modern container orchestration tools.

At the core of the container, technologies are Control Groups (cGroups) and namespaces. Control Groups work by allowing the host to share and also limit the resources each process or container can consume. This is important for resource utilization and security, as it prevents denial of service attacks on the host's hardware systems. Namespaces offer another form of isolation in the way of processes. Processes are limited to see only the process ID in the same namespace. For example, a network namespace would isolate access to the network interface and configurations, which allows the separation of network interfaces, routes and firewall rules [3].

The isolation and granular focus that containers bring have paved the way for microservices. It is facilitated by an open-source platform named Kubernetes. This enables deployment, maintenance, scaling, and failure recovery of containerized applications [4].

The DevOps domain has been revolutionized by technologies like Docker and Kubernetes which allows developers to build any architecture. Docker is used to build, ship and run any application anywhere. Images can be built by Docker. One or more containers can be spawned using an image. Docker manages an online repository named Dockerhub where indexed images are saved and managed [5]. An instruction file named Dockerfile is used to build an image in an incremental manner [6][7]. Images are built in a layered fashion.

An Apache web server image (httpd) can be used as a base image and on top of it, components like specific versions of PHP can be installed as additional layers. We can add new packages, change configuration files, create new users and groups, and even copy files and directories to the image [8].

Kubernetes can be used to orchestrate these containers. Documentations for installing and using Docker and Kubernetes are now widely available [9]. Many large companies are moving towards kubernetes to scale and manage the containers. The containerized environment allows for easier scalability of the infrastructure. Portability is another advantage. It is easier to handle containers running in public cloud, private cloud or hybrid cloud using kubernetes. Containerization gives you the agility to move faster to the market. The community around cloud native tools has made the job easier for many developers. Using kubernetes helps you better optimize your resources. Organizations are now able to build and deploy platform-independent advanced applications and systems in code using Kubernetes APIs [10].

However, Kubernetes comes with some challenges as well. One of them is the termination of Kubernetes resources. Running Kubernetes can be very expensive, especially when it is done inefficiently. Companies should start to also consider costs soon because this can save a lot of money and prevent the spread of bad practices. Practices such as auto-scaling, cost monitoring, resource limits can help towards this aspect. An automatic sleep mode and cleanup for unused Kubernetes namespaces and Clusters are measures to eliminate idle resources [11].

Some open-source projects (Eg. Kube janitor) cleans up (deletes) Kubernetes resources on a configured TTL (time to live) or a configured expiry date (absolute timestamp) [12]. But these are not suitable for specific requirements of organizations and do not automate the deletion process. A centralized cleanup service can monitor the kubernetes resources and terminate them when the deletion criteria are sufficient. Organizations that use kubernetes jobs to spawn resources in public or private clouds can tie up the deletion of kubernetes and cloud resources through the centraised cleanup service. These measures greatly reduce the cloud usage cost and also prevents the exhaustion of compute resources on kubernetes nodes.

II. TERMINATION MECHANISM IN KUBERNETES

This chapter discusses the termination mechanisms of some of the common kubernetes resources.

A kubernetes Job creates one or more Pods and executes them until a specified number of pods successfully terminate. A job completion stops the creation of new pods, but the existing pods are not deleted either. Errors, warnings and other diagnostic output of pods can be viewed by not deleting the terminated pods. Users can view the logs by keeping the terminated pods around and delete them manually whenever they wish. Jobs can be deleted using *kubectl delete* which also deletes all the pods it created. Specifying the backoff limit and active deadline seconds in the job specification are other ways of terminating a job. Backoff Limit indicates the number of times a job retries before failing. All the pods, even if it is running, will be terminated once the job reaches active deadline seconds. Pods are an abstraction of containers and the smallest deployable units of computing that can be created and managed in Kubernetes. A pod deletion request waits for a specified amount of time to perform deletion to allow graceful shutdown. In case of failed pods, the corresponding API objects remain in the cluster. They are removed only when a human or controller process manually deletes them which is a cumbersome task in big engineering teams.

One of the recent features is the TTL controller that limits the lifetime of resource objects that have finished execution based on a time-to-live value. Currently, this mechanism is available only for jobs.

III.NEED FOR CENTRALIZED CLEANUP SERVICE

Many enterprises are switching to multi-tenant Kubernetes clusters from their existing mechanisms to run their workloads. Applications from different teams, divisions, clients, or environments run on a single Kubernetes cluster. This multi-tenancy has advantages such as easier manageability, efficient utilization of resources and reduced fragmentation. Different engineering teams in organizations create resources such as jobs and pods in these clusters. Most of the termination mechanisms provided by kubernetes do not delete the resources. It is infeasible for developers to manually delete jobs/pods using kubectl commands. A generic cleanup implementation using cron jobs is not possible where the workloads are complicated. With the centralized cleanup service, labels and annotations of resources can be used to filter specific resources and deletion criteria of such resources can be pre-defined.

Engineering teams such as Quality Engineering, Performance team in organizations run thousands of system and unit tests in containers and many of them will end up in Error state because of various issues. Manually cleaning up such erroneous containers is a tedious process. These problems can be addressed by this proposed centralized service. Labels and annotations of resources can be used to filter specific resources and deletion criteria of such resources can be pre-defined. This reduces resource and memory issues in kubernetes, prevents hard eviction of pods from the nodes and helps in optimizing application and cluster performance

Many organisations use kubernetes jobs/cron jobs to spawn clusters in public or private clouds. Cluster is a group of nodes hosted on virtual machines and connected within a cloud. Name of such clusters and also their lifetime can be indicated in job annotations. These resources can be deleted by the cleanup service using available APIs. The method of integrating the deletion of kubernetes and cloud resources reduces the cloud cost and paves way for efficient utilization of cloud resources.

IV. DESIGN

The following section explains the design of the proposed system.

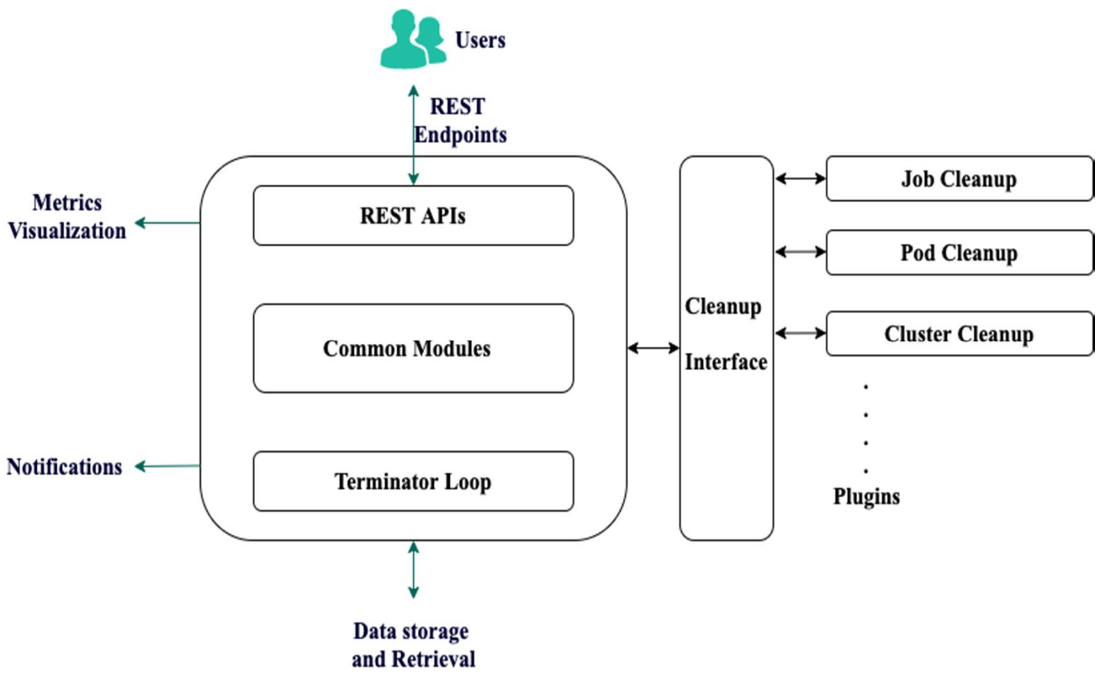

Fig 1. High level design diagram

A. Automated Deletion - Terminator Loop

The terminator loop lists the resources in all namespaces. Resources of interest can be filtered by their naming, labels and selectors. For example, System Test jobs can be named with the prefix 'st-' and jobs which create clusters can be named with prefix 'cc-'. Jobs which are in running state are not considered for deletion. The information required for the identification of deletion criteria such as lifetime, creation timestamp will be included in the annotations by the resource creator. The deletion of cluster creation jobs involves the deletion of clusters in the cloud as well. The deletions will be cascading. Kubernetes provides APIs in different languages to support management of resources in a programmatic fashion.

The automated deletion module runs periodically, checks for pre-defined deletion criteria of resources and deletes them if the criteria is met. The cleanup interface refers to an abstract base class where the necessary functionalities are defined. The subclasses which are inherited from this interface contain the actual cleanup logic. The architecture is plugin based. Each kind of resource can have its own cleanup implementation based on the requirements. This design allows the enterprises to add and manage new plugins when there is a need to terminate new kinds of resources. The service can be deployed as a kubernetes pod which will improve the scalability and ensure high availability.

B. User Interaction - REST APIs

The centralized cleanup service exposes REST Endpoints to make the service interactive. Users can query the datastore where information related to the deletions are stored. Details about resources where the deletion failed can help in analyzing the root cause. Developers can delete a resource before its lifetime. This call to delete endpoint is responsible for end-to-end cleanup. For example, a request to delete a kubernetes job must delete all its dependents and also the cloud resources created by it. Provision to extend the lifetime of a resource will be provided. The job specification in kubernetes can be modified by invoking specific endpoints.

C. Components

The proposed design involves many components that are interconnected. The plugin based architecture allows easy integration of new resource plugins. This seamless inclusion of cleanup plugins make the service more extensible and generic.

1) *Cleanup Interface:* It is a class which contains common attributes of resources like Resource name, Creation Timestamp, Deletion Timestamp, Resource Owner etc. Abstract methods in this class are overridden in subclasses. Child classes act as plugins. Eg. Job Cleanup implementation defines plugin-specific logic for checking deletion criteria and resource deletion of kubernetes jobs.
2) *Common Modules:* These modules provide functionalities that are common for both Terminator Loop and REST APIs. The modules must ensure low coupling and maximum cohesivity. A kubernetes module can perform calls to the api server. A query module for retrieving useful data from the datastore. Modules for visualization of metrics, for alerting the users in the events of resource deletions will be a value addition for the developers. Different tools such as Prometheus, MongoDB etc. can be used for these purposes.

V. ADVANTAGES

1) *Cost Reduction:* Cloud service providers charge the customers based on cloud resource usage. Releasing the resources after the job completion reduces unnecessary usage of cloud resources and reduces cost.
2) *Improves Productivity of Developers:* Automatic handling of termination process for Kubernetes resources improves productivity of the developers by saving time. Users need not worry about the risks of exceeding resource quotas for their namespace.
3) *Fastens the Software Development Process:* Bottlenecks for resource utilization rarely occur and developers need not depend on others for resource allocation. This high availability of resources will fasten the software development process in enterprises.
4) *Customizability:* REST Endpoints can offer customizability to the users. Manual deletion of kubernetes objects and their dependent resources from an UI requires no technical expertise and it binds a multi step process into a single one. Extension in the lifetime can help in obtaining more time for debugging which is needed in certain circumstances. Users can define custom deletion criteria without the restriction of using spec. active Deadline Seconds or .spec.ttlSeconds AfterFinished.

Hardware resources available in the company's data center are utilized in an optimized manner by using this service. It also eases the manageability for maintainers of different applications/services.

The proposed solution is not restricted to Kubernetes or Cloud resources. It is extensible and can be easily used to cleanup any kind of resources that may need a regular garbage collection within the company. All these advantages will assist different engineering teams of large enterprises.

VI. CONCLUSION

Kubernetes is gaining widespread adoption among enterprises as a container orchestration engine. Containers are getting used to deploy resources in public and private clouds. However, care must be taken when using kubernetes and cloud resources as inefficient handling of them can be very expensive. The termination mechanisms provided by the kubernetes do not meet the requirements of large engineering teams.

In this paper, we have proposed a design of a centralized cleanup service that is responsible for the cleanup of kubernetes objects and all of its dependents including cloud resources spawned by them. Publicly available APIs from kubernetes and cloud vendors can be used for graceful termination of resources. We have also discussed the benefits from this service. The automated deletion support and the REST endpoints reduces the cloud usage cost and improves productivity of users. Implementation of this centralized cleanup service will be a great value addition for the organizations.

VII. ACKNOWLEDGMENT

We acknowledge Cloudera for providing the opportunity to work on this project - 'Centralized Cleanup Service for Kubernetes and Cloud Resources' as part of internship. We thank R.V. College of Engineering, Bengaluru and its faculty for the support and guidance in writing this paper.

REFERENCES

[1] Vayghan, Leila Abdollahi, et al. "Kubernetes as an availability manager for microservice applications." arXiv preprint arXiv:1901.04946 (2019).
[2] Abhishek Verma, Luis Pedrosa, Madhukar Korupolu, David Oppenheimer, Eric Tune, John Wilkes, Large-scale cluster management at Google with Borg, Google Inc.
[3] EuroSys'15, April 21–24, 2015, Bordeaux, France
[4] Jonathan Baier. 2015. <i>Getting Started with Kubernetes</i>. Packt Publishing.
[5] L. Abdollahi Vayghan, M. A. Saied, M. Toeroe and F. Khendek, "Deploying Microservice Based Applications with Kubernetes: Experiments and Lessons Learned," 2018 IEEE 11th International Conference on Cloud Computing (CLOUD), 2018, pp. 970-973, doi: 10.1109/CLOUD.2018.00148..
[6] Projects in DevOps: Build real-world Processes' by Edunoix Learning Solutions, Online Web Courses
[7] Documentation of 'PaaS and Container clouds' by John Rofrano, IBM, NY University, 2015
[8] 'DevOps; Puppet, Docker and Kubernetes – Learning path' by Thomas Uphill, Arundel, Khare, Saito, Lee and Carol Hsu, Packt Publications, First Edition, 2017
[9] Shah, J., & Dubaria, D. (2019). Building Modern Clouds: Using Docker, Kubernetes & Google Cloud Platform. 2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC). doi:10.1109/ccwc.2019.8666479
[10] Vohra, D. (2016). Kubernetes Microservices with Docker. United States: Apress.
[11] Weissman B., Nocentino A.E. (2021) A Kubernetes Primer. In: Azure Arc-Enabled Data Services Revealed. Apress, Berkeley, CA. https://doi.org/10.1007/978-1-4842-6705-9_1
[12] Article on 'How to reduce your kubernetes cost' by Daniel Thiry, Sept 28, 2020
[13] Open source project - hjacobs/kube-janitor available in (https://codeberg.org/hjacobs/kube-janitor)