

Enhancement of observability using Kubernetes operator

Prerana Shenoy S. P.¹, Soudri Sai Vishnu², Ramakanth Kumar P.¹, Sahana Bailuguttu²

¹Department of Computer Science and Engineering, RV College of Engineering, Bangalore, India

²Department of Electronics and Communication Engineering, RV College of Engineering, Bangalore, India

Article Info

Article history:

Received May 4, 2021

Revised Nov 14, 2021

Accepted Nov 29, 2021

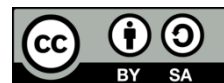
Keywords:

Cloud computing
Evaluation metrics
Helm
Kubernetes
Operator-SDK

ABSTRACT

Observability is the ability for us to monitor the state of the system, which involves monitoring standard metrics like central processing unit (CPU) utilization, memory usage, and network bandwidth. The more we can understand the state of the system, the better we can improve the performance by recognizing unwanted behavior, improving the stability and reliability of the system. To achieve this, it is essential to build an automated monitoring system that is easy to use and efficient in its working. To do so, we have built a Kubernetes operator that automates the deployment and monitoring of applications and notifies unwanted behavior in real time. It also enables the visualization of the metrics generated by the application and allows standardizing these visualization dashboards for each type of application. Thus, it improves the system's productivity and vastly saves time and resources in deploying monitored applications, upgrading Kubernetes resources for each application deployed, and migration of applications.

This is an open access article under the [CC BY-SA](https://creativecommons.org/licenses/by-sa/4.0/) license.



Corresponding Author:

Prerana Shenoy S. P.

Department of Computer Science and Engineering, RV College of Engineering

Mysore Rd, RV Vidyaniketan, Post, Bengaluru, Karnataka, India

Email: preranashenoysp.cs17@rvce.edu.in

1. INTRODUCTION

Kubernetes is an extensible, open-source platform for managing containerized microservices that facilitates both declarative configuration and automation [1]-[3]. It provides a rapidly growing ecosystem that allows extension capabilities in various forms, one of them being Kubernetes Operators. Kubernetes allows applications deployed on the cluster to expose system metrics that are utilized by the application, like CPU utilization, memory usage, network bandwidth, request rate, etc., depending on the type of application. When operating on a cloud datacenter, it is very important to be able to generate the metrics, visualize them using efficient visualization solutions, and alert the system based on the unwanted behavior of the metrics. These metrics are essential to track and monitor the health of the data center. Any unhealthy behavior such as a sudden spike in CPU or memory utilization must be monitored and alerts triggered for appropriate action to be taken.

Kubernetes has been extended and efficiently used over the years to suffice the needs and capabilities of upcoming trends. It can be used to provision a generic platform that facilitates dynamic resource provisioning as shown in [4], [5] mentions how cost-efficient orchestration can be done with containers in cloud environments. It specifically investigates pricing models of the acquired resources, fault tolerability of applications, and Quality of Service (QoS) requirements of the running applications. [6] discusses various cloud computing characteristics such as services, storage, and deployment models. It also discusses several security threats, storage issues, and challenges to cloud computing. Prometheus is used for monitoring along with Alert Manager, which is used for alerting based on rules established for those

monitored metrics. Its future enhancements include estimating the number of resources such as CPU and memory that applications consume over time for efficient use of the cluster resources. Kubernetes can be extended to create frameworks that can integrate Kubernetes with Prometheus and Grafana [7], [8]. Prometheus and Grafana can also be leveraged on a scalable agentless system for monitoring and alerting in any cloud hosting environment as shown in [9].

We have chosen to use Kubernetes' capabilities to expose HTTP endpoints for metrics as it is simpler and most used. The capabilities of Kubernetes can also be extended in many ways to improve the efficiency of Kubernetes [10], [11]. The research is now moving towards operators due to its stability and capabilities. Kubernetes operator extends the capabilities of Kubernetes and can be tailor-made to suit a wide range of use cases [12]-[14]. We have developed an operator that extends the capabilities of Kubernetes by deploying the application that creates our custom resource, exposing its metrics for Prometheus Operator [15] to use and configure alerting rules based on them with Alert Manager [16]. It also deploys certain dashboards in Grafana based on the type of application. Service meshes are a dedicated infrastructure layer on microservices that do not impose any modifications. They support high performance, adaptability, and availability [17], [18]. We have used Istio ServiceMesh in our project to provide traffic management, observability, and security. We have decided to use Docker as our runtime container interface along with Kubernetes for Deployment based on [19]-[21].

Kubernetes requires every application to deploy various resources such as deployment, service, resources for visualization and monitoring. These resources have a standard template and only vary in metadata and certain application-specific parameters. Currently, a user needs to create these resources manually and deploy them to the cluster. This results in a lot of duplicated code being written manually and gives scope to human errors. Wrongly defined resources can affect the working of applications in Kubernetes. It is also essential to monitor the health and metrics of the deployed applications to ensure high availability. Users find it difficult to maintain all standard dashboards and alerting rules that need to be included for all applications of a specific type and monitor the correctness of the Kubernetes resources that are deployed. Currently, Kubernetes upgradation and migration is a huge task. It involves users editing lines containing version information in every resource associated with every application that is deployed in Kubernetes. This is a very time taking and tedious process.

To tackle the above problem, we have developed a Kubernetes Operator that automates the deployment of required resources and takes care of the generation of metrics, visualization of metrics in an open-source tool called Grafana, and configuration of alerting rules with Prometheus Operator for the metrics it exposes. For applications and microservices to use our operator, they need to define and deploy a custom resource of our operator. This custom resource contains information about the customization of resources that is required by the operator. The proposed operator saves resources of the system and improves the reliability and stability of the system by closely monitoring the applications. It contributes in three ways. First, it eliminates human error and improves the productivity of the system by automating the majority of code that was manually written before. Users can now write just the custom resource for the operator. Second, it improves the monitoring of applications deployed in the cluster by allowing users to standardize dashboards that visualize the metrics generated by them. It also allows standardizing alerting rules applied to these applications. Users can add additional dashboards and alerting rules based on their application. Third, it vastly saves time and resources while upgrading Kubernetes resources and migration of applications. Upgrading the API Version of any of the Kubernetes resources can be done only in one place in our operator, and all applications that have been deployed using our operator reflect the upgrade. The operator scans through the cluster to find custom resources in real-time and makes the changes.

2. METHOD

The Kubernetes operator is developed using the operator-SDK framework that uses Helm as its language. Helm is a package manager and a templating language for Kubernetes. It enables developers to easily package and deploy various types of applications and services onto Kubernetes clusters. Our operator deploys the application that exposes the metrics to Prometheus, visualizes those metrics in Grafana, and configures alerting rules for the metrics with Alert Manager. The Alert Manager handles alerts sent by the Prometheus servers. Alert Manager can be integrated with PagerDuty, which is a cloud service for alerting [22]-[24]. The operator was created by scaffolding a new project using the operator-SDK Command Line Interface (CLI). The operator uses its reconciling logic to create a new helm chart containing the resources needed or add to the existing one. In this way, the operator performs its functions using Kubernetes resources.

2.1. Proposed kubernetes operator architecture

The architecture of our operator in Figure 1 is used to facilitate contact centers but can be used by any cluster of microservices to enhance their observability in Kubernetes. The operator mainly consists of a custom resource (CR) and a custom controller. The custom resource will be discussed in-depth in the next section. Here we will discuss the custom controller as it is responsible for the tasks completed by the operator. The custom controller watches the cluster for changes to the custom resource in a loop called the control loop, which converts the current state of the cluster to its desired state as defined by the operator.

When a custom resource is created in the cluster, the operator triggers the custom controller to deploy all the resources that are needed by the desired state. We have designed our operator to deploy the following resources.

- Deployment: the application calling the CR will be deployed as a container along with all the necessary attributes taken from the user if necessary.
- Service: deployment exposes its metrics through this service using the port provided by the user.
- ServiceMonitor: this is a custom resource defined by the Prometheus Operator. It discovers targets based on matching labels and provides the metrics to Prometheus.
- PrometheusRule: this is a custom resource defined by the Prometheus Operator. It comprises the alerting rules for the metrics exposed and the rules can be standardized or not based on the requirement.
- VirtualService: defines a set of traffic routing rules to apply when a host is addressed. Each routing rule defines matching criteria for the traffic of a specific protocol.
- ConfigMap: the dashboards can be mentioned as JavaScript Object Notation (JSON) files in the configmap and these are automatically fetched by Grafana.

The Kubernetes API Server monitors these resources and ensures availability. For example, consider it creates a Deployment in the cluster to run the application. It further watches out for changes to this resource and immediately reflects any changes in a reliable manner. Kubernetes can also be used as availability manager, and its scheduling can be improved by [25].

When these resources are deployed by the operator, the following processes take place.

- Deployment runs the application with the defined configuration. These applications expose their metrics like CPU utilization, memory usage, and network bandwidth, at a defined endpoint using the service.
- ServiceMonitor discovers the targets based on matching labels and provides the metrics to Prometheus.
- Prometheus configures all the rules given by the PrometheusRule resource with the Alert Manager. Prometheus additionally lists the targets, rules, and status of the alerts in the Prometheus dashboard.
- Using the dashboards mentioned in the configmaps, Grafana dashboards are created. Each of the dashboards is saved as a JSON object and fed to the configmap. When a configmap is deployed, the sidecar in Grafana picks it up and renders the dashboard [26].

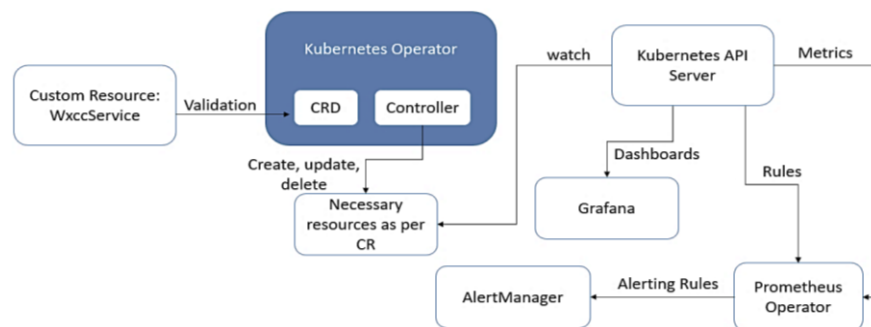


Figure 1. Proposed kubernetes operator architecture

The custom controller is a control loop that watches the cluster for changes to our operator's custom resource and makes sure that the current state matches the desired state. If the custom resource is deleted, all the resources that were initially created by the custom resource will be deleted by the operator. The operator also allows versioning if the schema defined in the custom resource definition is to be changed after the deployment of the operator. The user must define the new schema under a different version and provide the conversion strategy if needed. A conversion strategy is a webhook server that recognizes old custom resources in the cluster and converts it to the desired custom resource to the operator in the current version.

This will allow multiple versions to exist in the system together, and when all the custom resources are converted to the new version, the old version can be made invalid. The custom resource is first validated

against the custom resource definition. If it is valid, it starts creating the resources asked by the user for that application. The Deployment of these resources then leads to the monitoring of the application, as explained previously. If the metrics exceed the defined threshold, an alert is triggered by the Alert Manager [16].

2.2. Proposed custom resource

The schema of our custom resource is defined by the custom resource definition (CRD) of our operator. When a custom resource is deployed in the cluster, the operator validates it with the CRD, and if it is valid, it triggers the controller. If there are multiple versions, each of their schemas is mentioned in the CRD where one and only one version is marked as storage version. To enable a version, its Served flag can be enabled. WxccService CRD is shown in Figure 2.

```

apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  name: wxccservices.com.cisco.ccc
spec:
  group: com.cisco.ccc
  names:
    kind: WxccService
    listKind: WxccServiceList
    plural: wxccservices
    singular: wxccservice
  scope: Namespaced
  versions:
  - name: v1
    schema:
      openAPIV3Schema:
        description: WxccService is the Schema for the wxccservices API
        properties:
          apiVersion:
            description: 'APIVersion'
            type: string
          kind:
            description: 'Kind'
            type: string
          metadata:
            type: object
          spec:
            description: Spec defines the desired state of WxccService
            type: object
            properties:
              global:
                type: object
                properties:
                  appName:
                    type: string
                x-kubernetes-preserve-unknown-fields: true
                required: ["appName"]
              service:
                type: object
                x-kubernetes-preserve-unknown-fields: true
            deployment:
              type: object
              properties:
                image:
                  type: object
                  properties:
                    repository:
                      type: string
                    tag:
                      type: string
                    pullPolicy:
                      type: string
                  allOf:
                    - required: ["repository"]
                    - required: ["tag"]
                enable:
                  type: boolean
                x-kubernetes-preserve-unknown-fields: true
                required: ["enable"]
            x-kubernetes-preserve-unknown-fields: true
          status:
            description: Status defines the observed state of WxccService
            type: object
            x-kubernetes-preserve-unknown-fields: true
            type: object
          served: true
          storage: true
          subresources:
            status: {}

```

Figure 2. Example of custom resource definition

The example of our customer resource in Figure 3 gets validated with the CRD that leads to the deployment of all the necessary resources. All the resources are optional and can be opted out of if the user decides to deploy the Kubernetes resource on their own or has already been deployed. `appNamePrefix-appName-appNameSuffix` is used to name the resources uniquely. Maps like Service, deployment provide information for those respective resources. Istio provides information for the VirtualService and PrometheusOperator provides information for its custom resources like ServiceMonitor and PrometheusRule. Grafana defines the information needed to create the dashboards.

```

apiVersion: com.cisco.ccc/v1
kind: WxccService
metadata:
  name: wxccservice-sample
spec:
  global:
    appName: helloapp
    appNamePrefix: prefix
    appNameSuffix: suffix
  service:
    port: 8080
    ports: []
  deployment:
    image:
      repository: repo_name/image_name
      tag: valid_tag
      enable: true
      replicaCount: 2
    resources:
      limits:
        cpu: 1
        memory: 1Gi
      requests:
        cpu: 100m
        memory: 60Mi
  istio:
    gateway:
      domain: domain_name
      name: default-gateway
  prometheusoperator:
    servicemonitor:
      enable: true
      port: http
      path: prometheus/metrics
    prometheusrule:
      enable: true
      rules:
        - record: node_memory_MemFree_percent
          expr: 100 - (100 * node_memory_MemFree_bytes / node_memory_MemTotal_bytes)
  grafana:
    defaultDashboardsEnabled: true
    dashboardcms:
      annotations:
        strategy.spinnaker.io/versioned: "false"
        strategy.spinnaker.io/recreate: "true"
    dashboards:
      wxccservice-sample-http-metrics.json:
        #dashboard information

```

Figure 3. Example of custom resource–WxccService

3. RESULTS AND DISCUSSIONS

Applications in Kubernetes are deployed manually using “`kubectl create`” commands or through a CI/CD pipeline. To enable a monitoring and alerting system, additional resources are created along with the application deployment. All users have to manually ensure they have written the right code for each resource they are deploying and preserve the correctness of resources. If there is an API version upgradation of a Kubernetes resource, which often happens as the resources are evolving, such resources of all the applications have to be updated with the same. Using the operator we created, all the resources of all applications can be managed in one place including any API version upgradation, standard rules for all applications set by the organization, and correctness of the resources. If there are any updates to be done to the CRD or the operator logic, a new version of the operator can be involved and a smooth upgradation of the existing custom resource to the new one can be done, and the old version can be eliminated eventually.

For experimental analysis, the operator was deployed in a new namespace in the cluster, and access to other namespaces was given to it. The operator searched through all the namespaces to find its custom resources and verified it with its custom resource definition (CRD). After validating the custom resource, it successfully deployed the application, exposed its metrics for monitoring, and registered itself with the AlertManager for alerting unwanted behavior. It was easy to migrate existing applications on the cluster to be deployed and watched by our operator.

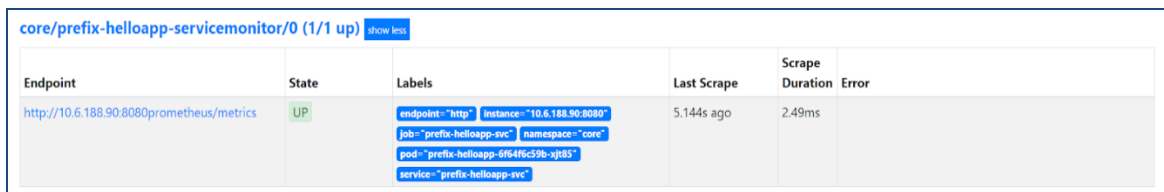
3.1. Experimental setting

We used Kubernetes version: 1.19.4, go version: go1.15.5 and docker-desktop version: 3.1.0 to test this operator. The operator was developed in Helm using the operator-SDK framework version: v1.7.2. It was tested using the KUBernetes Test Tool (KUTTL). Tests were provided as YAML files and Test assertions were mentioned as partial YAML documents. Additionally, a Go-based application was deployed to the cluster using the proposed custom resource, an example of which is shown in Figure 2.

3.2. Experimental results and analysis

Once the custom resource for an application is created, the operator deploys the application as a deployment in Kubernetes. It also ensures that the metrics are exposed for monitoring and alerting for unwanted behavior. These applications whose custom resources are deployed can be seen as a target in the Prometheus dashboard as shown in Figure 4. When the metrics exceed the defined threshold mentioned in the rules, an alert is triggered by the Alert Manager. The metrics can be visualized as Grafana dashboards. The status of the application and its metrics/traffic can be seen in Kiali as well as shown in Figure 5.

The operator replaces around 800 lines of code that each application needs with around 30-50 lines of code of the custom resource. This operator vastly improves productivity by automating the deployment of the application and monitoring for unwanted behavior of its metrics which is currently done by deploying each necessary resource manually. The operator watches over all the applications using its custom resource and any change to the custom resource is efficiently reflected in the Kubernetes resources deployed. When there is any modification, such as an API version upgradation of resources for all applications, it is very efficient to use our operator as the upgradation needs to be done only at one place in the operator code rather than at every resource of every application. The operator makes sure it gets reflected in all services without disturbing the monitoring and alerting system. This upgrade is graceful and occurs with negligible downtime. The operator is light and consumes fewer resources. Figure 6 and Figure 7 depict the memory usage and CPU usage, respectively, of the operator taken from Grafana. As we can see, it is highly scalable as there is only a gradual increase in resource utilization upon adding additional applications. This operator saves time and resources for the users by automating the generation, monitoring, and visualization of applications with minimal resource usage.



Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://10.6.188.90:8080/prometheus/metrics	UP	endpoint="http" instance="10.6.188.90:8080" job="prefix-helloapp-svc" namespace="core" pod="prefix-helloapp-5f04fc59b-xj85" service="prefix-helloapp-svc"	5.144s ago	2.49ms	

Figure 4. Target created in Prometheus for a sample application

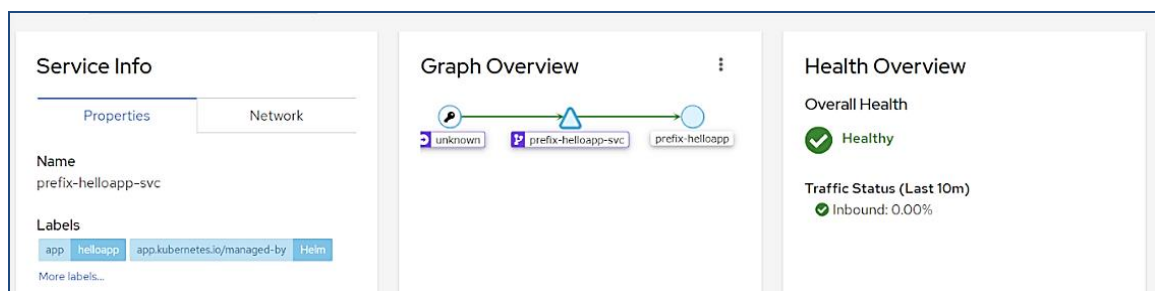


Figure 5. Status of the application in Kiali

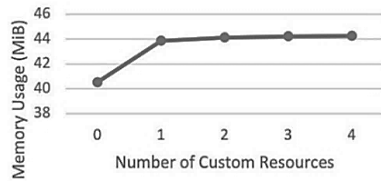


Figure 6. Memory usage of Wxcc operator for different number of custom resources

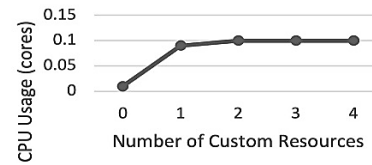


Figure 7. CPU usage of Wxcc operator for different number of custom resources

4. CONCLUSION

Monitoring applications in the Kubernetes cluster is important as it gives the users better visibility into the application's performance. Alerting and visualization enable developers to ensure high availability by working on the application and fixing it on time. In this paper, we have developed a Kubernetes operator that automates deployment, visualization, and monitoring for services that are deployed on the cluster and vastly improves productivity. The operator enables the users to maintain standard dashboards and alerting rules, if necessary. Since all the resources are deployed and controlled by the operator, the API version upgrade is now simplified. As a part of future work, the operator can be modified to deploy more resources. Visualization of metrics can be enhanced for different types of metrics.

ACKNOWLEDGEMENTS

We wish to acknowledge Cisco Systems Inc., Mr. Rajiv Shankar Daulath, Mr. Ravi Kiran Vadlamani, Mr. Sundaram Ravi, and other colleagues from Cisco Systems Inc. for supporting us throughout the project. Their guidance and encouragement are much appreciated.





REFERENCES

- [1] J. Shah and D. Dubaria, "Building modern clouds: Using docker, kubernetes & Google cloud platform," in *2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC)*, 2019, pp. 0184-0189, doi: 10.1109/CCWC.2019.8666479.
- [2] L. A. 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.
- [3] N. S. M. Pakhrudin, M. Kassim, and A. Idris, "A review on orchestration distributed systems for IoT smart services in fog computing," *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 11, no. 2 pp. 1812-1822, Apr. 2021, doi: 10.11591/ijece.v11i2.pp1812-1822.
- [4] C.-C. Chang, S.-R. Yang, E.-H. Yeh, P. Lin, and J.-Y. Jeng, "A kubernetes-based monitoring platform for dynamic cloud resource provisioning," *GLOBECOM 2017-2017 IEEE Global Comm. Conf.*, 2017, pp. 1-6, doi: 10.1109/GLOCOM.2017.8254046.
- [5] R. Buyya, M. A. Rodriguez, A. N. Toosi, and J. Park, "Cost-efficient orchestration of containers in clouds: A vision, architectural elements, and future directions," *Journal of Physics: Conference Series*, vol. 1108, p. 012001, Nov. 2018, doi: 10.1088/1742-6596/1108/1/012001.
- [6] W. Hassan, T.-S. Chou, X. Li, P. Appiah-Kubi, and T. Omar, "Latest trends, challenges and Solutions in Security in the era of Cloud Computing and Software Defined Networks," *International Journal of Informatics and Communication Technology (IJ-ICT)*, vol. 8, no. 7, pp. 162-183, Dec. 2019, doi: 10.11591/ijict.v8i3.pp162-183.
- [7] N. Sukhija and E. Bautista, "Towards a Framework for Monitoring and Analyzing High Performance Computing Environments Using Kubernetes and Prometheus," *2019 IEEE SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI*, 2019, pp. 257-262, doi: 10.1109/SmartWorld-UIC-ATC-SCALCOM-IOP-SCI.2019.00087.
- [8] N. Sukhija *et al.*, "Event management and monitoring framework for HPC environments using Service Now and Prometheus," in *Proceedings of the 12th International Conference on Management of Digital EcoSystems*, 2020, doi: 10.1145/3415958.3433046.
- [9] M. Brattstrom and P. Morreale, "Scalable Agentless Cloud Network Monitoring," *2017 IEEE 4th International Conference on Cyber Security and Cloud Computing (CSCloud)*, 2017, pp. 171-176, doi: 10.1109/CSCloud.2017.11.
- [10] D. Balla, C. Simon, and M. Maliosz, "Adaptive scaling of Kubernetes pods," in *NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium*, pp. 1-5, 2020, doi: 10.1109/NOMS47738.2020.9110428.
- [11] C. Cox, D. Sun, E. Tarn, A. Singh, R. Kelkar, and D. Goodwin, "Serverless inferencing on Kubernetes," *arXiv preprint arXiv:2007.07366*, 2020.
- [12] O. Arouk and N. Nikaiein, "Kube5G: A Cloud-Native 5G Service Platform," *GLOBECOM 2020 - 2020 IEEE Global Communications Conference*, 2020, pp. 1-6, doi: 10.1109/GLOBECOM42002.2020.9348073.
- [13] A. Mahajan and T. A. Benson, "Suture: Stitching safety onto kubernetes operators," in *Proceedings of the Student Workshop*, 2020, pp. 19-20, doi: 10.1145/3426746.3434055.
- [14] N. Zhou *et al.*, "Container orchestration on HPC systems through Kubernetes," *J Cloud Comp*, vol. 10, no. 16, pp. 1-14, Feb. 2021, doi: 10.1186/s13677-021-00231-z.
- [15] G. Carcassi, J. Breen, L. Bryant, R. W. Gardner, S. Mckee, and C. Weaver, "SLATE: Monitoring distributed kubernetes clusters," in *Practice and Experience in Advanced Research Computing*, 2020, pp. 19-25, doi: 10.1145/3311790.3401777.
- [16] C. Ariza-Porras, V. Kuznetsov, and F. Legger, "The CMS monitoring infrastructure and applications," *Computing and Software for Big Science*, vol. 5, no. 1, pp. 1-12, 2021, doi: 10.1007/s41781-020-00051-x.





- [17] W. Li, Y. Lemieux, J. Gao, Z. Zhao, and Y. Han, "Service Mesh: Challenges, State of the Art, and Future Research Opportunities," *2019 IEEE International Conference on Service-Oriented System Engineering (SOSE)*, 2019, pp. 122-1225, doi: 10.1109/SOSE.2019.00026.
- [18] L. Larsson, W. Tärneberg, C. Klein, E. Elmroth, and M. Kihl, "Impact of etcd deployment on Kubernetes, Istio, and application performance," *Journal of Software: Practice and Experience*, vol. 50, pp. 1986-2007, Aug. 2020, doi: 10.1002/spe.2885.
- [19] M. G. Xavier, M. V. Neves, F. D. Rossi, T. C. Ferreto, T. Lange, and C. A. F. De Rose, "Performance Evaluation of Container-Based Virtualization for High Performance Computing Environments," *2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, 2013, pp. 233-240, doi: 10.1109/PDP.2013.41.
- [20] D. Bernstein, "Containers and Cloud: From LXC to Docker to Kubernetes," in *IEEE Cloud Computing*, vol. 1, no. 3, pp. 81-84, Sept. 2014, doi: 10.1109/MCC.2014.51.
- [21] A. Putri, R. Munadi, and R. Negara, "Performance analysis of multi services on container Docker, LXC, and LXD," *Bulletin of Electrical Engineering and Informatics*, vol. 9, no. 5, pp. 2008-2011, Oct. 2020, doi: 10.11591/eei.v9i5.1953.
- [22] S. Chandran, S. Chandrasekar, and N. E. Elizabeth, "Konnect: An Internet of Things (IoT) based smart helmet for accident detection and notification," in *2016 IEEE Annual India Conference (INDICON)*, 2016, pp. 1-4, doi: 10.1109/INDICON.2016.7839052.
- [23] N. Divyasudha, P. Arulmozhivarman, and E. R. Rajkumar, "Analysis of Smart helmets and Designing an IoT based smart helmet: A cost effective solution for Riders," in *2019 1st International Conference on Innovations in Information and Communication Technology, ICICT 2019*, Chennai, India, 25-26 April, pp. 1-4, 2019, doi: 10.1109/ICICT1.2019.8741415.
- [24] P. Prasetyawan *et al.*, "A prototype of IoT-based smart system to support motorcyclists safety," *Journal of Physics Conference Series*, vol. 1810, no. 1, Mar. 2021, doi: 10.1088/1742-6596/1810/1/012005.
- [25] Z. Wei-guo, M. Xi-lin, and Z. Jin-Zhong, "Research on kubernetes' resource scheduling scheme," *Proceedings of the 8th Int. Conference on Communication and Network Security, ICCNS 2018*, 2018, pp. 144-148, doi: 10.1145/3290480.3290507.
- [26] N. Sabharwal and P. Pandey, "Container Reporting & Dashboards," in *Monitoring Microservices and Containerized Applications*, Berkeley, CA: Apress, 2020, pp. 169-182, doi: 10.1007/978-1-4842-6216-0_6.

BIOGRAPHIES OF AUTHORS







Prerana Shenoy S. P.     is a final year BE student of Computer Science and Engineering Department of RV College of Engineering. She is currently interning at Cisco Systems as a Technical Undergraduate Intern. Her areas of interest include Cloud Computing, Machine Learning, and Computer Vision. She can be contacted at email: preranashenoy29@gmail.com.







Soudri Sai Vishnu     is a final year BE student majoring in Electronics and communication engineering at RV college of engineering. Currently, he is interning with Cisco Systems as a Technical Undergraduate Intern. His areas of interest include Cloud computing, Application development, and Machine learning. He can be contacted at email: soudrisaivishnu@gmail.com.



Dr. Ramakanth Kumar P.     has received Ph.D. in Computer Science from Mangalore University, Karnataka, India. Currently, he is working as a Head of the Department (HoD) in the dept. of Computer Science and Engineering, RV College of Engineering, Bangalore, India. He has experience of 25 years in teaching and 14 years in R&D. His areas of interest include Digital Image Processing, Pattern Recognition, Natural Language processing. He can be contacted at email: ramakanthkp@rvce.edu.in.



Mrs. Sahana Bailuguttu     has received her M.Tech in Computer Network Engineering from Dayananda Sagar College of Engineering. Currently, she is working as Assistant Professor in the Department of Electronics and Communication, RV College of engineering. She has 14 years of teaching experience. Her areas of interest include Communication Systems and Networking. She can be contacted at email: sahanab@rvce.edu.in.