DSIM

Demystifying Kubernetes Service Types: ClusterIP, NodePort , LoadBalancer & ExternalName

Himanshu Tiwari[1],
[1]International Center for AI and Cyber Security Research and Innovations (CCRI), Asia University, Taiwan (e-mail: nomails1337@gmail.com).

ABSTRACT A variety of service types are available from Kubernetes, the container orchestration enabler, to manage networking and communications within clusters. The primary service types offered by Kubernetes—ClusterIP, NodePort, LoadBalancer, and ExternalName—will be the subject of this essay. We shall examine their special traits, practical uses, and workings. This thorough manual will assist you in comprehending the nuances and making knowledgeable networking decisions in your cluster environment, whether you are seeking for internal access, external visibility, or complex routing.

KEYWORDS Kubernetes, NodePort, ClusterIP, LoadBalancer, ExternalName

I. INTRODUCTION

Kubernetes[1] is a strong and cutting-edge platform that stands out in the constantly changing world of container orchestration. In essence, Kubernetes provides a wide range of services that are essential for promoting networking and communication within a cluster. In this in-depth post, we set out on a journey through the complexities of Kubernetes services, examining their features, functionalities, and use cases in detail.

Modern software deployment has been revolutionized by Kubernetes, which enables programmers to easily manage containerized applications in scalable, dynamic settings. However, maximizing its potential necessitates a deep comprehension of its service portfolio. We'll explore Kubernetes services like ClusterIP [2], NodePort [3], LoadBalancer [4], ExternalName [5] , and even the Ingress [6] universe in this exploration.

We try to demystify these service types with the use of vivid images and thorough explanations, giving you the information and insights required to make wise choices about networking and communication in Kubernetes clusters. This article gives you the knowledge you need to successfully navigate the Kubernetes ecosystem [7] , whether you're looking for internal or external access, load balancing capabilities, or complex routing solutions. Come along on this adventure with us as we explore the inner workings of Kubernetes services[8] and equip you with the knowledge you need to master container orchestration like a pro.

II. MATERIAL AND METHODS

The article combines research, real-world experience, and illustrative images to provide a thorough overview of Kubernetes services and their functionality. The following supplies and techniques were used to create this helpful article:

1. **Literature Review**: In-depth investigation was done to learn more about Kubernetes services, their varieties, and best practices. For this, it was necessary to research official Kubernetes documentation, online guides, articles, and pertinent literature.
2. **Practical Experience**: Insights from actual Kubernetes usage are incorporated into the text, ensuring that the knowledge is not simply theoretical but also useful in practical situations. The author uses their own knowledge gained from working with Kubernetes clusters.
3. **Graphic Illustrations**: The article has diagrams and illustrations to help with comprehension. The concepts and procedures related to Kubernetes services are illustrated in these graphics, which were produced using graphic design tools.
4. **Example YAML Manifests**: To demonstrate how to configure and deploy these services, sample YAML manifests for each type of Kubernetes service were created. To help readers implement Kubernetes services, these examples are provided in the article.

III. PROCESS AND EXPLAINATION KUBERNETES

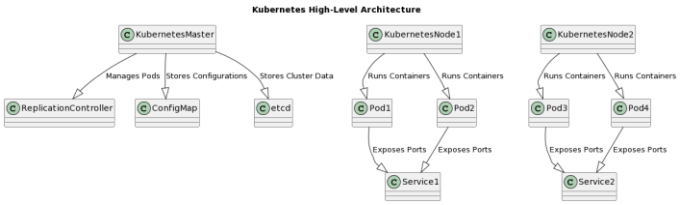

Fig 1. Kubernetes High-Level Architecture

Explanation:

a. **KubernetesMaster:** Kubernetes' control plane is responsible for managing the cluster as a whole and storing cluster data in a distributed database named etcd, as well as managing various resources such as replication controllers and config maps.
b. **KubernetesNode1 and KubernetesNode2:** These are the worker nodes within the cluster. They manage the containers (presented as Pods) and interact with KubernetesMatter.
c. **Pod1, Pod2, Pod3, and Pod4:** Pods are Kubernetes' smallest deployable units, each consisting of one or more containers that execute applications.
d. **Service1 and Service2:** Kubernetes services are utilized to expose and load balancing traffic to a cluster of Pods.
e. **ReplicationController [9]:** Ensures that there are a certain number of pod replica instances running at any given time. If a Pod replica fails, it will be replaced.
f. **ConfigMap [10]:** Organizes configuration information into key-value pairs that can be processed by Pods.
g. **etcd:** Distributed key-value store (DKV) is used to store cluster configuration and state data.

How It Works:

a. **Desired State Declaration:** YAML [11] (or JSON) files are used by users to specify the desired status of their applications. These files specify the number of Pod instances, their associated configurations, services and more.
b. **API Server:** These configuration files are sent by the users to the Kubernetes application programming interface server, which serves as the point of entry for cluster operations.
c. **etcd:** The API server maintains the configuration information in the file etcd, guaranteeing that the desired state of the cluster is always present and uniform.
d. **Scheduler:** The Scheduler component monitors Pods that have not yet been assigned a node and chooses the appropriate node based on criteria such as resource requirements and restrictions.
e. **Kubelet:** It runs on each node and verifies that the Pods containers are running and functioning as expected.
f. **Service Proxy:** Manages the transmission of network data to the pods. When a user connects to a service, it directs the data to a Pod that is connected to the Service.
g. **Controllers:** Controllers such as ReplicationController continually check the cluster's status and make changes to guarantee that the current state corresponds to the one specified by the user.
h. **User Interface:** The Kubernetes platform offers a user interface to facilitate the monitoring and administration of applications running in a cluster.

IV. PROCESS AND EXPLAINATION OF CLUSTER IP SERVICE

- ClusterIP is the default Kubernetes service type.
- This process generates a Service resource that has an IP address that originates from the cluster pool.
- ClusterIP services can only be accessed within the cluster or via kube-proxy.
- Ideal for internal application access within the cluster.

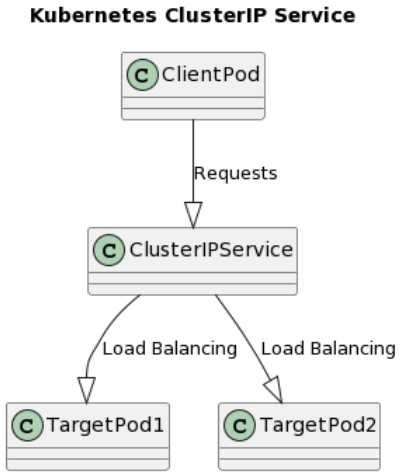

Fig2. Process of Kubernetes ClusterIP Service

Explanation:

a. **ClientPod [12]:** This represents a Kubernetes Pod requesting access to a service offered by other Kubernetes Pods.
b. **ClusterIPService [13]:** The ClusterIP Service is responsible for connecting the client Pod to the target Pod. The client Pod has an internal cluster IP address.

c. **TargetPod1 and TargetPod2:** These are the Pod(s) that offer the real-world service the client Pod desires to access. These Pod(s) may be reproductions of a single application.

How It Works:

a. **Client Request:** The ClientPod triggers a request for access to a service that is provided by the destination Pods. It queries the ClusterIP Service DNS name for access to the service.
b. **Load Balancing:** The ClusterIP Service performs a load balancing function. Upon receipt of a request from a client Pod, it forwards the request to one of the target Pods available. Load balancing facilitates the distribution of traffic between the target Pods.
c. **Accessing the Service:** The request is processed and the service is delivered by the target Pod that receives it. The ClusterIP Service subsequently relays the response to the client Pod.
d. **Internal Communication:** The Kubernetes cluster serves as the hub for all communication between the client pod and the target Pods as well as inside the target Pods. For accessing the service, the ClusterIP Service offers a reliable internal IP address.
e. **DNS Resolution:** Clients connect to the ClusterIP Service using its DNS name. Clients can find services using their DNS names thanks to Kubernetes' provision of DNS resolution within the cluster.

Example YAML Manifest:

```yaml
apiVersion: v1
kind: Service
metadata:
  name: "nginx-service"
  namespace: "default"
spec:
  ports:
    - port: 80
  type: ClusterIP
  selector:
    app: "nginx"
```

V. PROCESS AND EXPLAINATION OF NODEPORT SERVICE

- NodePort [14] directs traffic to a pod by opening a TCP port on each worker node.
- Although it creates an external access point to services, it depends on worker nodes being reachable.
- Externally accessible within a VPC or, if available, through a public IP.

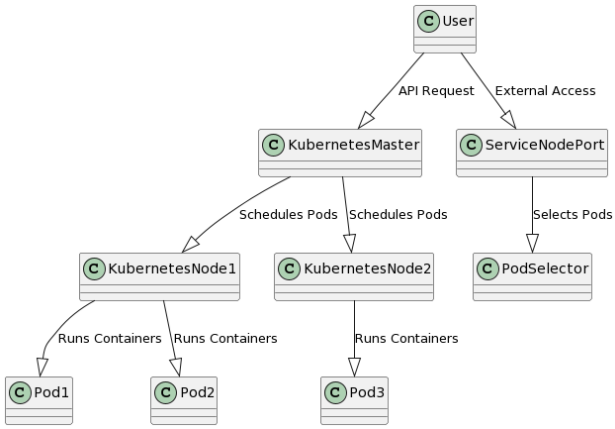

Fig 3. Process of NodePort Service in Kubernetes

Explanation:

a. **User:** Access requests from external users or services to a Kubernetes cluster's running applications.
b. **KubernetesMaster [15]:** The Kubernetes control plane, which looks after the cluster.
c. **KubernetesNode1 and KubernetesNode2:** the cluster's worker nodes, on which the application Pods are scheduled and executed.
d. **Pod1, Pod2, and Pod3:** pods that are operating containers of the application that must be made publicly accessible.
e. **Service (NodePort):** The Pods were exposed through the NodePort Service. On each node, it opens a static port that may be utilised for external access.
f. **PodSelector [16]:** Based on labels and selectors, the Service utilises a pod selector to decide which Pods to transmit traffic to.

How It Works:

a. **User Access Request:** Requests to reach the application running in the Kubernetes cluster are sent by users or outside services.
b. **Kubernetes API [17] Request:** The Kubernetes API server, which serves as the hub for all cluster activities, responds to user requests first.
c. **Pod Scheduling:** The application pods (Pod1, Pod2, and Pod3) are scheduled by KubernetesMaster onto the available worker nodes (KubernetesNode1 and

KubernetesNode2). The containers for the application are operated by these Pods.

d. **NodePort Service Creation:** In the Kubernetes manifest, the user defines a NodePort Service resource with a specified port number. Based on labels and selectors, this Service chooses the Pods to which traffic should be sent.
e. **Node Port Assignment:** Each node in the cluster has its assigned port opened by the NodePort Service (for example, port 30001). These ports are constant and static between nodes.
f. **Traffic Forwarding:** The traffic is routed by the Service to one of the chosen Pods (e.g., Pod1 or Pod2) whenever an external user or service reaches one of the nodes in the cluster on the designated NodePort (e.g., Node1:30001).
g. **Load Balancing:** The NodePort Service carries out fundamental load balancing by dividing up incoming traffic among the chosen Pods. This makes sure that external users may access any of the application's running instances, regardless of how many there may be.
h. **Application Response:** The chosen Pod responds to the user or external service after processing the incoming request.

Example YAML Manifest:

```yaml
apiVersion: v1
kind: Service
metadata:
  name: "nginx-service"
  namespace: "default"
spec:
  ports:
    - port: 80
      nodePort: 30001
  type: NodePort
  selector:
    app: "nginx"
```

VI. PROCESS AND EXPLAINATION OF LOADBALANCER SERVICE

- A load balancer created by LoadBalancer, such as the AWS Classic Load Balancer, is external.
- It generates NodePort and ClusterIP services automatically.
- Applied to direct traffic from the external load balancer to the cluster's pods.
- Appropriate for external service exposure thanks to features like SSL/TLS termination.

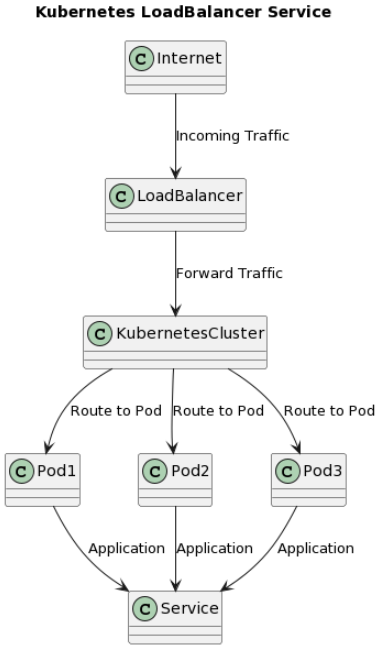

Fig.4 Process of Kubernetes LoadBalancer Service

Explanation:

a. **Internet:** represents clients or external users attempting to access your Kubernetes cluster-based application.
b. **LoadBalancer:** This is an outside load balancer that the cloud provider offers (for example, AWS ELB or GCP Load Balancer). It directs incoming traffic to the Kubernetes cluster by sitting in between the internet and the cluster.
c. **KubernetesCluster:** Numerous nodes (worker nodes) in the Kubernetes cluster serve as pod hosts. Your application's deployment and scalability will be managed by the cluster.
d. **Pod1, Pod2, Pod3:** Your application is executing on these pods. In Kubernetes, pods are the smallest deployable units and often include one or more containers.
e. **Service:** A logical set of pods and an access policy are defined by the Kubernetes LoadBalancer Service. It distributes requests to one of the pods in the set and serves as a reliable endpoint for outside traffic.

How It Works:

a. **Service Definition:** In your Kubernetes manifest file, include a LoadBalancer Service and a selection to designate which pods should accept traffic.

b. **Kubernetes API:** The Kubernetes API server receives the service configuration.
c. **Service Controller:** To deploy an external load balancer, the Kubernetes Service Controller processes the request and talks with the cloud provider's API (such as AWS or GCP).
d. **Load Balancer Creation:** AWS Elastic Load Balancer (ELB) or Google Cloud Load Balancer are two examples of external load balancers that the cloud provider can construct depending on the stated Service.
e. **External IP Assignment:** An external IP address, such as a public IP address, is given to the external load balancer.
f. **Traffic Routing:** The load balancer's external IP address receives incoming traffic from the internet.
g. **Load Balancer Rules:** Based on load balancing rules (such as round-robin, least connections, etc.), the load balancer sends traffic to one of the pods.
h. **Pod Routing:** The chosen pod that is running the application receives traffic.
i. **Pod Response:** The request is processed by the programme within the pod, which then replies.
j. **Scaling:** The LoadBalancer Service automatically adjusts the routing configuration to accommodate the new pods as the cluster scales by adding or deleting pods.

Example YAML Manifest:

```yaml
apiVersion: v1
kind: Service
metadata:
  name: "nginx-service"
  namespace: "default"
spec:
  ports:
    - port: 80
  type: LoadBalancer
  selector:
    app: "nginx"
```

VII. PROCESS AND EXPLAINATION OF EXTERNALNAME SERVICE

Requests are redirected to an external domain defined in its configuration by ExternalName, which functions as a DNS proxy.

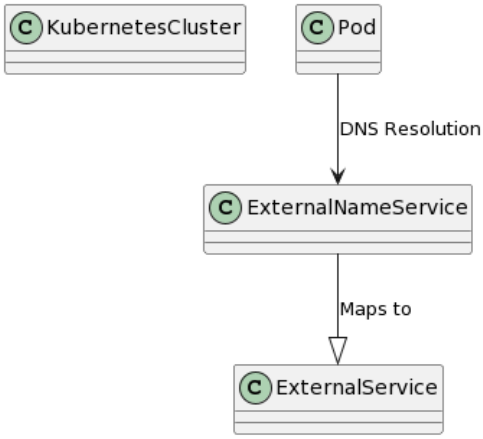

Fig.5 Process of ExternalName Service

Explanation:

a. **Kubernetes Cluster:** The whole Kubernetes cluster on which your application and services are operating is represented by this object.
b. **ExternalName Service:** This is a Kubernetes service of the type "ExternalName." Between an internal DNS name and an exterior DNS name or IP address, it serves as a mapping.
c. **Pod:** Your application containers are housed on pods, which are the smallest deployable units in Kubernetes.
d. **External Service:** This indicates an external service that is not a part of the Kubernetes cluster, such as a database or an API.

How It Works:

a. **Configuration:** A Kubernetes manifest (YAML) file is used to define an ExternalName Service and construct it. You provide the DNS name (or IP address) of the external service you wish to access in the externalName field of the YAML file when you define the type as ExternalName.
b. **Internal DNS Resolution:** The Kubernetes cluster's pods desire access to the outside service. They make use of the ExternalName Service's DNS name.
c. **DNS Resolution:** The Kubernetes DNS service (CoreDNS) intercepts requests sent by pods to the DNS name connected to the ExternalName Service.
d. **Mapping:** CoreDNS locates the associated external DNS name (or IP address) by checking the DNS name in the ExternalName Service settings.

e. **External Access:** CoreDNS essentially reroutes the request to the external service by returning the external DNS name (or IP address) to the Pod.
f. **Pod Communication:** Using the DNS name supplied by the ExternalName Service, the Pod may now connect with the external service.

Use Cases:

- **Database Connections:** To link your application pods to a database located outside the cluster, utilise an ExternalName Service.
- **Third-Party APIs:** You may use an ExternalName Service to give a nice DNS name for any third-party APIs or services that your application needs to interact with.
- **Load Balancer Integration:** Use ExternalName Services to direct traffic from your cluster to external services if you have an external load balancer or ingress controller.

Example YAML Manifest:

```yaml
apiVersion: v1
kind: Service
metadata:
  name: "google-service"
  namespace: "default"
spec:
  ports:
    - port: 80
  type: ExternalName
  externalName: google.com
```

VIII. PROCESS AND EXPLAINATION OF INGRESS NOT A SERVICE

Ingress (Not a Service):

- Ingress sets routing rules for an Ingress Controller but is not a service type.
- It allows for sophisticated traffic management by supporting path-based or host-based routing.
- requires an ingress controller, such as the ALB ingress controller for AWS, in order to operate.

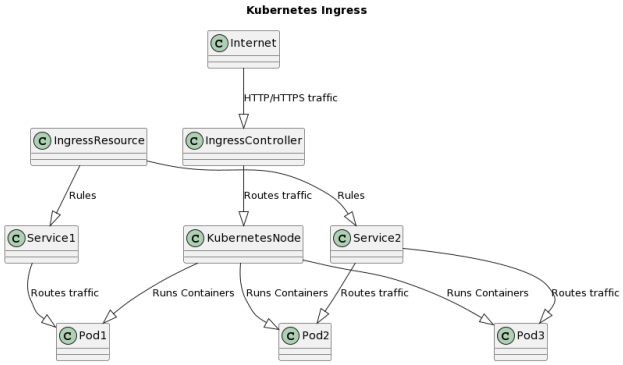

Fig 6. Process of Kubernetes Ingress

Explanation:

a. **Internet:** represents external customers making HTTP or HTTPS requests to services in the Kubernetes cluster.
b. **IngressController:** This is a piece of software that the cluster has installed. It keeps an eye out for Ingress resources and controls how external traffic is routed to internal services according to the guidelines set out in the Ingress resource.
c. **KubernetesNode:** workers in the cluster that house containers running Pods.
d. **IngressResource:** To specify how traffic should be sent to cluster services, the user defines an ingress resource. Rules based on hostnames and paths are specified.
e. **Service1 and Service2:** Services for Kubernetes that expose and control internal Pods. Services work with Ingress rules to direct traffic from outside sources to the right Pods.
f. **Pod1, Pod2, and Pod3:** Pods running containerized applications.

How It Works:

a. **Ingress Resource Creation:** Ingress resources are created by users who include routing rules based on hostnames and paths. For instance, they may specify that queries to example.com/app1 and example.com/app2 should go to different services.
b. **Ingress Controller:** The ingress controller (such as Nginx's or ALB's) keeps an eye out for changes to ingress resources. The controller examines the rules and configurations associated with a newly created or changed Ingress resource.
c. **Routing Logic:** The ingress controller configures routing logic based on the rules of

the ingress resource. It chooses which services and Pods inside the cluster should get which requests.

d. **External Access:** The external IP address or domain name of the cluster receives HTTP or HTTPS requests from external clients (such as web browsers). These queries are received by the Ingress controller, which is accessible on HTTP port 80 and HTTPS port 443.
e. **Traffic Routing:** The hostnames and pathways of the incoming requests are examined by the Ingress controller. To direct traffic to the proper Kubernetes Services, it makes use of the stated rules.
f. **Service Load Balancing:** Requests are load balanced amongst Pods by the Services. As an illustration, if Service1 receives a request, it sends it to one of the Pods executing the Service1-related application.
g. **Pod Execution:** The selected Pod executes the requested application logic and returns a response.

IX. CONCLUSION

In conclusion, this post presents a thorough review of several Kubernetes service types, highlighting their unique qualities, use cases, and operational mechanics. These service types include ClusterIP, NodePort, LoadBalancer, ExternalName, and even touches on Ingress. These are the main conclusions:

1. **ClusterIP Service**: The default Kubernetes service type, ClusterIP, gives cluster-wide apps internal access. It is perfect for intra-cluster communication since it provides a reliable internal IP address for gaining access to services.
2. **NodePort Service**: Every worker node has a static port opened by NodePort, allowing access to services from the outside world. Although it offers external exposure, it is mainly utilised within a Virtual Private Cloud (VPC) or over a public IP and depends on the accessibility of worker nodes.
3. **LoadBalancer Service**: A reliable method for external service exposure, LoadBalancer builds an external load balancer supplied by the cloud provider. It is appropriate for cases that demand for capabilities like SSL/TLS termination and automatically establishes NodePort and ClusterIP services.
4. **ExternalName Service**: When configured, ExternalName serves as a DNS proxy, rerouting queries to a given external domain or IP address. It is helpful for connecting to external services from within the Kubernetes cluster, such as databases or third-party APIs.
5. **Ingress**: Ingress sets routing rules for an Ingress Controller, allowing for sophisticated traffic control based on hostnames and pathways, while not being a service type. To function properly, it needs an ingress controller, such as ALB's ingress controller for AWS.

In order to assist readers understand how each service type functions inside a Kubernetes cluster, the article also includes comprehensive graphics and descriptions for each service type. This article is a useful tool for making knowledgeable networking decisions in Kubernetes settings, whether you require internal or external access, load balancing, or sophisticated routing capabilities.

VI. REFERENCES

1. Bernstein, D. (2014, September). Containers and Cloud: From LXC to Docker to Kubernetes. *IEEE Cloud Computing*, *1*(3), 81–84. https://doi.org/10.1109/mcc.2014.51
2. Pecka, N., Ben Othmane, L., & Valani, A. (2022, May 19). Privilege Escalation Attack Scenarios on the DevOps Pipeline Within a Kubernetes Environment. *Proceedings of the International Conference on Software and System Processes and International Conference on Global Software Engineering*. https://doi.org/10.1145/3529320.3529325
3. Larsson, L., Tärneberg, W., Klein, C., Elmroth, E., & Kihl, M. (2020, August 7). Impact of etcd deployment on Kubernetes, Istio, and application performance. *Software: Practice and Experience*, *50*(10), 1986–2007. https://doi.org/10.1002/spe.2885
4. Takahashi, K., Aida, K., Tanjo, T., & Sun, J. (2018, January 28). A Portable Load

Balancer for Kubernetes Cluster. *Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region*. https://doi.org/10.1145/3149457.3149473
5. Erdenebat, B., Bud, B., & Kozsik, T. (2023). Challenges in service discovery for microservices deployed in a Kubernetes cluster – a case stud. *Infocommunications Journal*, *15*(Special Issue), 69–75. https://doi.org/10.36244/icj.2023.5.11
6. Abdollahi Vayghan, L., Saied, M. A., Toeroe, M., & Khendek, F. (2018, July). Deploying Microservice Based Applications with Kubernetes: Experiments and Lessons Learned. *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*. https://doi.org/10.1109/cloud.2018.00148
7. Alawneh, M., & Abbadi, I. M. (2022, December 12). Expanding DevSecOps Practices and Clarifying the Concepts within Kubernetes Ecosystem. *2022 Ninth International Conference on Software Defined Systems (SDS)*. https://doi.org/10.1109/sds57574.2022.10062874
8. Pereira Ferreira, A., & Sinnott, R. (2019, December). A Performance Evaluation of Containers Running on Managed Kubernetes Services. *2019 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. https://doi.org/10.1109/cloudcom.2019.00038
9. Vohra, D. (2016, January 1). *Creating a Multi-Container Pod*. https://doi.org/10.1007/978-1-4842-1907-2_13
10. *Enabling HPC Workloads on Cloud Infrastructure Using Kubernetes Container Orchestration Mechanisms*. (2019, November 1). IEEE Conference Publication | IEEE Xplore. https://doi.org/10.1109/CANOPIE-HPC49598.2019.00007
11. Bartlett, J. (2022, December 17). *Basic Kubernetes Management*. Apress eBooks. https://doi.org/10.1007/978-1-4842-8876-4_9
12. *Pods-as-Volumes | Proceedings of the Seventh International Workshop on Container Technologies and Container Clouds*. (n.d.). ACM Conferences. https://doi.org/10.1145/3493649.3493653
13. Vayghan, L. A. (2019, January 15). *Kubernetes as an Availability Manager for Microservice Applications*. arXiv.org. https://doi.org/10.48550/arXiv.1901.04946
14. N. Nguyen and T. Kim, "Toward Highly Scalable Load Balancing in Kubernetes Clusters," in *IEEE Communications Magazine*, vol. 58, no. 7, pp. 78-83, July 2020, doi: 10.1109/MCOM.001.1900660.
15. *VPKIaaS | Proceedings of the 11th ACM Conference on Security & Privacy in Wireless and Mobile Networks*. (n.d.). ACM Conferences. https://doi.org/10.1145/3212480.3226100
16. Qadir, S. M. N. (2023). Kubernetes Network Policies and Security Implication Basic Concepts and Configuration Guidance. *SSRN Electronic Journal*. https://doi.org/10.2139/ssrn.4536996
17. Yilmaz, O. (2021). Extending the Kubernetes API. *Extending Kubernetes*, 99–141. https://doi.org/10.1007/978-1-4842-7095-0_4
18. Wang, L., Li, L., Li, J., Li, J., Gupta, B. B., & Liu, X. (2018). Compressive sensing of medical images with confidentially homomorphic aggregations. IEEE Internet of Things Journal, 6(2), 1402-1409.
19. Gupta, S., Gupta, B. B., & Chaudhary, P. (2018). Hunting for DOM-Based XSS vulnerabilities in mobile cloud-based online social network. Future Generation Computer Systems, 79, 319-336.
20. Bhushan, K., & Gupta, B. B. (2017). Security challenges in cloud computing: state-of-art. International Journal of Big Data Intelligence, 4(2), 81-107.
21. Plageras, A. P., Stergiou, C., Kokkonis, G., Psannis, K. E., Ishibashi, Y., Kim, B. G., & Gupta, B. B. (2017, July). Efficient large-scale medical data (ehealth big data) analytics in internet of things. In 2017 IEEE 19th Conference on Business informatics (CBI) (Vol. 2, pp. 21-27). IEEE.