Deployment of Applications Using Nginx Ingress Controller

Gautham S[1], Maddula Abhijit[2], Prof. Sahana .B[3]

[1,2]Student, Dept. of Electronics and Communication, R.V. College of Engineering, Bangalore, INDIA -560102

[3]Assistant Professor, Dept. of Electronics and Communication, R.V. College of Engineering, Bangalore, INDIA -560102

[1]gauthams.ec17@rvce.edu.in, [2]mabhijit.ec17@rvce.edu.in, [3]sahanab@rvce.edu.in

Abstract: *Cloud computing is a method of storing and manipulating data by utilizing a network of remote servers. Cloud computing is becoming increasingly popular owing to its large storage capacity, ease of access, and wide range of services.Virtualization entered the picture when cloud computing progressed, and technologies or software such as virtual machines emerged. However, when customers' computational needs for storage and servers rose, virtual machines were unable to meet those expectations owing to scalability and resource allocation limitations. As a result, containerization came into picture.Containerization refers to the packaging of software code together with all of its necessary elements such as frameworks, libraries, and other dependencies such that they are isolated or segregated in their own container. Kubernetes used as an orchestration tool implements ingress controller to route external traffic to deployments running on pods via ingress resource. This enables effective traffic management among the running applications avoiding unwanted blackouts in the production environment.*

Keywords: **Virtualization, Kubernetes, Ingress Controller, Containerization.**

1. INTRODUCTION

Cloud Computing is getting popular these days as a most essential computing model in which computing resources are made available on-demand to the user as needed. Cloud computing is a powerful computing model in which a network of remote servers are used to store the data and tasks are assigned to a combination of connections, software and services accessed over a network. The vast processing power of Cloud Computing is made possible through distributed, large-scale computing clusters, often in concert with server virtualization software, like VMware ESX Server, Citrix Xen Servers, and parallel processing. This network of servers and connections is altogether known as the Cloud. Other key points in cloud computing lies in its properties like Reliability, Scalability and Elasticity of resources which makes cloud computing as a best computing model. Anyone on the internet can use the cloud to store their data in the cloud and use its vast resources. Cloud computing also offers a variety of service models.

In Cloud Computing, terms like Virtualization and Containerization play a major role in provisioning physical resources, such as processors, disk storage, and broadband networks. Virtualization is nothing but creation of a virtual version of an OS, a server, a storage device or network resources. Or simply abstraction of physical resources for users. In the Cloud, these physical resources are regarded as a pool of resources, these resources thus can be allocated on demand. Containerization, on the other hand, has emerged as a significant theme in software development as an alternative to or complement to virtualization. It entails encapsulating or packing software code and all of its dependencies in order for it to operate reliably and universally on every infrastructure. Virtual Machine is a Virtual Software Computer that is like a physical computer runs an OS and application. Containers have a logical packaging framework from which programs or applications can be abstracted from the environment in which they currently run. This decoupling makes it possible to run container-based applications quickly and reliably, regardless of whether the target setting is a private data center, the public cloud, or even a developer's personal laptop.

2. LITERATURE SURVEY

The paper[1] presented a way of improving the scheduling algorithm of the container by integrating grey system theory with the Long Short-Term Memory (LSTM) neural network prediction approach by assessing the overall design and scheduling strategy of Kubernetes. Based on the results of the experiments, the paper concludes that the method may decrease the fragmentation problem of resources of the cluster's working nodes and boost the use of cluster resources.

The paper [2] contrasts specification characteristics of MANO with the core container orchestration techniques of Kubernetes. It assesses the extent to which CNFs (Containerized Network Functions) can be handled using the Kubernetes platform. To support the analysis, a set of tests on stress and chaos were conducted on the Kubernetes testbed using an example virtual IP Multimedia System.

The paper [3] discusses the design and deployment details for a data analysis pipeline for Single Particle Imaging (SPI) studies in the Kubernetes system. The paper also examines various software consumption patterns for distinct payloads. Traditional HPC programs, applications requiring GPU calculations, GUI-based software, and software that may be naturally parallelized by separating the data into numerous separate portions are all pipeline software components (Data parallelism). Individual deployment strategies provided for each payload type are also discussed in the paper.

The paper referenced [4] showcases that with the growing use of containerized virtualization in data centers and clouds, providing high-throughput and zero-copy data transmission between such containers is critical. RDMA (Remote Direct Memory Access) allows you to bypass the kernel for packet processing by offloading it to RDMA-enabled network interface cards (NICs). Paper also explains that the existing solutions for RDMA with containers either relied on proprietary container orchestrators or do not allow the control plane to manage the underlying RDMA traffic. Paper builds on prior Kubernetes work to provide an architecture that offers control over RDMA bandwidth needs within a Kubernetes cluster.

The paper referenced [5] explains the increase in the number of applications in the world and the need to compete with such a large volume of apps, a strong, easy-to-scale, and adaptable application is essential to stay in the market and outperform competitors. Microservices architecture and continuous deployment strategies address the aforementioned difficulties and help enterprises enhance their productivity. Paper delves into the various methods for deploying micro services in a CI/CD pipeline utilizing Rundeck, Docker containers, Docker-Compose, and Kubernetes. Furthermore, it analyses the benefits and downsides of different orchestration approaches, describing appropriate use cases for each one. It finishes by stating that Kubernetes is the most effective approach to deploy micro services for high availability and scalability.

3. BRIEF OUTLINE

The required dependencies are installed on the host machine with pre-calculated hardware and software specifications of Intel® Core™ i7 processor, 16 GB RAM, minimum disk space of 500 GB SSD, and Windows 10 operating system. The tools and the required software are updated to the latest available configuration to enhance security and functionality. The intended web applications to be deployed are divided into several sub-applications with respect to its Business units. A three-tier architecture which consists of the frontend, application and backend layers has been adopted. Application specific images have been built either by using Docker Desktop or AWS instances, with the supporting libraries, files, and configurations. These generated custom images are used as templates to create runtime instances called containers which require minimum CPU and RAM to run the application as intended. The application is deployed on a Kubernetes cluster consisting of two nodes controlled by a master node running on an AWS Ubuntu Bionic 18.04 instance. The traffic is effectively managed by the Ingress Controller by Nginx installed on the cluster. The client requests for a particular sub-application are directed to the necessary nodes via the Ingress Resource. The corresponding deployments are created with respect to the images and exposed to a Node IP or Cluster IP service accordingly. The number of

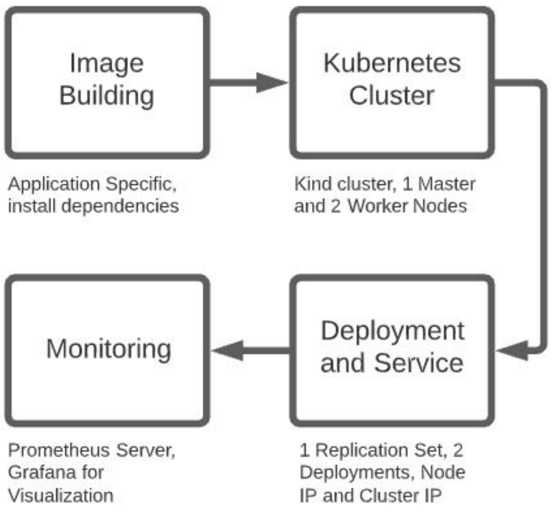


Image Building
Application Specific, install dependencies
Kubernetes Cluster
Kind cluster, 1 Master and 2 Worker Nodes
Monitoring
Prometheus Server, Grafana for Visualization
Deployment and Service
1 Replication Set, 2 Deployments, Node IP and Cluster IP

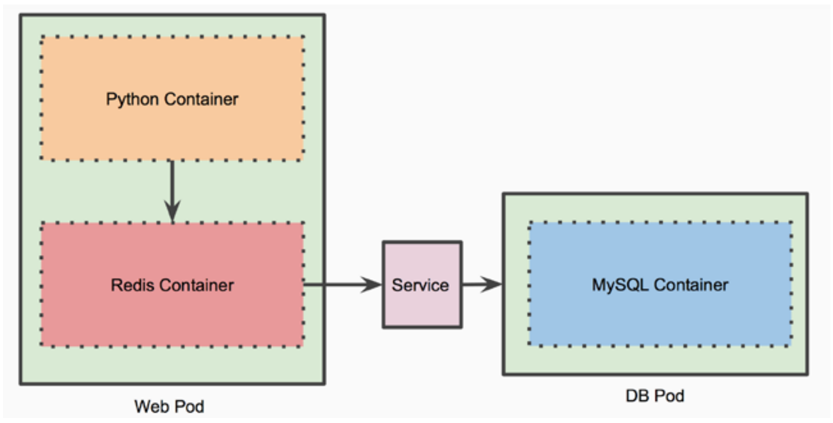
replications of the pods are managed by the replication controller, which is by default set to one.

The replications can be scaled up or scaled down depending on the incoming traffic monitored by the master node. The deployments are exposed to a Node IP service with port 8080 to make the deployment accessible for the external client. Each node is equipped with a frontend and a backend pod which communicate with each other using the Cluster IP service and ensure the application runs smoothly. Finally, unit and integration tests are performed on the developed cluster to ensure the system satisfies the requirements and ensures the application adheres to the six-sigma rule. The cluster is monitored continuously to check for possible complications using third party tools such as Prometheus Server and Grafana for visualization.

4. IMPLEMENTATION

YAML Ain't Markup Language is a data serialization language that meets the data requirements of the user. It is intended to be user-friendly and to operate well with other programming languages. It is useful for data management and has Unicode readable characters. Configuration files for pods are written using YAML language.

YAML uses the concepts of lists and dictionaries similar to the Python programming language. The YAML code is started with three dashes ('-') and a detailed specification is given at the beginning like version of API, Object, Kind and Meta-data related to the application.



The YAML codes for the web-pod and db-pod are developed and run on the worker node-1. Pods are able to internally communicate with each other using a cluster IP service. And the user is able to interact with these pods externally using node IP service. The web-pod employs the python API to retrieve and insert user-data into the database. The Redis container acts as a cache to temporarily hold data retrieved from db-pod. And db-pod holds the user-data information.

The whole setup is accessible via the ingress controller with the paths specified in the ingress resource. The API file is divided into 4 main parts i.e., Authentication, Initialization, Adding and Retrieving users. The Flask module authenticates the entry into the database credentials of which have been provided in the db-pod.yml. Init module initializes the database by running a series of SQL queries provided via the API interface. Add-user module enables the addition of user-data into the initialized database. While the get-user module retrieves data from the database with the implementation of cache to retrieve frequently accessed data. The python 3.9 and HTML dependencies are installed to make the HTML files and API files executable on the pods. All the Front-end files are copied from the local directory into the image directory i.e., “/usr/share/nginx/html". The Docker file is built and the image is created and pushed into the Docker hub repository from where the image is pulled to create the deployment for the database application in the created kubernetes cluster.

The HTML and css files are copied from the local directory into the image directory i.e., “/usr/share/nginx/html".This is achieved by developing a Docker file that specifies the installation of Nginx: Alpine dependencies to run the associated front-end application. The built image is again uploaded onto the Docker-hub repository and pulled to create the deployment for the Hospital reservation application in the created kubernetes cluster. A config file is written and used to create a Kubernetes cluster, which will use a pre-built node image to bootstrap a Kubernetes cluster. The command kind create cluster —config config would create a cluster with one master and two workers using the setup shown in figure1. The pods are scheduled on the nodes that are managed by the master node and the cluster would be named kind by default. The command “k get no” displays a list of the cluster's nodes and their associated Node IPs.

Once the Kubernetes cluster is deployed, the Nginx controller is installed, and the traffic is routed via the ingress resource depending on the client request. Two applications are deployed under the resource specified by a path in the host file, each running on separate nodes. To achieve this, a Kubernetes Deployment configuration must be created which tells Kubernetes how to generate and update application instances. The Kubernetes control plane schedules the application instances contained in a Deployment to run on particular Nodes in the cluster after it is created. Figure 5.2 depicts the development of deployments from the yaml files required for the Database application, as well as a list of the cluster's running pods.

The pods are exposed to cluster IP and node IP services to enable external and internal communication. A Kubernetes Service is an abstraction that provides a logical grouping of Pods that all serve the same functionality and run anywhere in the cluster. Each Service is given its own IP address when it is created (also called clusterIP). This address is bound to the Service's lifecycle and will not change while it is active. Pods can be configured to communicate with the Service, with the assurance that communication with the Service will be load-balanced to a pod that is a member of the Service. Postman is a tool for testing APIs and Curl is a command-line utility that allows users to transport data via URLs. Figure 6 shows how to access the application by first initializing the database, then constructing the user data payload by providing a user ID, a user name, a NodeIP, and a NodePort. The command curl http://<NodeIP>:<NodePort>/user/user Id> can be used to view database data.

When the database application is accessed via the web browser, the User interface could be observed by adding users to the database. These user additions would trigger an API response which further reacts with the web and db pods to perform the required backend operations for successful registry of the user. The second application, hospital reservation system is also being accessed from the same cluster running on a different node. The command “curl NodeIP: Node Port number enables us to access the application by returning the HTML content of the home page of the web application. When the web application is accessed via a web browser, the actual visual of the home page could be observed.

5. CONCLUSION

Containers are lightweight, scalable and isolated VMs in which applications are run. Container Orchestration is referred to as automated arrangement, coordination, and management of software containers. Kubernetes is the solution to most of our problems regarding the platform for automating deployment, operations of application containers, scaling, and it also provides for the container centric infrastructure.

The project implements AWS as a cloud service. Kubernetes cluster is created as per our need.

After its creation, pods are deployed to them. In order to communicate between the pods services are created. The software used is based on Nginx + Python. HTML based application is created. One more application is created for the database. All our data that is entered is stored here. The advantage of storing databases in the cloud is, the data is portable across all environments. The data is secure and we need not worry about the security. This application can be modified and be operated by millions of users at the same time. With Kubernetes load balancing does not remain a major concern. Also it becomes easy to start, stop and update the application as and when required. The advantage of using cloud computing is that there are infinite sources which can be accessed. With Container Orchestration the scaling or removal or addition of containers has become easier. By building our application, blackouts can be reduced. It will also follow the 6 sigma rule i.e., 99.999999 percent which is the active time for the application. This percentage of active hours means in a year the application will only be unavailable for 10 minutes. Our application logic has a Dynamic Website Host on a web server. It uses an API backend micro service deployed as a container.

6. FUTURE SCOPE

Kubernetes is the building block of the IT infrastructure. For a lot of years the customers have used bare-metal physical servers, private and public cloud and virtual machines. Some even used multiple public clouds to meet the required specifications. With the advancement in technology the demand is for hybrid environments or rather for immaterializing the IT environment. Now each developer wants consistent ways for building and managing apps irrespective of the footprint they are on. While creating big applications, it cannot be built every time before running so in that case web applications such as Jenkins are to be used.

ACKNOWLEDGEMENT

We would like to express our gratitude to our guide Prof. Sahana B for guiding us in each step of project.

REFERENCES

1. Y. Yang and L. Chen, “Design of kubernetes scheduling strategy based on lstm and grey model,” in 2019 IEEE 14th International Conference on Intelligent Systems and Knowledge Engineering (ISKE), IEEE, 2019, pp. 701–707
2. M. Gawel and K. Zielinski, “Analysis and evaluation of kubernetes based nfv management and orchestration,” in 2019 IEEE 12th International Conference on Cloud Computing (CLOUD), IEEE, 2019, pp. 511–513
3. A. Tesliuk, S. Bobkov, V. Ilyin, A. Novikov, A. Poyda, and V. Velikhov, “Kubernetes container orchestration as a framework for flexible and effective scientific data analysis,” in 2019 Ivannikov Ispras Open Conference (ISPRAS), IEEE, 2019, pp. 67–71.
4. C. Link, J. Sarran, G. Grigoryan, M. Kwon, M. M. Rafique, and W. R. Carithers, “Container orchestration by kubernetes for rdma networking,” in 2019 IEEE 27th International Conference on Network Protocols (ICNP), IEEE, 2019, pp. 1–2.
5. H. Rajavaram, V. Rajula, and B. Thangaraju, “Automation of microservices application deployment made easy by rundeck and kubernetes,” in 2019 IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT), IEEE, 2019, pp. 1–3.