Published by :
http://www.ijert.org

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 09, September-2020

# Object Storage and Kubernetes Containerization of Object storage Application

Sonali Prakash Jadhav
M.Tech, Computer Science & Engineering,
Vishwakarma Institute of Technology,
Pune, India

***Abstract:*** **These days, developers are called on to write applications that run across multiple operating environments, including dedicated on-prem servers, virtualized private clouds, and public clouds such as AWS and Azure. Traditionally, applications and the tooling that support them have been closely tied to the underlying infrastructure, so it was costly to use other deployment models despite their potential advantages. This meant that applications became dependent on a particular environment in several respects, including performance issues related to a specific network architecture adherence to cloud provider-specific constructs, such as proprietary orchestration techniques and dependencies on a particular back-end storage system. Containers have provided us with more flexibility for running cloud-native applications on physical and virtual infrastructure. Containers package up the services comprising an application and make them portable across different compute environments, for both development/test and production use.**

**Keywords: Containerization, Dockerizing object storage application, Dockerfile to create docker image, *Docker Compose to* run multiple containers as a single service.**

## 1. INTRODUCTION

This thesis researches container technologies using Docker and Kubernetes. The main objective is to create a Dockerfile, which forms the base image for the deployment. The image is then used to deploy to Kubernetes. The idea came from the application development team with a need to make the deployment process more streamline for environment setup and testing purposes. Although automation is not included in this thesis, the basis is made from which the creation of the automated deployment pipeline can be started. The goal of this thesis is to find a quick and efficient way to deploy all components and new versions of the application in a test and potentially a production environment. The research for this thesis conducted from a practical viewpoint.

As most modern software developers can attest, containers have provided us with more flexibility for running cloud-native applications on physical and virtual infrastructure. Containers package up the services comprising an application and make them portable across different compute environments, for both development/test and production use. And because containers draw on resources of the host OS, they are much lighter weight than virtual machines. This means containers make highly efficient use of the underlying server infrastructure.

PaaS tries to get around these issues, but often at the cost of imposing strict requirements in areas like programming languages and application frameworks. Thus, PaaS is off limits to many development teams.

Kubernetes eliminates infrastructure lock-in by providing core capabilities for containers without imposing restrictions. It achieves this through a combination of features within the Kubernetes platform, including Pods and Services. The reason Kubernetes is chosen instead of the native Docker cluster, Docker Swarm, is its scalability, portability and self-healing attributes. Kubernetes has been around longer than Docker Swarm and therefore has much more documentation. It also has more widespread 3 rd. party applications support available.

### *A. Context*

Cloud storage has changed the rules for deploying simpler, infinitely scalable and more affordable storage. So it makes little sense to burden a cloud storage platform with storage systems that are based on 20th century file systems that inhibit administration, scalability and cost.

Selecting the correct underlying storage system can greatly impact the success or failure of implementing cloud storage. The characteristics of object storage are ideally aligned with a cloud storage infrastructure, delivering a superior cloud storage experience with better scalability, accessibility and affordability.

Object storage is a much better fit for cloud infrastructures. Instead of using a complex, difficult to manage and antiquated file system, object storage systems leverage a single flat address space that enables the automatic routing of data to the right storage systems, specifies the content lifecycle and keeps both active and archive data in a single tier with the appropriate protection levels. This allows object storage to provide better value by aligning the value of data and the cost of storing it without requiring oppressive management overhead to manually move data to the proper tier while providing infinite scalability to support the capacity-on-demand capability of cloud storage. Object storage is also designed to run at peak efficiency on commodity server hardware.

With Object Storage, you can safely and securely store or retrieve data directly from the internet or from within the cloud platform. Object Storage offers multiple management interfaces that let you easily manage storage at scale. The elasticity of the platform lets you start small and scale seamlessly, without experiencing any degradation in performance or service reliability. Object Storage is a regional service and is not tied to any specific compute instance. You

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 09, September-2020

can access data from anywhere inside or outside the context of Cloud Infrastructure, as long you have internet connectivity.
Kubernetes is a vendor-agnostic cluster and container management tool, open-sourced by Google in 2014. It provides a "platform for automating deployment, scaling, and operations of application containers across clusters of hosts". Above all, this lowers the cost of cloud computing expenses and simplifies operations and architecture.
Kubernetes is a cluster and container management tool. It lets you deploy containers to clusters, meaning a network of virtual machines. The basic idea of Kubernetes is to further abstract machines, storage, and networks away from their physical implementation. So it is a single interface to deploy containers to all kinds of clouds, virtual machines, and physical machines.
Provisioning, in this context, means a process of preparing, equipping and making available a platform for the Object Storage software components to function in a conducive environment, ensuring reliable functionality.
Kubernetes marks a breakthrough for DevOps because it allows teams to keep pace with the requirements of modern software development. In the absence of Kubernetes, teams have often been forced to script their own software deployment, scaling, and update workflows. Kubernetes allows us to derive maximum utility from containers and build cloud-native applications that can run anywhere, independent of cloud-specific requirements. This is clearly the efficient model for application development and operations we've been waiting for.

### *B. Background*

In the pre-Kubernetes era, infrastructure and app development were inescapably intertwined. As complexities grew and teams evolved, we saw "DevOps" emerge as a bridge between development and operations in an attempt to resolve the age-old delivery trouble arising from developers throwing things over the wall to ops and then ops having to deal with production issues on the other side. DevOps rose to be a new subculture within existing teams, sometimes yielding new "DevOps teams" and even leading to a new class of tools and methodologies.

#### *1) Traditional Deployment Era*

Early on, organizations ran applications on physical servers. There was no way to define resource boundaries for applications in a physical server, and this caused resource allocation issues. For example, if multiple applications run on a physical server, there can be instances where one application would take up most of the resources, and as a result, the other applications would underperform. A solution for this would be to run each application on a different physical server. But this did not scale as resources were underutilized, and it was expensive for organizations to maintain many physical servers.

#### *2) Virtualized Deployment Era*

As a solution, virtualization was introduced. It allows you to run multiple Virtual Machines (VMs) on a single physical server's CPU. Virtualization allows applications to be isolated between VMs and provides a level of security as the information of one application cannot be freely accessed by another application.
Virtualization allows better utilization of resources in a physical server and allows better scalability because an application can be added or updated easily, reduces hardware costs, and much more.
Each VM is a full machine running all the components, including its own operating system, on top of the virtualized hardware.

#### *3) Container Deployment Era*

Containers are similar to VMs, but they have relaxed isolation properties to share the Operating System (OS) among the applications. Therefore, containers are considered lightweight. Similar to a VM, a container has its own file system, CPU, memory, process space, and more. As they are decoupled from the underlying infrastructure, they are portable across clouds and OS distributions.

### *C. Working of container orchestration*

When you use a container orchestration tool, like Kubernetes or Docker Swarm (more on these shortly), you typically describe the configuration of your application in a YAML or JSON file, depending on the orchestration tool. These configurations files (for example, docker-compose.yml) are where you tell the orchestration tool where to gather container images (for example, from Docker Hub), how to establish networking between containers, how to mount storage volumes, and where to store logs for that container. Typically, teams will branch and versions control these configuration files so they can deploy the same applications across different development and testing environments before deploying them to production clusters.
Containers are deployed onto hosts, usually in replicated groups. When it's time to deploy a new container into a cluster, the container orchestration tool schedules the deployment and looks for the most appropriate host to place the container based on predefined constraints (for example, CPU or memory availability). You can even place containers according to labels or metadata or according to their proximity in relation to other hosts—all kinds of constraints can be used.
Once the container is running on the host, the orchestration tool manages its lifecycle according to the specifications you laid out in the container's definition file (for example, it's Dockerfile).
The beauty of container orchestration tools is that you can use them in any environment in which you can run containers.
Compose is a tool for defining and running multi-container Docker applications. With Compose, you use a YAML file to configure your application's services. Then, with a single command, you create and start all the services from your configuration.
Compose works in all environments: production, staging, development, testing, as well as CI workflows.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 09, September-2020

## 2. KUBERNETES AND DOCKER SWARM

| Features | Kubernetes | Docker Swarm |
|---|---|---|
| Installation & Cluster Configuration | Installation is complicated; but once setup, the cluster is very strong | Installation is very simple; but cluster is not very strong |
| GUI | GUI is the Kubernetes Dashboard | There is no GUI |
| Scalability | Highly scalable & scales fast | Highly scalable & scales 5x faster than Kubernetes |
| Auto-Scaling | Kubernetes can do auto-scaling | Docker Swarm cannot do auto-scaling |
| Load Balancing | Manual intervention needed for load balancing traffic between different containers in different Pods | Docker Swarm does auto load balancing of traffic between containers in the cluster |
| Rolling Updates & Rollbacks | Can deploy Rolling updates & does automatic Rollbacks | Can deploy Rolling updates, but not automatic Rollbacks |
| Data Volumes | Can share storage volumes only with other containers in same Pod | Can share storage volumes with any other container |
| Logging & Monitoring | In-built tools for logging & monitoring | 3rd party tools like ELK should be used for logging & monitoring |

## 3. APPROACH USED

Instead of using Docker as a standalone process, they can be combined with a virtual machine. All
Hypervisors are a good platform for the Docker host: Virtual Box, Hyper-V, AWS EC2 Instance. No matter the hypervisor, Docker will perform well. Sometimes a virtual machine might be the place to run the docker container , but you don't necessarily need to. The container can be run as a stand-alone service on top of bare metal During the early years of virtual machines, they gained popularity for their ability to enable higher levels of server utilization, which is still true today. By mixing and combining Docker hosts with regular virtual machines, system administrators can maximize efficiency from their physical hardware. Building a container cluster on top of a virtual machine, whether it was made with Docker Swarm or Kubernetes, enables the usage of all the resources provided by the physical machine to maximize performance.

To build an image from an application a Dockerfile is needed. The purpose of the Dockerfile is to automate the image building process, in which all the necessary dependencies and libraries are installed. This project required the configuring of a multi-stage Dockerfile. This enables defining15 multiple build stages in the same file. It makes the build process more efficient through reducing the size of the final image. In the first phase all the needed dependencies are installed and compressed into an artifact. An artifact in Linux environments is usually a compressed .tar.gz- file.

In the second phase the artifact is taken and extracted. The result is a ready Docker image, which can be further used in a deployment.

Although premade images are available for use from the Docker Hub, it is sometimes better to make a specific Dockerfile. This way, you know how the final image is built, what licenses and/or properties it contains. The downside of this is that the responsibility to keep the Dockerfile updated falls on the organization. It is also possible to combine prepared images with self-made Dockerfiles to maximize efficiency.
The Dockerfile supports 13 different commands, which tell the Dockerfile how to build the image and how to run it inside a container. There are two phases in the building process: *Build* and *Run.* In the BUILD -phase you determine the commands which are executed during the build process. In the RUN- phase the commands specified are run when the container is run from the image. The two commands WORKDIR and USER can be used by both phases.

## 4. SYSTEM DESIGN

The two main components used in this thesis are Docker and Kubernetes. Docker is used to create a Docker image of the application by using a Dockerfile. A Dockerfile has all the instructions on how to build the final image for deployment and distribution. The images that are made are reusable perpetually. The image is then used by Kubernetes for the deployment. The benefits of Docker are, for example, the reusability of once created resources and the fast setup of the target environment, whether it is for testing or production purposes. This is achieved through container technologies made possible by Docker and Kubernetes. Container technology is a quite new technology which has been growing for the past five years.

Once the Docker image is created with the Docker platform, it is ready to be used with the Kubernetes container platform. With the Docker platform a base image is created, which is then used by the Kubernetes deployment platform. At best this is done with a press of a button. The ease-of-deployment eliminates possible human errors in the process, which makes the deployment reliable, efficient and quick. The reason Kubernetes was selected is its versatility, scalability and the potential automatization of the deployment process. The technologies are still quite new and are being developed every day to improve the end-user experience, which already is enjoyable.

The field of Development and Operations (DevOps) benefit greatly from containerization in the form of automating the deployment. There are several types of software to create a continuous integration and deployment pipeline (CI/CD). This enables the DevOps team to deploy an application seamlessly to the targeted environment. Compared to normal virtual machines, containerized platforms require fewer configurations and can be deployed quickly with the CI/CD 6pipeline. Container technologies also solve the problem of software environment mismatches because all the needed dependencies are installed inside the container and they do

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 09, September-2020

not communicate with the outer world. This way the container is isolated and has everything it needs to run the application. With containers, all the possible mismatches between different software versions and operating systems are canceled out. It enables the developers to use whichever programming language and software tool they want to, if they can run it without problems inside the container. This combined with the deployment process, makes the whole ordeal agile, highly scalable and most importantly, fast. With Docker and Kubernetes, it is possible to create a continuous integration- and deploymentpipeline which for example guarantees a quickly deployed development version of an application to test locally.

## 5. DESIGN OF PROPOSE SOLUTION

To build an image from an application a Dockerfile is needed. The purpose of the Dockerfile is to automate the image building process, in which all the necessary dependencies and libraries are installed. This project required the configuring of a multi-stage Dockerfile.

In the first phase all the needed dependencies are installed and compressed into an artifact. An artifact in Linux environments is usually a compressed .tar.gz- file.

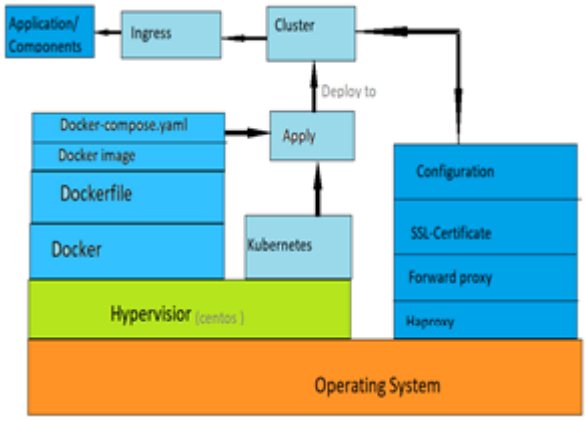


Fig.1 Workflow and description of resources used to achieve the result

In the second phase the artifact is taken and extracted. The result is a ready Docker image, which can be further used in a deployment.

Although premade images are available for use from the Docker Hub (https://hub.docker.com/), it is sometimes better to make a specific Dockerfile. This way, you know how the final image is built, what licenses and/or properties it contains. The downside of this is that the responsibility to keep the Dockerfile updated falls on the organization. It is also possible to combine prepared images with self-made Dockerfiles to maximize efficiency.

## ACKNOWLEDGMENT

I would like to thank A A Bhilare, Assistant Professor of Department of Computer Science and Engineering for his support and encouragement at the various stages of this Dissertation work.

## REFERENCES

[1] Anthony T. Velte, T. J. “Cloud Computing a Practical Approach,” TATA McGraw HILL Edition. 2010
[2] Architecture of the Kernel-based Virtual Machine (KVM). Retrieved from: http://www.linux-kongress.org/2010/slides/KVM-Architecture-LK2010.pdf and http://it20.info/2007/06/a-brief-architecture-overview-of-vmware-esx-xenand-ms-viridian/2006-2010
[3] Deborah, D. and Giacomo, “Evaluating Cloud Computing: How It Differs To Traditional IT Outsourcing,” Master Thesis in Information Technology and Management. Jonkoping International Business School, Jonkoping University.2010
[4] https://kubernetes.io/docs/reference/kubectl/docker-cli-to-kubectl/, 2015.
[5] https://www.bluematador.com/blog/running-haproxy-docker-containers-kubernetes
[6] https://hub.docker.com/r/million12/haproxy/dockerfile
[7] https://docs.docker.com/get-started/
[8] Gibson, J., Rondeau, R., Qing, T. “Benefits and Challenges of Three Cloud Computing Service Models,” Fourth International Conference on Computational Aspects of Social Networks (CASoN), pp. 198-205, 2012