SELF-HOSTED KUBERNETES: DEPLOYING DOCKER CONTAINERS LOCALLY WITH MINIKUBE

*

Ruchika Muddinagiri
Information Technology
Vidyalankar Institute of Technology
Mumbai, India
rrmuddinagiri@gmail.com

Shubham Ambavane
Information Technology
Vidyalankar Institute of Technology
Mumbai, India
shubhamdambavane@gmail.com

Simran Bayas
Computer Engineering
Rajiv Gandhi Institute of Technology
Mumbai, India
bayas.simran39@gmail.com

***Abstract*—Containerization is a cutting-edge DevOps technology which unifies the IT operations and Development domains. In recent times, virtualization using Virtual Machines has become an overkill for its large overhead on systems. As a lightweight alternative, containerization offers containers that constitute a package of an application along with all its dependencies that is required for it to execute. Containerization platforms help in building containers from images. Docker is a widely popular containerization platform. Containerization Orchestration tools manage these containers. Kubernetes is the front-runner of the emerging market of container orchestration tools. These software work together seamlessly in order to successfully implement containerization both locally and on the cloud. In this paper, we aim to deploy the container orchestration tool Kubernetes on a local system with a Docker sample container. The purpose of this is to ensure that all the configurations and management needed for a Docker container is set successfully on the local system before it is deployed onto the cloud or on the premise. The on-premise deployment use case is very important in domains such as finance and healthcare where organizations hesitate to upload confidential information on to the cloud for security reasons but still require scaling of their applications.**

***Index Terms*—Container, Containerization, Cloud, Docker, Kubernetes, MiniKube, Deployment**.

I. INTRODUCTION

A. What is Containerization?

In traditional software development, when an application created on a certain computing environment is run on another, errors or bugs occur during the deployment. Software develop-ers solve this problem using containerization. Containerization is the process of bundling an application together with all its related configuration files, libraries and dependencies required for it to run efficiently. [2] Such a bundle is called a container. It is a lightweight alternative to Virtual Machines.

Virtual Machine	Container
Virtual machine runs on hypervisor.	Docker Engine is used to create, run and deploy applications in containers.
Each virtual machine has one or more operating systems on which we run applications.	Containers are run on the host operating system and as well as on virtual machine's operating system.
Size of virtual machines are in several gigabytes.	Size of containers are in terms of megabytes.
Virtual machine uses more memory and it takes minutes to start.	Containers use less memory and start within 50 milliseconds.

Fig. 1. Difference between Virtual Machine and Container.

B. What is a Container?

A container packages an application's components as building blocks of the complete application. [1]A Container consists of an entire runtime environment: an application plus all its dependencies, libraries and other binaries and configuration files needed to run it, bundled into one package. Thus, con-tainers are isolated from the host system that they run on and also from other containers. Containers offer an environment as close as possible to a VM but without the overhead of running a separate operating system and simulating all the hardware. Different containers share the operating system of the host machine with other containers. It runs as a discrete process, making it lightweight. There are different container platform providers like Docker, Portainer, AWS ECR, Marathon, etc. By far, the most popular one is Docker.In the image below, the architecture of a container has been demonstrated.

ISBN -978-1-7281-1901-4

Authorized licensed use limited to: Cornell University Library. Downloaded on August 26,2020 at 09:44:23 UTC from IEEE Xplore. Restrictions apply.

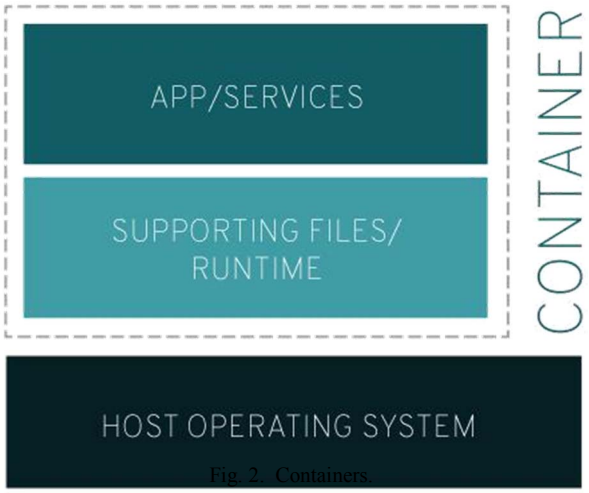

Fig. 2. Containers.

A container consists of the application and its supporting files (dependencies) within itself and it runs on a host operating system.

C. *Insight into the working of Docker*

Docker is a platform for developers to develop, deploy, and run applications within containers[1]. In Docker[2],

1) A container is launched by running an image. An image is an executable package that includes everything needed to run an application–the code, runtime, libraries, envi-ronment variables, and configuration files. In Docker, this image is called as DockerFile. A DockerFile is always built upon another base image.
   The Docker Hub is a public repository of various base docker images. We can also create our own pub-lic/private repositories on the Docker daemon.
2) Accessing resources like networking interfaces and disk drives is virtualized inside this environment, which is isolated from the rest of the system, post mapped to the outside world, and information about files that must be copied to that environment is provided in the DockerFile.
3) DockerFiles or images are used to "build" containers using the "docker build" command. A container is a runtime instance of an image–what the image is loaded in memory when executed.
4) The build of the application defined in the DockerFile behaves exactly the same wherever it runs. The Docker client communicates with the Docker daemon to create containers. All the images are stored either in the public or private registries. This is shown in the diagram below:

D. *What are Container Orchestration Tools?*

Docker is used for the creation of containers. However, we need tools that can help in the management of these containers. They are required because usually, a single application con-sists of many containers that work together. Thus, container orchestration includes the management of a container's life cycle, especially in a large and dynamic environment.

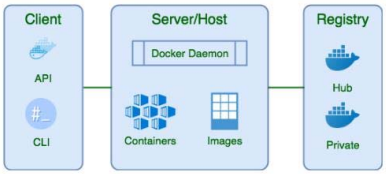

Fig. 3. Working of DockerFile.

There are various container orchestration tools available like Kubernetes, Docker Swarm, Apache Mesos, etc. They provide a framework for managing and integrating containers at scale. Containers are deployed on hosts, usually in replicated groups. Container orchestration tools schedule the deployment and the most appropriate host is found to place the container based on predefined constraints such as Computer Processing Unit, availability of memory, etc. While the container runs on the host, the container orchestration tool manages the life cycle of the container according to the specifications laid in the container's definition file, for example DockerFile. The container orchestration tools can be used in any environment which can run containers, from traditional on-premises servers to public cloud instances running on Amazon Web Services (AWS), Google Cloud Platform (GCP), Microsoft Azure, etc. Kubernetes is currently one of the leaders for the container orchestration tool.

E. *Why to choose Kubernetes for a containerization tool?*

Container orchestration tools have their pros and cons, Kubernetes, on the other hand, stands out from other con-tainerization tools based on factors such as architecture, HA needs, flexibility, and learning curve.

II. THE THEORY OF KUBERNETES

A. *What is Kubernetes?*

Real production applications span multiple containers. These containers have to be deployed across multiple server hosts. Kubernetes orchestration allows to build an application with services that span multiple containers and to schedule and manage these containers.[2] It automates Linux con-tainer operations thereby eliminating many of the manual processes involved. Kubernetes is popularly used with Docker containers. It is increasingly being adopted by enterprises,

ISBN -978-1-7281-1901-4

Authorized licensed use limited to: Cornell University Library. Downloaded on August 26,2020 at 09:44:23 UTC from IEEE Xplore. Restrictions apply.

governments, cloud providers and vendors due to its active community and feature set.[1]

B. Terms in Kubernetes:

In order to work with Kubernetes, we need to be acquainted with certain terms:[2]

1) Node: It is a worker machine.
2) Cluster: A group of nodes.
4) Pods: They are the atomic units which can contain one or more containers.
5) Deployments: They provide the most common way to create and manage pods.
6) Service: A service declares how pods can be accessed. It is the virtual server inside the node. When pods need to communicate, they use 'services'.

C. Ways to use Kubernetes

1) Hosted
Hosted Kubernetes refers to the usage of cloud ser-vices for working with Kubernetes. It only requires a subscription with a cloud provider (with Kubernetes services) to run and manage containerized applications. It is mainly used by enterprises or DevOps companies during production stages. It is the easier option to choose for scaling machines and to provide greater availability. Some popular hosted Kubernetes services are: Google's cloud Kubernetes Engine, Azure Kuber-netes Service, Amazon's Elastic Container Service for Kubernetes, IBM's Cloud Container Service, etc.

2) Self-Hosted
Self-hosted Kubernetes allows a user to run Kubernetes on a single server in order to test/deploy applications with it. It does this by automating steps used to deploy Kubernetes clusters. Kubeadm is the tool used in various distributions of Kubernetes to run locally on a single machine. It is basically used to test/deploy containerized applications with Kubernetes locally before putting it up for production. In Windows, the most popularly used Kubernetes software is MiniKube. MiniKube is essentially Kubernetes running very minimally on a local VM.

In this paper, we are focusing on how to create self-deployed solutions on local machines.

D. How to work with MiniKube

The local machine we worked on is a Windows 10 ma-chine. In order to work with Kubernetes,we need to have Docker Toolbox, kubectl, Minikube and Virtual Box.

1) Why Minikube: The main advantage of using MiniKube for Windows is that it supports several drivers including Virtual-Box and Hyper-V. We can use almost all the Kubernetes add-ons with MiniKube.
2) Why Docker-Toolbox: Minikube already comes with a docker daemon pre-installed, however it needs to communicate with a docker client. Docker ToolBox comes equipped with the docker client. The Docker client can then be used to connect to the docker daemon. Why Virtual Box: MiniKube creates a node as a virtual machine using Virtual-Box.
3) Why Virtual Box
MiniKube creates a node as a virtual machine using Virtual-Box.In Minikube, only a single node cluster is created. The virtual machine called boot2docker is created in VirtualBox using 2 GB RAM and 2 assigned CPUs. Boot2docker is a minimalist Linux distribution solely to run docker containers. MiniKube does notsupport multi-node clusters, however, for development purposes, a single node cluster is enough.

In Minikube, only a single node cluster is created. The virtual machine called boot2docker is created in VirtualBox using 2 GB RAM and 2 assigned CPUs. Boot2docker is a minimalist Linux distribution solely to run docker containers. MiniKube does not support multi-node clusters, however, for development purposes, a single node cluster is enough.

III. PRACTICAL IMPLEMENTATION

A. Flowchart of implementation

The flow of the implementation is shown in the form of the form chart shown below.

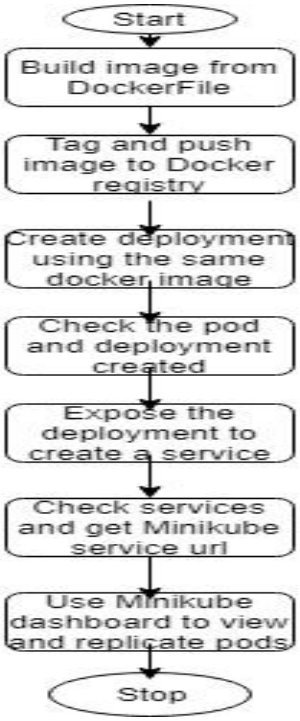

Fig. 4. Flowchart of Implementation.

B. Methodology of Implementation

In our paper, we focus on deploying/testing Kubernetes locally by deploying application containers on a local system

ISBN -978-1-7281-1901-4

Authorized licensed use limited to: Cornell University Library. Downloaded on August 26,2020 at 09:44:23 UTC from IEEE Xplore. Restrictions apply.

before it is deployed into the production stages. The benefit of doing this is that a local machine Kubernetes solution can help developers configure and run a Kubernetes cluster on the cloud.

For demonstration purposes, we need an application, which will be used to construct the Docker File from which the container will be built. In order to deploy/test Kubernetes locally, we consider a sample python application to set up a webserver on the local machine at a particular port. The python application uses the http.server and the socketserver packages to set up a webserver on the local machine at port number 8890. Thus, the webserver is made accessible from this port and it displays a certain predefined html page.

Once the application has been prepared, a DockerFile for the application is required. DockerFiles are used to build containers. Hence, a DockerFile is built upon a previously existing image of python:3-alpine.

The DockerFile which acts as the container image:

```
FROM python:3-alpine
ADD . /app
WORKDIR /app
CMD ["python", "server.py"]
```

Fig. 5. Docker File as Container Image.

Here, our DockerFile is built upon an existing image of the python:3-alpine image and it is stored in the ./app directory. The working directory is shifted to ./app and the server.py application is executed using the python server.py command. This is the code that will be deployed by minikube when it executes the dockerfile or container.

After creating the Docker, minikube and docker daemon are used to create and deploy the container. Starting minikube on command line and then starting the minikube-docker environment is done as follows:

```
minikube start
minikube docker-env
```

After starting the docker daemon, firstly, a private registry is required to allow only authorized users to use the registry. This private registry needs a docker login where official docker credentials are submitted. This authenticates any user attempt-ing to push or pull from the registry. This is similar to having our own private repository where confidential information can be uploaded from authorized individuals. Then, the container image is built from the DockerFile previously created. Tagging the image with the name of the repository and pushing it to the private registry is done using the following commands.

```
docker tag 127.0.0.1:5000 127.0.0.1:5000/pyweb
docker push 127.0.0.1:5000/pyweb
```

The image that is created an be viewed in the list of docker images using the following command.

```
docker images
```

After the container image has been successfully created and pushed to the registry, we need to deploy it in a pod with minikube. Follow these steps:

```
kubectl create deployment
    pyweb –image=127.0.0.1:5000/pyweb
    kubectl get pods
kubectl get deployments
```

After creating the deployment, expose it to port 8890 (where the python application is configured to open a web server) so that it can be accessed from outside the cluster.

```
    kubectl expose deployment pyweb –port=8890 –
    target-port=8890 –type=LoadBalancer
kubectl get services
minikube service pyweb –url
```

After creating the deployment, it is exposed to port 8890 (where the python application is configured to open a web server) so that it can be accessed from outside the cluster.

```
    kubectl expose deployment pyweb –port=8890 –target-
    port=8890 –type=LoadBalancer
kubectl get services
minikube service pyweb –url
```

The last command will return an ip-address. This ip-address is where the Kubernetes cluster has been set up by Minikube. Enter the address in the browser and see the service set up.

The dashboard provided by Minikube is also a great tool to view all the pods, deployments and services.

The MiniKube dashboard is a visual representation of all the Deployments, Pods and Replica Sets that have been created using MiniKube. It provides a lot of options to view the age of the pods, their replicas, etc.

The most important feature of the MiniKube dashboard is the option to scale the pods. Thus, we can create many different pods of the same container that has been deployed. This demonstrates the scaling feature of Kubernetes where immediate replication of the pods takes place. Kubernetes works wonders for scaling. This is one of the most important features that is being demonstrated in this paper.

This completes the demonstration of how an application is made, a DockerFile for that application is created, an image is built out of that dockerfile, how that image is deployed using minikube and how the deployment can be scaled.

IV. CONCLUSION AND APPLICATIONS

Thus, a docker container has been successfully deployed with Minikube locally. Minikube offers great command-line operations using kubectl that is configured to work with MiniKube. The in-built MiniKube Docker daemon is especially useful in creating Docker containers that are to be deployed within MiniKube. MiniKube also offers on-the-click scaling options in order to manage workloads. After testing whether the application container is running successfully with MiniKube, it can be easily deployed to the cloud for pro-

ISBN -978-1-7281-1901-4

Authorized licensed use limited to: Cornell University Library. Downloaded on August 26,2020 at 09:44:23 UTC from IEEE Xplore. Restrictions apply.

duction by IT companies. This helps in efficiently managing and configuring containers to work with Kubernetes. The applications of using MiniKube with Docker containers are many. As mentioned, it is especially useful in domains such as finance and healthcare where uploading sensitive customer and patient data to the cloud is a security risk. Servers can implement the applications in containers which can be then easily scaled to run for the entire organization using MiniKube. In addition, method of local testing before deploying to the cloud can be very useful to organizations.

REFERENCES

[1] Arsh Modak, Prof. S.D.Chaudhary, Prof. P.S.Paygude, Prof. S.R.Idate,"Techniques to Secure Data on Cloud: Docker Swarm or Kubernetes?"in 2nd International Conference on Inventive Communication and Computational Technologies., 2018 pp. 10-11.

[2] Jay Shah,Dushyant Dubaria,"Building Modern Clouds: Using Docker, Kubernetes Google Cloud Platform" presented at the IEEE conference, Jan 7-9, 2019 pp. 0184-0185-0187.

[3] Abhishek Jain.Docker Tutorial - How to create Docker Image for Python Application: In Initial Caps. (Release date). Accessed: August,2019.[Online Video]. Available:https://youtu.be/0F100Rd22HM

[4] https://docs.docker.com/

[5] https://kubernetes.io/docs/tutorials/hello-minikube/

[6] https://www.linuxnix.com/what-is-containerization-devops/targetText=Containerization

[7] https://blog.newrelic.com/engineering/container-orchestration-explained/

[8] https://www.backblaze.com/blog/vm-vs-containers/

[9] https://kubernetes.io/docs/tutorials/hello-minikube/

ISBN -978-1-7281-1901-4

Authorized licensed use limited to: Cornell University Library. Downloaded on August 26,2020 at 09:44:23 UTC from IEEE Xplore. Restrictions apply.