DAT515 Final Report : Deploying Stateless Apps with Docker, Kubernetes, and AWS EC2

Group05

Nourin Mohammad
Rihab Alzurkani
Shaima Ahmad Freja
nm.biswas@stud.uis.no
rh.al-zurkani@stud.uis.no
Sa.freja@stud.uis.no
University of Stavanger, Norway

ABSTRACT

Cloud Computing is quite a complex process, but the use of Docker and Kubernetes can significantly simplify it. A simple stateless web application was created to take in user contact information and store them in a database for storage and retrieval. This web application was created using PHP and MySQL to display the contact information form where user can input their Name , Phone Number and email and store it in the database for use later. Cloud Computing Technologies such as Docker for containerization , Kubernetes for orchestration and Amazon EC2 instance were used to deploy the web application. This project allowed us to delve deep into the workings and understanding of Docker, Kubernetes and AWS Cloud (EC2 Instance).

1 INTRODUCTION

Cloud computing is a technology paradigm that involves delivering various computing services and resources over the internet. In the past companies had to store all their data and software on physical hard drives and servers. As a company's business grows, so does their storage needs. Now in this modern age, cloud technology provides companies to scale, accelerate innovation, drive business agility and reduce costs.

In this project we dive deep into the world of cloud computing, with hands on experience with a range of tools and platforms that are at the forefront of this transformation. Using openstack which is an open source cloud computing infrastructure software project [1] , to the containerization evolution offered by Docker [9], the orchestration capabilities for containers at scale by Kubernetes [7] and last but not least the vast infrastructure as a Service (IaaS) offered by Amazon Web Service (AWS) [8].

Moreover, with the evolving changes in software developement such as adoption of microservices and serverless designs, cloud technology now offers built in support and effortless integration. This report will not only look into the traditional cloud infrastructure but also delve into cloud native API's, microservices , and serverless computing through AWS Lambda.

The project aims to achieve the following objectives:

(1) Develop a stateless web application using multiple containers.
(2) Gain practical expertise in Docker for containerization, Kubernetes for orchestration, and AWS EC2 instance deployment through AWS Cloud Services.
(3) Create the necessary Dockerfiles and Docker Compose configurations for application deployment.
(4) Configure Kubernetes services and deployment files for application deployment.
(5) Demonstrate an understanding of AWS EC2 instance and AWS Lambda functions by deploying the web application using Docker and Kubernetes on our AWS EC2 instance.
(6) Identify deployment challenges and develop strategies to overcome them.

2 APPLICATION SELECTION

We developed a simple stateless web application that was created using PHP and MySQL. The web application is a simple contact form that takes in the user contact information and saves it in the database. The information can then be retrieved from the database and displayed for the user to see as shown in Figure 1.

Figure 1: Contact Info Web Application

Lecturer: Prachi Vinod Wadatkar.

Final Report (DAT515), IDE, UiS
2023.

The following are the factors considered for choosing a simple stateless web application :

- Multi-container : The chosen application is simple, to make it suitable for containerization and deployment within the requirements.
- Stateless Nature: This application is stateless, to ensure compatibility with containerized environments. The reason belongs to the meaning that it depends on keeping the session state or persistent data on the local file system.
- Web-Based: This characteristic allows users to interact with it via a web browser.
- Technology Stack: The application should be constructed using technologies consistent with containerization, for example, PHP for server-side scripting.
- Github link : Group05-Contact Info PHP Web Application

3 CLOUD SETUP - OPENSTACK PLATFORM

In order to deploy our web application using Docker and Kubernetes, we need to work on a Cloud Server that provides us with a virtual servers and resources that can be used to install docker and kubernetes. For the purpose of our project, we used the Openstack Platform which is an open source platform using the Infrastructure as a Service (IaaS) in both the public and private cloud.

In order to setup the cloud, we are alloted an account per group from the university. Once logged, we created an instance with the required network, security and interface setting in order to access the Virtual Machine.

4 DOCKER CONTAINERIZATION

This section focuses on the Docker containerization phase [9]. As mentioned , our application is developed using PHP and MySQL. To deploy it using Docker, we need to create the necessary docker files that will build and run the images and then use docker-compose to create and run the containers. The docker containerization platform helps us to create, deploy, and run applications easily by using containers. In docker, multiple containers can run simultaneously, regardless of the images being the same or different. This containerization popularity has increased among developers and system administrators because it encompasses the applications complete file system with all its dependencies [4].

A docker image is a set of instructions that is used to build a docker container, it is essentially like a snapshot in a virtual environment. A docker image is pulled from the Docker hub and a docker container which is a runtime instance is created from that image.

The first step to using docker is to install it in our Ubuntu VM. We first connect to our Ubuntu Server and follow the step by step Docker Installation provided to us in the lab. Successful installation of docker will return the docker version when the command **sudo docker –version** is run. Once the installation is complete, we will then create the necessary docker files to deploy our web application.

With regards to our application, we have to pull two images to create two container for to deploy our web application. The following are the steps to build and run the image to create the containers :

- We create a docker file [5]in which we define the images we want to pull. In our case we will pull the PHP: 7.4 Apache. as well as install the mySQL extension required for the PHP. This will be copied in our ./var/www/html.
- Now to build the image from our docker file , we will navigate to the directory where our docker file is saved and run the following command : **"docker build -t dockerfile:latest .** This will run the dockerfile and create the two containers required for our application, PHP Containers and MySQL Container. Figure 2 show a snapshot of our docker as well as an image that provides a visual representation of how docker image and container is built for our application.
- Once the image is built, we will use docker compose to create our containers and deploy our web application. In order to ensure that we can display our web application on the browser we first have to do port forwarding and log into our UNIX server. This will ensure that there are no issues when trying to display our web page in the browser.
- Docker Compose : As our application is using multi containers , we will use docker compose which is a tool in docker that allows us to define and run multi-container Docker applications. The docker compose uses a YAML file to configure our application services.
- Docker Compose YAML File Configuration : We first declare the dependencies which will allow the MYSQL image and container to be built before the PHP container to ensure that the information stored is loaded. Then we decalare the two services that our application will be using respectively the PHP and MySQL.
- Figure 3 shows the docker compose file with the Services declared and the port mapping. We also mount the volume in the PHP container and we allocate around 1GB of storage to the containers. The volume is allocated incase the container fail or crashes unexpectedly, the data can be retrieved.The app-sql data volume is shared between the two containers. In order to run the docker compose file , we move into the directory where the file is saved and run the command **docker-compose up**.
- Once the docker compose command is run, it will first build the containers from the dockerfile (so we don't need to run the dockerfile seperately) then it will run container running the PHP file in the allocated port 8080. Our application was deployed successfully using docker and we were able to display it in the browser.

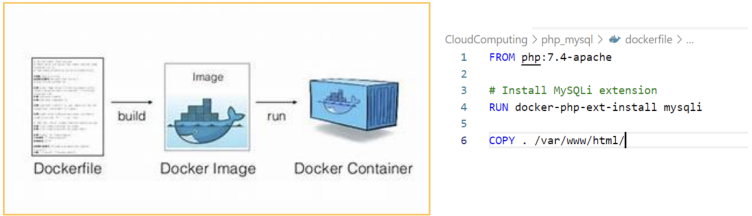

Figure 2: Caption

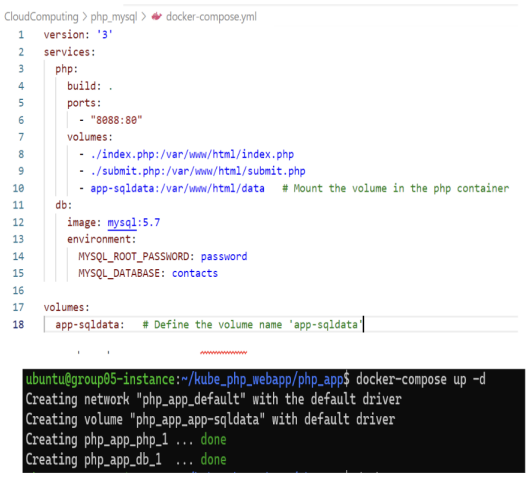

Figure 3: Caption

Advanced Features in Docker : Docker Health Checks Docker allows us to define healthchecks for our containers. Healthchecks can be used to determine the health status of a containerized application and trigger actions based on its health, such as automatic failover or scaling. We used this for our application to check if the containers are healthy or fail.

There are obviously many advantages of using docker to deploy applications, we list a few of them in regards to our web application

- **Isolation:** Docker containers encapsulate an application and its dependencies, ensuring isolation from the underlying system and other containers. This isolation prevents conflicts and ensures consistent behavior across different environments.
- **Portability:** Docker containers are highly portable. They can run consistently on any platform that supports Docker, be it a developer's laptop, a test server, or a production environment.
- **Efficiency:**Docker containers share the host operating system's kernel, which reduces overhead compared to traditional virtualization. Containers are lightweight and start quickly, making them efficient for resource utilization and scaling.
- **Security:** Docker provides security features like isolated name spaces and control groups, which help contain and manage potential security threats. Additionally, Docker Hub offers official images and scanning tools to ensure image integrity.

5 KUBERNETES ORCHESTRATION

Kubernetes is a robust open-source container orchestration platform known for its ability to automate the deployment, scaling, and management of vast containerized applications. It simplifies the deployment and scaling applications across diverse environments and offers a comprehensive feature set including auto scaling, self-healing, load balancing, and storage management.

Kubernetes comprises several components, including Pods, Deployments, and Services. A Pod is the smallest computing unit computing unit, consisting of one or more containers that share storage and network resources [2]. Deployments are responsible for creating and managing Pods while Services [3] responsible for exposing an interface to the created pods , enabling network access both within the cluster and externally, accessible from browsers.

These components, along with the control plane and nodes, work together to provide a robust platform for deploying and managing containerized applications.

As with docker, we also need to install Kubernetes in our Ubuntu Server. The installation of the Kubernetes is a bit complex but following the Kubernetes Installation Guide provided to us in the Lab, the installation was successful. To check if the kubernetes is installed correctly, we need to run **sudo kubectl version -client** and **sudo kubeadm version**. This will return the version of Kubernetes that is installed.

Once the Kubernetes is installed, we start the container, initialize the control plane and start a cluster. By default, Kubernetes Cluster will not schedule pods on the master/control-plane node for security reasons but because we are on a testing environment we need to schedule Pods on control-plane node to maximize resource usage.

To deploy our contact list application in Kubernetes, we followed these steps:

- **PHP App Image :** We began by pushing the Docker images for our PHP application to a Docker Hub repository. This step is crucial because it makes the Docker container image available to the Kubernetes cluster, allowing us to create pods from it.
- **Manifest Files:** Next, we prepared Kubernetes manifest files using YAML. These files configured the necessary components: Persistent Volume Claims (PVCs), Deployments, and Services.
- **Persistent Volume Claims (PVCs):** We created PVCs to mount a persistent volume into our pods. PVCs are used to ensure data persistence and can be shared across multiple pods.
- **Deployments:** Deployments are used to define and manage the application's lifecycle. They specify details like which container images to use, the replicaset, and the update strategy. Deployments automatically create and manage pods, replacing any failed pods with new ones.
- **Services:** We created a Service YAML file to define and manage services and to expose our PHP application. Services act as an interface to the pods, ensuring load balancing traffic across multiple Pods and enable network access both within the cluster and externally by Depending on the type of service **NodePort**.
- **Github link for Kubernetes Deployment and Services Files:** The following repository **Kubernetes Files** contains all the deployment and services files used to deploy our application in Kubernetes.

Figure 4 shows the php-deployment Yaml file, the Deployment is ensuring that there are al-ways two replicated Pods running the php-app application, and it uses the specified Docker im-age from Docker Hub for the containers within those Pods. In doing so, it simplifies the man-agement and scaling of our PHP application within the Kubernetes cluster.

```
CloudComputing > php_mysql > kubernetes_deployment > ! php-deployment.yaml
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: php-app-deployment
5  spec:
6    replicas: 2
7    selector:
8      matchLabels:
9        app: php-app
10   template:
11     metadata:
12       labels:
13         app: php-app
14     spec:
15       containers:
16         - name: php-app
17           image: docker.io/group05/docker_group05:latest
18           ports:
19             - containerPort: 80
```

Figure 4: Caption

Figure 5 shows the php-service yaml file, In the service file we can specify the ports for Pods and the service port. This configuration allows us to access the container from the web browser when we define the service type as NordPort. This service type is designed to handle the re-quests from outside the cluster.

```
CloudComputing > php_mysql > kubernetes_deployment > ! php-service.yaml
1  apiVersion: v1
2  kind: Service
3  metadata:
4    name: php-app-service
5  spec:
6    selector:
7      app: php-app
8    ports:
9      - protocol: TCP
10       port: 80
11       targetPort: 80
12   type: NodePort
```

Figure 5: Caption

Advanced Features: In our deployment we tried to use the Secret which is a crucial component for maintaining secu-rity, used for securely store and manage sensitive information such as username and password. Link to **app-secret.yaml**

In order to check that all our pods are running, we use the command **kubect get all**. As shown in Figure 6, the PHP pods and the SQL pods are all up and running showing the successful deployment of our application using Kubernetes.

```
ubuntu@group05-instance:~/kube_php_webapp$ kubectl get all
NAME                                   READY   STATUS    RESTARTS   AGE
pod/db-7dc766ffb-kzt2n                 1/1     Running   0          17d
pod/php-app-deployment-7fcd58b5c5-9bmlx   1/1     Running   0          17d
pod/php-app-deployment-7fcd58b5c5-vkqn4   1/1     Running   0          17d
pod/wordpress-77dd8cc87d-gfj8m         1/1     Running   0          19d

NAME                    TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
service/db              NodePort    10.96.160.57    <none>        3306:32327/TCP   17d
service/kubernetes      ClusterIP   10.96.0.1       <none>        443/TCP          27d
service/php-app-service   NodePort    10.98.252.190   <none>        80:31671/TCP     17d
service/wordpress       NodePort    10.96.48.165    <none>        80:30325/TCP     19d

NAME                               READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/db                 1/1     1            1           17d
deployment.apps/php-app-deployment   2/2     2            2           17d
deployment.apps/wordpress          1/1     1            1           19d

NAME                                         DESIRED   CURRENT   READY   AGE
replicaset.apps/db-7dc766ffb                 1         1         1       17d
replicaset.apps/php-app-deployment-7fcd58b5c5   2         2         2       17d
replicaset.apps/wordpress-77dd8cc87d         1         1         1       19d
```

Figure 6: Caption

6 AWS EC2 DEPLOYMENT

In this phase, we cover the deployment of our containerized application on an AWS EC2 instance:

Amazon Elastic Compute Cloud (Amazon EC2) is a web service provided by Amazon Web Services (AWS) that allows users to rent virtual servers, known as instances, in the cloud. The following are some of the benefits of using the Amazon EC2 to deploy our web application :

- **Scalable Compute Capacity:** EC2 enables us to easily scale our compute capacity up or down based on our application's needs.
- **Instance Lifecycle:** EC2 instances can be started, stopped, terminated, and restarted as needed. This flexibility allows us to manage our resources efficiently and minimize costs.
- **Security and Networking:** EC2 instances can be deployed within Virtual Private Clouds (VPCs), which allow us to define our own network configuration, including subnets, routing, and access control. We can also attach security groups and network ACLs to control inbound and outbound traffic to our instances.
- **Storage Options:** EC2 instances can be attached to different types of storage, including Amazon Elastic Block Store (EBS) for persistent block storage and Amazon S3 for object storage. This allows us to choose the right storage solution for our application's data needs.

The following details our setup of the Amazon EC2 Instance and the deployment of our Web Application using the AWS EC2 instance :

- **AWS EC2 Instance Setup:** Sign up to the AWS console, log in with credentials, and then launch the EC2 instance with termination protection. Termination protection prevents from accidentally terminating an EC2 instance by giving the instance name. Choose Amazon Machine Image (AMI) which provides the information required to launch an instance.
- **Instance Type:** We choose the Ubuntu as our Instance with the type t2.medium. The t2.medium instance provides 2 virtuals CPU's and around 4GB of RAM.
- **Key Pair:** We used the existing key pairs vocky, Amazon EC2 then installed the key on the guest OS when the instance was launched. That way, we can log in to the instance, it also provides the private key, in this way will be authenticated for connecting to the instance.
- **Network setting:** We kept the default Network settings in which the instance will run and by default, it will assign a public IP address.
- **Security Group Rule:** We put an inbound security rule for SSH whise default is 22 and HTTP whose default port is 80. Then we launch the instance.

Once the instance is launched, it will show that the instance is running and it has passed the status check. Now we can connect to the instance using ssh with the vockey key pair as shown in Figure 7.

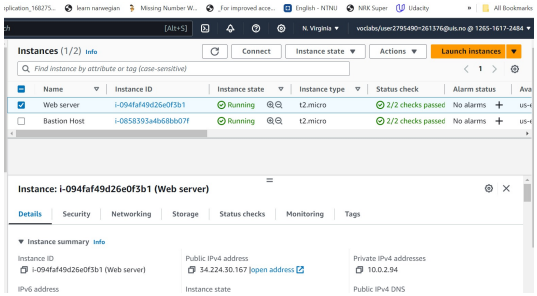

Figure 7: Caption

In the EC2 Instance Ubunty server, we need to follow the same steps to install the docker and kubernetes. After the installation is successful for both, we will create two folders , Docker and Kubernetes.

- **Docker Folder: Docker Image Transfer** Copy all the files including the PHP files , dockerfile and docker compose yml file into the folder. Run the command: **docker-compose up** . This will build and create the containers for the php and MySQL and we can view our application in the browser.
- **Kubernetes Folder:** In the Kubernetes Folder, we copy all the deployment and services files for PHP and MySQL. We deploy each of the files using **kubectl apply -f <filename>**. the deployments files will be created. We view whether the pods are running or not by running the following command : **kubectl get all**. It showed that the PHP pods are running but the SQL pod was in pending status, the error could be due to the fact that there was not sufficient volume. Due to this reason, we were not able to view our application in the web browser.

7 CONCLUSION AND DISCUSSION

Overall, this project allowed us to get hands on experience in using the latest cloud computing. Throughout this project, we faced a few challenges some of which are highlighted below :

- Flask Implementation using Docker : Our first web application was created using Python Flask. When we tried to deploy the application using Docker, there was a continuous permission denied error when it was trying to build the Flask Image. After many attempts to fix, the error could not be fixed due to time constraint and thus we changed our application to using PHP image.
- Initial Deployment Error in Kubernetes : In the initial stage of deploying our web application using Kubernetes, we had an error where Kubernetes could not load the MYSQL pod. It was a giving loop back connection error. After some troubleshoot, we realised the error was due to inconsistent sql username and password in the dockerfiles and kubernetes deployment file. Once that was fixed, the deployment was then successful.
- Deployment of Kubernetes Unsuccessful in Amazon EC2 Instance : The kubernetes deployment in the Amazoin EC2 instance was unsuccessful due to the state of mysql pod being pending. The reason was discovered that sufficient storage is not avialable for the MySQL to be running the EC2 instance as we were giving limited resources by the AWS Cloud and the storage was being shared by all group members.

In conclusion, Cloud computing provides businesses with scalability, cost-efficiency, flexibility, and robust security measures. Docker enables the creation of portable, efficient, and isolated containers for consistent application deployment. Kubernetes automates container orchestration, ensuring high availability, scaling, and self-healing. AWS EC2 instances offer flexibility, scalability, cost-effectiveness, security, and seamless integration with the broader AWS ecosystem. Together, these technologies empower organizations to build and manage modern, resilient, and agile IT infrastructures that meet the demands of today's digital world.

8 FUTURE WORK

There are several advanced features available within Docker, Kubernetes and the AWS EC2 instance for us to expand within our web application. Some of which we plan to explore and implements for our web application are :

- **Expand our application:** We can expand our current web application to include user authentication and more in order to make it more complex.
- **Kubernetes Secrets Management:** Kubernets has built-in support for managing secrets, allowing you to securely store and access sensitive information, such as API keys and database passwords, within your containers. Secrets are encrypted and can be accessed only by authorized services [6].
- **Kubernetes StatefulSets :**StatefulSets are used for stateful applications that require stable network identities and persistent storage. They ensure that pods are consistently named and maintain their state across rescheduling or scaling operations.
- **Kubernets PodSecurityPolicy (PSP)**: PSPs define security policies that restrict what pods can do, including what capabilities they have, which filesystems they can access, and which users and groups they can run as.
- **Kubernetes Advanced Scheduling:** Kubernetes supports advanced scheduling features like node affinity, node anti-affinity, and taints and tolerations to influence pod placement and control where pods can be scheduled.
- **AWS Elastic Load Balancing (ELB):**ELB distributes incoming traffic across multiple EC2 instances to ensure high availability, fault tolerance, and improved application scalability. It integrates seamlessly with Auto Scaling to adjust instance capacity based on demand.
- **Enhanced Networking:** EC2 instances can be configured with enhanced networking options such as Elastic Network Adapter (ENA) and Scalable Network Interface (ENA), which offer higher network performance and lower latency.

REFERENCES

[1] 2010. The Most Widely Deployed Open Source Cloud Software in the World. https://tex.stackexchange.com/questions/3587/how-can-i-use-bibtex-to-cite-a-web-page
[2] 2010. Pods. https://kubernetes.io/docs/concepts/workloads/pods/

[3] 2010. Using a Service to Expose Your App. https://kubernetes.io/docs/tutorials/kubernetes-basics/expose/expose-intro/
[4] IBM Cloud. 2022. What is Docker. https://www.ibm.com/topics/docker
[5] Dokcer Docs. 2010. Overview of best practices for writing Dockerfiles. https://docs.docker.com/develop/develop-images/dockerfile_best-practices/
[6] Kube By example. 2022. What are Secrets? https://kubebyexample.com/learning-paths/kubernetes-fundamentals/what-are-secrets
[7] RedHat Inc. 2020. What is Kubernetes? https://www.redhat.com/en/topics/containers/what-is-kubernetes
[8] Amazon EC2 Instance. 2023. What is Amazon EC2? https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html
[9] Babak Bashari Rad, Harrison John Bhatti, and Mohammad Ahmadi. 2017. An introduction to docker and analysis of its performance. *International Journal of Computer Science and Network Security (IJCSNS)* 17, 3 (2017), 228.