

cloud assig

by Mahamood Lidheem

Submission date: 24-Apr-2023 04:48PM (UTC+0100)

Submission ID: 203228244

File name: 32530_Mahamood_Lidheem_cloud_assig_488474_824929057.docx (783.06K)

Word count: 5384

Character count: 31307



Prifysgol Cymru
Y Drindod Dewi Sant
University of Wales
Trinity Saint David

Assignment for

CLOUD ENGINEERING PRINCIPLES AND PRACTICE

by

Mahamood Lidheem

To

Dr. Mike Dacey

Dr. Tim Bashford

April 2023

1 Table of Contents

1	TABLE OF CONTENTS	1
2	LIST OF FIGURES	2
3	LIST OF TABLES	2
4	INTRODUCTION	3
5	THE BENEFITS OF CONTAINERISATION	3
6	PORTABILITY	4
7	PUBLISHING IMAGES	5
8	CONTAINER VOLUMES.....	7
9	CONTAINER NETWORKING	8
10	CREATING YOUR OWN APPLICATION	10
10.1	SHORT VIDEO DEMONSTRATION.....	14
11	CLOUD COMPUTING CONCEPTS (LOAD BALANCING, SCALABILITY, APPLICATION GATEWAYS)	14
12	INVESTIGATION OF THE EXPANSION OF CLOUD COMPUTING AND SERVERLESS FUNCTIONS.....	16
13	CONCLUSION	19
14	REFERENCES.....	20

2 List of figures

FIGURE 1: DOCKERFILE	5
FIGURE 2: DOCKER HUB LOGIN, DOCKER BUILD	6
FIGURE 3: DOCKER TAG	6
FIGURE 4: DOCKER PUSH	6
FIGURE 5: DOCKER HUB	7
FIGURE 6: DOCKER VOLUME CREATION	8
FIGURE 7: CONTAINER TO LOCAL HOST MACHINE COMMUNICATION	8
FIGURE 8: DOCKER BRIDGE NETWORK	9
FIGURE 9: SIMPLE WEB SERVER TO DISPLAY HTML PAGES	13
FIGURE 10: PUSH THE DOCKER IMAGE TO A DOCKER HUB	13

3 List of tables

TABLE 1: FAAS VS CONTAINERIZATION	19
---	----

4 Introduction

In recent years, containerization and serverless computing have emerged as two significant technological advancements for software development and deployment. Serverless computing, also known as Function-as-a-Service (FaaS), allows developers to build and run applications without the need to manage servers, infrastructure, and other underlying resources. Containerization, on the other hand, involves packaging an application's code and dependencies in a self-contained and isolated environment, enabling it to run consistently across different IT infrastructures. Together, these two technologies have revolutionized modern software development and deployment by enabling faster, more reliable, and cost-effective application building and deployment.

5 The benefits of containerisation

Containerization is the process of creating an isolated and self-contained environment for a software application, called a container. This container includes the application's executable code, along with all its necessary dependencies such as configuration files, libraries, and frameworks, allowing the application to run seamlessly on any IT infrastructure. Essentially, containerization encapsulates an application's code and dependencies in a portable package that can be easily deployed and managed across different computing environments.

Containerization offers several benefits, including:

1. **Portability:** A container creates an executable package of software that is abstracted away from (not tied to or dependent upon) the host operating system, and hence, is portable and able to run uniformly and consistently across any platform or cloud. Efficiency: Containers are lightweight and fast, allowing for rapid application deployment and startup, which can help reduce time-to-market for new features and updates [1].
2. **Scalability:** Containers are highly scalable, making it easy to add or remove containers in response to changes in demand, which helps optimize resource usage and reduce costs.
3. **Consistency:** The consistency and fidelity of a modular development environment provide predictable results when moving code between development, test, and production systems. By verifying that the container encapsulates exact versions of necessary libraries and packages, it is possible to minimize the risk of bugs due to slightly different dependency revisions [2]. In other words, Containers ensure that applications run consistently across different environments, reducing the risk of configuration errors or version incompatibilities, which enhances the reliability and stability of application deployments.
4. **Microservices architecture support:** Containers enable the development and deployment of microservices architecture, allowing organizations to break down applications into smaller components, independent parts, with each part having its

own distinct set of responsibilities. which can improve flexibility, scalability, and agility [3].

5. **Simplified management:** Containerization enables the automation of application deployment, scaling, and management, which can reduce the workload on IT teams and improve the reliability and consistency of application deployments.

6 Portability

Portability is an important aspect of software development, and containerization technologies like Docker have made it easier to achieve. A good example of demonstrating portability is to build a Docker file to create an image that can run on any system that supports Docker. This ensures that the application is easily transferable and can be deployed on any infrastructure, providing a high level of flexibility and cost savings. For this example, we will be using NodeJS as a programming language and visual studio code as a code editor.

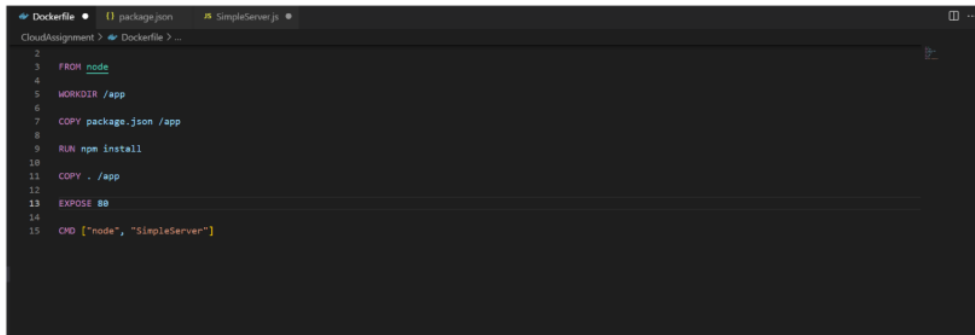
Steps to create an image based on a docker file:

- we need to go to the folder that contains our code and create a new file and name it "Dockerfile" (without any file extension), This is a special name which will be identified by Docker.
- In the first line of the Dockerfile, we will specify the base image that we will use, in our case it's NODE, typically here we start with the FROM instruction.
- Next, specify the working directory inside the container where our application will be located using the WORKDIR instruction (WORKDIR /app)
- Copy the files that our application requires into the container using (copy . .) , we basically specify two paths here each dot specify one path, the first dot tells that all the folders, sub folders and files here in our project should be copied. And the second dot is the path inside of the image where those files should be stored. for Every image and for also every container created based on an image has its own internal file system which is totally detached from our file system on our machine. It's hidden away inside of the Docker container And actually here it is a good idea to not use the root folder, the root entry in your Docker container, we can give any name in our case will name it APP and the command will be (copy . / APP).
- To copy the files required by our application into the container, we use the COPY instruction in our Dockerfile. The instruction takes two arguments, the source, and the destination. The source specifies the files to be copied, and the destination specifies where the files should be stored in the image.
In the source argument, we use (.) to specify the current directory, which includes all the folders, subfolders, and files in our project. In the destination argument, we specify the path inside the image where the files should be stored. It is important to note that every image and container created based on an image has its own internal file system that is totally detached from our file system on our machine. This internal file system is hidden away inside the Docker container.
It is generally not recommended to use the root folder as the destination for the COPY instruction, as this can cause security issues. Instead, we can create a new folder inside the image and use it as the destination. For example, we can create a folder called APP

and use the command (COPY . /APP) to copy the files from our project into the APP folder inside the image.

Using the COPY instruction in this way allows us to package all the necessary files for our application into the Docker image, making it easier to deploy and run the application on different systems.

- Our application requires dependencies to be installed, we will use run command to install them (RUN npm install)
 - The EXPOSE command is used to document the ports that the container will listen on at runtime, and is used to inform other developers or system administrators about the container's network requirements in other words it is an optional command. In our case we will use it (Expose 80)
 - Finally, specify the command that should be run when the container is started (CMD ["node", "server.js"])
 - To create our image we use the command (docker build .) after running this command image is created with an image id.
 - To start our container, we use the command docker run with the image id we had after creating our image docker run + image id
- Once the image is built, it can be used to run the application inside a Docker container on any platform that supports Docker.



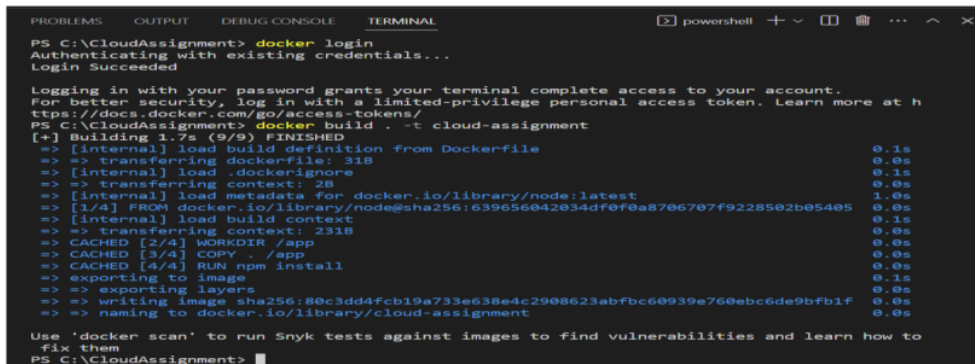
```
1 FROM node
2
3
4
5 WORKDIR /app
6
7 COPY package.json /app
8
9 RUN npm install
10
11 COPY . /app
12
13 EXPOSE 80
14
15 CMD ["node", "SimpleServer.js"]
```

Figure 1:Dockerfile

7 Publishing images

To push an image to Dockerhub, we first need to have a Dockerhub account and have logged in to Dockerhub via the terminal. Once you have done that, we can follow these steps:

- We Build our Docker image using the docker build command (docker build .)



```

PS C:\CloudAssignment> docker login
Authenticating with existing credentials...
Login Succeeded

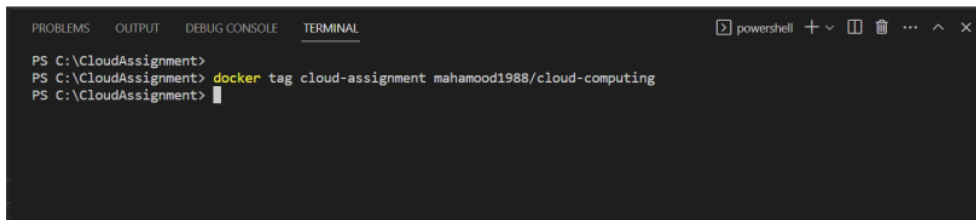
Logging in with your password grants your terminal complete access to your account.
For better security, log in with a limited-privilege personal access token. Learn more at h
tps://docs.docker.com/go/access-tokens/
PS C:\CloudAssignment> docker build -t cloud-assignment
[+] Building 1.7s (9/9) FINISHED
=> [internal] load build definition from Dockerfile 0.1s
=> => transferring dockerfile: 31B 0.0s
=> [internal] load .dockerignore 0.1s
=> [internal] load metadata for docker.io/library/node:latest 0.0s
=> [1/4] FROM docker.io/library/node:sha256:639656042034df0fa8706707f9228502b05405 1.0s
=> [internal] load build context 0.1s
=> => transferring context: 231B 0.0s
=> CACHED [2/4] WORKDIR /app 0.0s
=> CACHED [3/4] COPY . /app 0.0s
=> CACHED [4/4] RUN npm install 0.0s
=> exporting to image 0.1s
=> => exporting layers 0.0s
=> => writing image sha256:80c3dd4fcb19a733e638e4c2908623abfbc60939e760ebc6de9bfb1f 0.0s
=> => naming to docker.io/library/cloud-assignment 0.0s

Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to
fix them
PS C:\CloudAssignment>

```

Figure 2:docker hub login, docker build.

- We Tag our Docker image with your Dockerhub username and the image name or image id using tag command (tag "image id" username / repository name) in our case the command will be (tag cloud-assignment mahamood1988/cloud-computing)



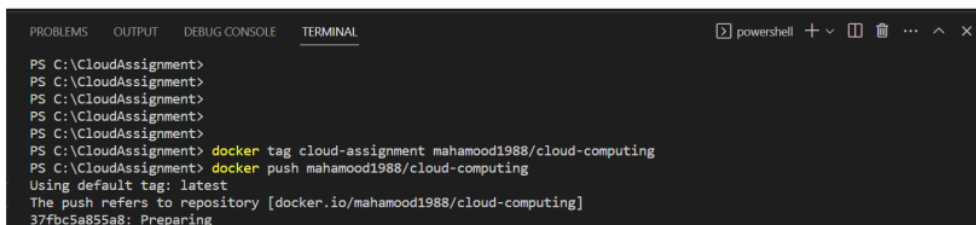
```

PS C:\CloudAssignment>
PS C:\CloudAssignment> docker tag cloud-assignment mahamood1988/cloud-computing
PS C:\CloudAssignment>

```

Figure 3:Docker Tag

- Push your Docker image to Dockerhub using the docker push command (push username/ repository name) in our case the command will be docker push mahamood1988/cloud-computing.



```

PS C:\CloudAssignment>
PS C:\CloudAssignment>
PS C:\CloudAssignment>
PS C:\CloudAssignment>
PS C:\CloudAssignment> docker tag cloud-assignment mahamood1988/cloud-computing
PS C:\CloudAssignment> docker push mahamood1988/cloud-computing
Using default tag: latest
The push refers to repository [docker.io/mahamood1988/cloud-computing]
37fbc5a855a8: Preparing

```

Figure 4: Docker Push

This will upload the image to Docker Hub and make it publicly available. Other people can then pull the image from Docker Hub using the docker pull command.

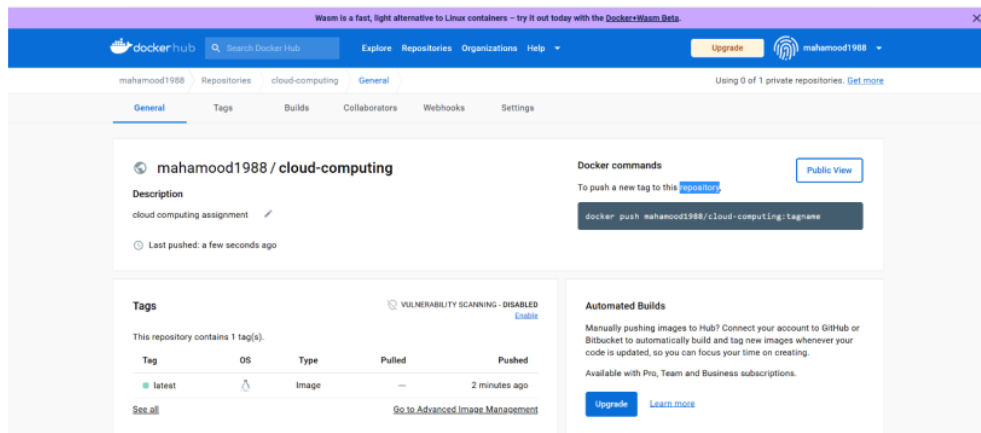


Figure 5:Docker Hub

8 Container volumes

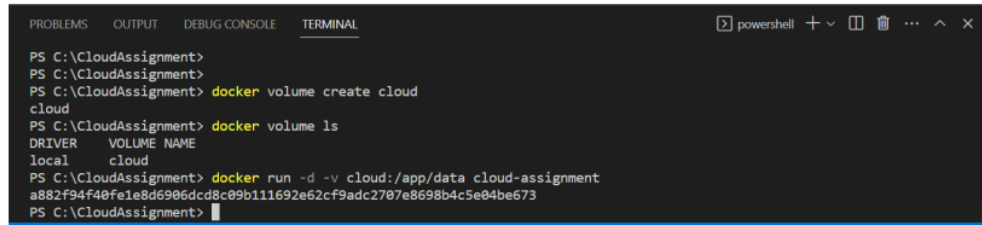
in reality, we are going to remove containers quite a bit, so If we change something about our code and be built a new image and start a new container will not restart the old container. And We want to use the new container which uses the latest code update. So therefore, in that case, we would be losing all the data. And that is exactly the problem we are facing now. so, we know that containers are able to write data but that this data is lost when the container is removed and that is the problem.

The solution is volumes which is a docker built in feature that will help us with persisting data and solve the problem we mentioned earlier.

Volumes are folders on your host machine, our computer which are mounted which basically means, made available or mapped into containers.

We make the Docker aware of these files which are then mapped to folders inside of a Docker container.so this instruction is only taking a snapshot of the path and files you would tell it to copy, and then it copies these files and folders into the image.so There's no on-going relation or connection it is just one time snapshot and that solved the problem we have described earlier. Here is example of using volume commands:

- We Open a terminal and run the create command to create new volume called computing (docker volume create cloud).
- To make sure the volume has been created we run the command (docker volume ls) to list all the volumes and we run the command (docker volume inspect cloud).
- Now you can use the volume in a Docker container. to start a container and mount the "cloud" volume to the container's "/app/data" we use the command (docker run -d -v cloud:/app/data cloud-assignment).



```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
PS C:\CloudAssignment>
PS C:\CloudAssignment>
PS C:\CloudAssignment> docker volume create cloud
cloud
PS C:\CloudAssignment> docker volume ls
DRIVER      VOLUME NAME
local       cloud
PS C:\CloudAssignment> docker run -d -v cloud:/app/data cloud-assignment
a882f94f40fe1e8d6906dcd8c09b111692e62cf9adc2707e8698b4c5e04be673
PS C:\CloudAssignment>

```

Figure 6: Docker Volume Creation

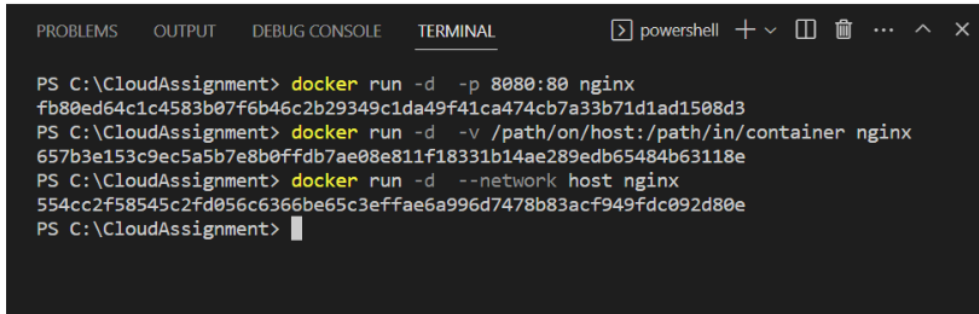
This will start a container using the image (cloud-assignment) and mount the "cloud" volume to the container's "/app/data" directory. Any data that is written to the "/app/data" directory inside the container will be stored in the "cloud" volume.

9 Container Networking

Container networking refers to the communication and connectivity between containers in a containerized environment. Containers are lightweight and portable units of software that can be run in isolation from the host operating system and other containers on the same host. Container networking enables communication between these isolated containers and with other systems in the network.

Docker takes care of the networking aspects so that the containers can communicate with other containers and also with the Docker Host. If you do an ifconfig on the Docker Host, you will see the Docker Ethernet adapter. This adapter is created when Docker is installed on the Docker Host [4]. There are several approaches to container networking, including:

- **Host Networking:** In this approach, containers share the network namespace with the host system, allowing them to use the same IP address and port space as the host. Container to local host machine communication in Docker can be achieved in several ways such as Port mapping and Shared volumes.



```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
PS C:\CloudAssignment> docker run -d -p 8080:80 nginx
fb80ed64c1c4583b07f6b46c2b29349c1da49f41ca474cb7a33b71d1ad1508d3
PS C:\CloudAssignment> docker run -d -v /path/on/host:/path/in/container nginx
657b3e153c9ec5a5b7e8b0ffdb7ae08e811f18331b14ae289edb65484b63118e
PS C:\CloudAssignment> docker run -d --network host nginx
554cc2f58545c2fd056c6366be65c3effae6a996d7478b83ac949fdc092d80e
PS C:\CloudAssignment>

```

Figure 7: Container to Local Host Machine Communication

- **Overlay Networking:** Overlay networking enables communication between containers running on different hosts. It creates an overlay network on top of the physical network, providing a virtual network that spans multiple hosts.
- **Network Plugins:** Many container orchestration platforms such as Docker Swarm, Kubernetes, and Mesos support network plugins that extend the networking capabilities of the platform. These plugins enable advanced networking features such as load balancing, service discovery, and network security.
- **Bridge Networking:** In bridge networking, containers are connected to a virtual bridge that provides a virtual network interface to communicate with other containers and the outside world.

An example of docker network is the connection between two containers on the same host using Bridge networking, by default, Docker creates a bridge network for containers. This allows containers to communicate with each other, but not with the host machine or other networks. Each container on the bridge network is assigned a unique IP address and can communicate with other containers on the same network using this IP address.

To create bridge network, we use the following steps:

- Create docker bridge network using command (`docker network create my-assignment`).
- Create two images called first-image and second-image using command (`docker build -t first-image .`) and (`docker build -t second-image .`).
- Running two containers called first-container and second-container and connect them to the same network we created earlier (`my-assignment`)
- Executing ping command to check the connectivity between both containers and to execute this command(`docker exec -it first-container ping second-container`), I had to run a shell inside our containers and install ping program inside our containers using the following commands (`docker exec -it first-container /bin/bash`) and (`docker exec -it second-container /bin/bash`) and then installing iputils-ping inside both containers using the following command(`apt-get update && apt-get install -y iputils-ping`)

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
-> CACHED [stage-1 5/5] COPY . /app
0.0s
-> exporting to image
0.0s
-> exporting layers
0.0s
-> writing image sha256:e6c864aaaa15eadd27bd4f9d81ff81a8213bf91659e51d2954cb44ad37168b4
0.0s
-> naming to docker.io/library/second-image
0.0s

Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them
PS C:\CloudAssignment> docker run -d --name first-container --network my-assignment first-image
f4442bc7d7dfe8d9deec0897a078aa4be4d54d6d8cb076d33a2807ba64
PS C:\CloudAssignment> docker run -d --name second-container --network my-assignment second-image
fbbe565e18e1961aff4fcd4c0c9f913594519f708980490fd8741f95133c962a
PS C:\CloudAssignment> docker exec -it first-container ping second-container
PING second-container (172.18.0.3): 56(84) bytes of data:
64 bytes from second-container.my-assignment (172.18.0.3): icmp_seq=1 ttl=64 time=0.144 ms
64 bytes from second-container.my-assignment (172.18.0.3): icmp_seq=2 ttl=64 time=0.289 ms
64 bytes from second-container.my-assignment (172.18.0.3): icmp_seq=3 ttl=64 time=0.096 ms
64 bytes from second-container.my-assignment (172.18.0.3): icmp_seq=4 ttl=64 time=0.152 ms
64 bytes from second-container.my-assignment (172.18.0.3): icmp_seq=5 ttl=64 time=0.552 ms
64 bytes from second-container.my-assignment (172.18.0.3): icmp_seq=6 ttl=64 time=0.081 ms
64 bytes from second-container.my-assignment (172.18.0.3): icmp_seq=7 ttl=64 time=0.100 ms

```

Figure 8:Docker Bridge Network

10 Creating your own application

To Write a simple application running as a Docker image, such as a simple web server to display HTML pages we will follow these steps:

- **Choosing a programming language and framework:** To create an application using Docker, we first need to choose a programming language and framework. Docker supports a wide range of programming languages and frameworks, including .NET, Angular, Django, Express, FastAPI, Flask, Node.js, React | Rails, Spark, Spring Boot, Vue.js, Go, Java, JavaScript, PHP, Python, Ruby, Rust and TypeScript [5]. For our application we will be using NODEJS as our frameworks .
- **Writing application code:** Once you have chosen a programming language and framework, we can start writing our application code using visual studio code. Our application will consists of the following files styles.css, Dockerfile, package.json and simpleserver.js:

styles.css

```
html {
  font-family: sans-serif;
}

body {
  margin: 0;
}

section,
form {
  padding: 1rem;
  border-radius: 12px;
  box-shadow: 0 2px 8px rgba(0, 0, 0, 0.26);
  margin: 2rem auto;
  max-width: 40rem;
}

.form-control {
  margin: 0.5rem 0;
}

input {
  font: inherit;
}

input,
label {
  display: block;
```

```
label {
  font-weight: bold;
  margin-bottom: 0.5rem;
}

button {
  background-color: #2f005a;
  border: 1px solid #2f005a;
  color: white;
  cursor: pointer;
  padding: 0.5rem 1.5rem;
}

button:hover,
button:active {
  background-color: #50005a;
  border-color: #50005a;
}
```

package.json

```
{
  "name": "cloud-assignment",
  "version": "1.0.0",
  "description": "",
  "main": "SimpleServer.js",
  "author": "Mahamood Lidheem / Master course student",
  "license": "UWTSD",
  "dependencies": {
    "express": "^4.17.1",
    "body-parser": "1.19.0"
  }
}
```

SimpleServer.js

```
const server = require("http").createServer((req, res) => {
  res.writeHead(200, { "Content-Type": "text/html" });
  res.write("<html><body><h1>Welcome to Cloud Engineering Assignment!</h1><p>This is simple web server to display HTML pages.</p></body></html>");
  res.end();
});

server.listen(80);
```

- **Create a Dockerfile:** A Dockerfile is a set of instructions that tells Docker how to build an image. A typical Dockerfile is made up of these instructions: A FROM instruction that tells Docker what the base image is, An ENV instruction to pass an environment variable, A RUN instruction to run some shell commands (for example, install-dependent programs not available in the base image) and A CMD or an ENTRYPOINT instruction that tells Docker which executable to run when a container is started [6].

Dockerfile

```
FROM node
WORKDIR /app
COPY package.json /app
RUN npm install
COPY . /app
EXPOSE 80
CMD ["node", "SimpleServer"]
```

- **Build a Docker image:** To build a Docker image, we need to run the docker build command. This command reads the instructions in the Dockerfile and creates a Docker image.

```
PS C:\CloudAssignment> docker build -t cloud-assignment .
```

- **Run the Docker container:** Once we have built a Docker image, we can run it as a Docker container. We can use the docker run command to start a container (assignment-container) from the image you built in the previous step (cloud-assignment).

```
PS C:\CloudAssignment> docker run -d --name assign-container cloud-assignment
```

- **Test our application:** After you have started the Docker container, you can test your application by accessing it through a web browser through our local host (<http://localhost:3000/>)

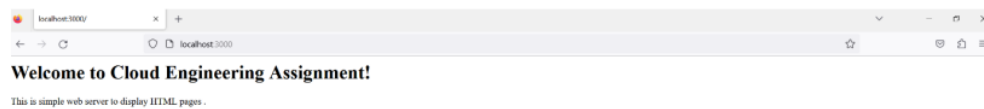


Figure 9:simple web server to display HTML pages

- **Push the Docker image to a docker hub:** If we want to share your Docker image with others or deploy it to a production environment, you can push it to a Docker hub, this is optional. And to do so we will be using the following instructions:
 - `docker tag cloud-assingment mahamood1988/cloud-assingment`
 - `docker push mahamood1988/cloud-assingment`

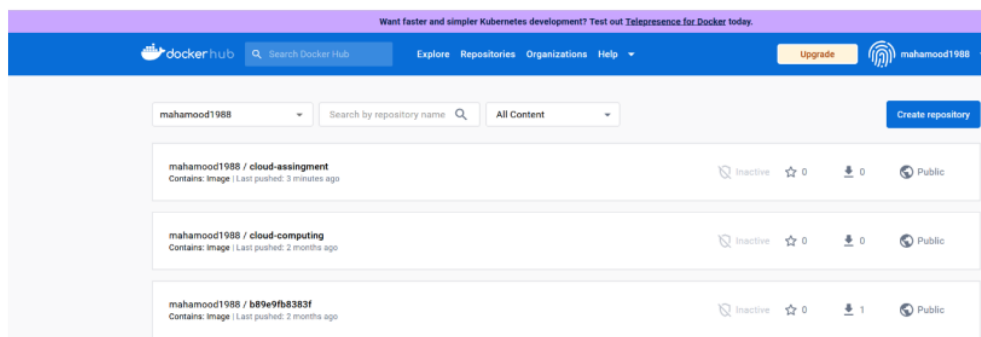


Figure 10: Push the Docker image to a docker hub

In summary, creating an application using Docker involves choosing a programming language and framework, writing the application code, creating a Dockerfile, building a Docker image, running a Docker container, testing the application, and pushing the Docker image to a registry if desired. Docker provides an efficient and portable way to package and distribute applications, making it a popular choice for modern application development and deployment.

10.1 Short video demonstration.

<https://www.youtube.com/watch?v=RhMbUz5ybGc>

11 Cloud computing concepts (load balancing, scalability, application gateways)

Brief descriptions of the three main concepts in cloud computing, along with how cloud providers implement them:

1. **Load Balancing:** Load balancing is the method of distributing network traffic equally across a pool of resources that support an application. Modern applications must process millions of users simultaneously and return the correct text, videos, images, and other data to each user in a fast and reliable manner. To handle such high volumes of traffic, most applications have many resource servers with duplicate data between them. A load balancer is a device that sits between the user and the server group and acts as an invisible facilitator, ensuring that all resource servers are used equally [7].

Cloud providers implement load balancing through a variety of techniques, including:

- **DNS-based load balancing:** This technique uses DNS to direct traffic to multiple servers with the same hostname. In the simplest deployment, a single computer hosts and serves the data for a domain; when a client requests resolution of the domain name, DNS returns the single server's IP address. Today many domains use multiple server machines – to protect against server failure or handle high traffic volume, among other reasons – and in this case DNS returns a list of all the servers' IP addresses in response to a name resolution request [8].
 - **Network Load Balancer:** The Network Load Balancing (NLB) feature distributes traffic across several servers by using the TCP/IP networking protocol. By combining two or more computers that are running applications into a single virtual cluster, NLB provides reliability and performance for web servers and other mission-critical servers [9]. This is a layer 4 load balancer that can handle millions of requests per second while maintaining high throughput. Examples include AWS NLB, Azure Load Balancer, and Google Cloud Load Balancer.
 - **Application Load Balancer:** This is a layer 7 load balancer that can route traffic based on application-level content, such as HTTP headers or cookies. Examples include AWS ALB, Azure Application Gateway, and Google Cloud HTTP(S) Load Balancer.
2. **Scalability:** Scalability refers to the ability of a system to handle increasing workload by adding resources such as CPU, memory, storage, and network bandwidth. Cloud providers implement scalability by offering the following features:
 - **Auto Scaling:** This feature automatically adjusts the number of resources in response to changes in demand. For example, AWS Auto Scaling can automatically add or remove EC2 instances based on predefined policies.

- **Elastic Load Balancing:** This feature can automatically distribute incoming traffic across multiple instances and can also scale instances up or down based on demand. For example, AWS Elastic Load Balancer can distribute traffic to instances in multiple availability zones and automatically add or remove instances based on demand.
 - **Container Orchestration:** This feature manages the deployment and scaling of containerized applications. Examples include Kubernetes on Google Cloud, Azure Kubernetes Service (AKS), and Amazon Elastic Kubernetes Service (EKS).
3. **Application Gateways:** Application gateways provide a single-entry point for incoming traffic, and can perform various functions such as routing, authentication, and security. Cloud providers implement application gateways through the following services:
- **API Gateway:** The API Gateway serves as the central point for all client requests to backend services, providing a range of features such as routing, security, traffic management, protocol transformation, and analytics and monitoring. In microservices architectures, where multiple backend services need to be exposed through a single-entry point, the API Gateway is particularly crucial. By abstracting the underlying services from clients, it simplifies the development and upkeep of intricate distributed systems. In cloud computing, managed API Gateway services are available from cloud providers, providing scalability, security, integration, management, and cost-effectiveness benefits. RESTful APIs, which adhere to the REST architectural style and use standard HTTP methods for communication, are a popular use case for API Gateways in the cloud.
 - **Web Application Firewall (WAF):** This service provides protection against common web exploits such as SQL injection and cross-site scripting (XSS). Examples include AWS WAF, Azure Web Application Firewall, and Google Cloud Armor.
 - **Azure Application Gateway:** is a web traffic load balancer that enables you to manage traffic to your web applications. Traditional load balancers operate at the transport layer (OSI layer 4 - TCP and UDP) and route traffic based on source IP address and port, to a destination IP address and port [10]. This is a layer 7 load balancer and application gateway that can perform various tasks such as SSL termination, URL-based routing, and cookie-based session affinity.
4. **Load balancing and scaling** are two interconnected concepts that work together to enhance application performance and availability in cloud computing environments. Load balancing is a process of distributing incoming network traffic across multiple servers to optimize resource utilization, reduce downtime, and improve application responsiveness. By evenly distributing traffic among servers, load balancing ensures that no single server is overwhelmed with traffic, which results in improved performance and availability.

However, there may be instances where servers are unable to handle the workload even with load balancing. This is where scaling becomes necessary. Scaling refers to the system's ability to handle increased traffic by adding resources such as CPU, memory, storage, and network bandwidth.

By scaling up or down in response to changes in demand, the application can maintain optimal performance and availability levels. Scaling allows additional resources to be added to manage increased traffic and workload, ensuring that the application continues to perform well and remain available.

Thus, load balancing and scaling work in tandem to ensure that the application can handle increased traffic, maintain high availability, and provide optimal performance to users. Load balancing distributes traffic across multiple servers, while scaling provides additional resources to manage increased demand, creating a system that can handle growing traffic and remain performant.

- 5. Demonstration Example:** This Example demonstrates the benefits of combining all three technologies together in providing a service to a customer. Let's imagine that a business offers an online shopping service that sees a lot of traffic around the holidays. Even after deploying load balancing to split traffic among several servers, the business still experiences delayed page loads and unsatisfied customers during peak hours. The business decides to achieve scalability by adding more servers to their cloud infrastructure during periods of high usage to get around this problem.

They will be able to manage the increasing traffic while maintaining the responsiveness and availability of the applications. Also, the business is worried about making sure that the sensitive information of its consumers is protected while they are purchasing.

They decide to use an application gateway, which examines and filters incoming traffic and adds an extra layer of protection, to address this. In addition to defending against standard security concerns like SQL injection and DDoS attacks, this aids in preventing unauthorised access.

During busy times, the business provides its clients with a seamless and safe online purchasing experience by combining load balancing, scalability, and application gateways. The application gateway guarantees the protection of sensitive client data, load balancing aids in the even distribution of traffic, and scalability enables the addition of resources to manage rising demand. Better performance, availability, and security are the end outcomes of this, which ultimately improve customer pleasure.

In conclusion, the performance, security, and availability of cloud-based applications and services can be significantly improved by combining load balancing, scalability, and application gateways.

12 Investigation of the expansion of cloud computing and serverless functions

Expansion of cloud computing:

There are several driving forces that have led to the expansion of cloud computing, including cost and flexibility. Here are some of the main factors:

- **Cost Savings:** One of the primary driving forces behind the expansion of cloud computing is cost savings. By leveraging cloud computing, organizations can reduce their upfront costs on hardware and software, as well as ongoing maintenance and upgrades. Cloud computing allows organizations to pay for only the resources they use, rather than having to invest in and maintain their own infrastructure. Cloud computing eliminates the capital expense of buying hardware and software and setting up and running on-site datacentres—the racks of servers, the round-the-clock electricity for power and cooling, and the IT experts for managing the infrastructure. It adds up fast [11].
- **Flexibility:** Moreover, cloud computing gives businesses more freedom. Without spending money on additional hardware, organisations can quickly scale their resource needs up or down with cloud computing. This flexibility allows organizations to be more agile and responsive to changing market conditions.
- **Accessibility:** Accessibility is another factor influencing the growth of cloud computing. Organizations can access their data and apps using cloud computing from any location in the world as long as they have an internet connection. This makes it possible to work remotely and to collaborate and produce more effectively.
- **Security:** While security concerns have been a barrier to adoption for some organizations, cloud computing providers have invested heavily in Cloud security. Cloud security is a discipline of cyber security dedicated to securing cloud computing systems. This includes keeping data private and safe across online-based infrastructure, applications, and platforms. Securing these systems involves the efforts of cloud providers and the clients that use them, whether an individual, small to medium business, or enterprise uses [12]. In many cases, cloud computing can provide greater security than on-premises solutions, due to the expertise and resources of cloud providers.

Serverless Functions and Containerization:

Functions as a service (FaaS) and containerization are two different approaches to deploying and running applications in the cloud.

Function as a Service (FaaS) is a software paradigm that is becoming increasingly popular. Multiple cloud providers offer FaaS as the interface to usage-driven, stateless (serverless) backend services. FaaS offers an intuitive, event-based interface for developing cloud-based applications [13].

In FaaS, the user uploads code to the cloud, and the provider enables a handle (e.g., a URL) for the code to be run. The choices of which resources to allocate, when to allocate them, and for how long to retain them, still have to be made, but they are shifted to the cloud provider [13]. In other words Serverless is an architecture that relies heavily on abstracting away the host environment to focus on code. It can be thought of as less server [14].

There are two examples of FAAS:

- **Azure Functions:** Azure Functions provide a serverless compute experience. A function is invoked by a trigger (such as access to an HTTP endpoint or a timer) and executes a block of code or business logic. Functions also support specialized bindings that connect to resources like storage and queues [14].
- **AWS Lambda:** is a compute service that lets you run code without provisioning or managing servers. Lambda runs your code on a high-availability compute infrastructure and performs all of the administration of the compute resources, including server and operating system maintenance, capacity provisioning and automatic scaling, and logging. With Lambda, you can run code for virtually any type of application or backend service. All you need to do is supply your code in one of the languages that Lambda supports [15].

Containerization, on the other hand, involves packaging applications and their dependencies into self-contained, portable units called containers. Containers are isolated from the host system and can be deployed and run on any compatible platform, providing flexibility and portability. Containers are typically used to run larger applications, rather than individual functions. There are several differences between FaaS and containerization. We will list some of the key differences between them in the following table:

Category	Functions as a Service (FaaS)	Containerization
Scalability	Automatically scales to meet demand	Requires more manual management and configuration for scaling
Deployment	Fast deployment and execution time	Longer deployment and configuration time
Abstracts	Runtime	OS Host
Cost	Cost-effective since resources are only consumed during execution	Potentially more expensive to run due to more manual management and configuration
Resource Usage	Small and focused functions	Larger and more complex applications
Dependency Management	Fewer dependencies, easier management	More dependencies and potentially more complex management
Lifetime	Milliseconds to Minutes	Minutes to Days
Use Cases	Ideal for event-driven, smaller applications	Ideal for larger, more complex applications

Table 1:FAAs vs Containerization

Overall, the choice between FaaS and containerization depends on the specific needs of the application and the resources available to manage it. Both FaaS and containerization have their own unique strengths and weaknesses, so developers should carefully consider the use case and resource requirements before making a decision.

13 Conclusion

In conclusion, containerization such as docker and serverless computing such as (Azure Functions) have altered the way that software is developed and deployed today, enabling developers to create, and run applications in a more efficient, dependable, and time- and money-saving manner.

Containerization has many advantages, such as portability, efficiency, scalability, consistency, support for microservices architecture, and streamlined management, by enabling developers to encapsulate applications and dependencies in a self-contained and isolated environment.

Combining serverless computing with application development makes it simpler to meet changing customer demands and maintain competitiveness in today's fast-paced business environment. Developers can concentrate on creating and deploying applications without having to worry about underlying infrastructure and resources.

Hence, containerization and serverless computing have developed into crucial elements of contemporary software development and deployment, and their continuous development and growth promise to revolutionise the sector even more.

14 References

- [1] "IBM," IBM, 2023. [Online]. Available: <https://www.ibm.com/uk-en/cloud/learn/containerization>.
- [2] AWS, "Docker on AWS whitepaper," Amazon Web Services, 2023.
- [3] Google Cloud, "Microservices Architecture," Google Cloud, 2023. [Online]. Available: [https://cloud.google.com/learn/what-is-microservices-architecture#:~:text=Microservices%20architecture%20\(often%20shortened%20to,its%20own%20realm%20of%20responsibility..](https://cloud.google.com/learn/what-is-microservices-architecture#:~:text=Microservices%20architecture%20(often%20shortened%20to,its%20own%20realm%20of%20responsibility..)
- [4] D. Hutten, Docker BULID. SHIP. RUN., 2022.
- [5] docker, "Framework And Languages," Docker, 2022. [Online]. Available: <https://docs.docker.com/samples/#frameworks>.
- [6] S. Bhat, Practical Docker with Python, Bangalore, Karnataka, India: pringer Science+Business Media, 2018 .
- [7] A. aws, "What Is Load Balancing? Load Balancing Explained- AWS," Amazon Web Services, Inc, 2023. [Online]. Available: <https://aws.amazon.com/what-is/load-balancing/>.
- [8] NGINX, "What Is DNS Load Balancing?," F5, Inc., 2023. [Online]. Available: <https://www.nginx.com/resources/glossary/dns-load-balancing/>.
- [9] Microsoft, "Network Load Balancing," Microsoft , 2023. [Online]. Available: <https://learn.microsoft.com/en-us/windows-server/networking/technologies/network-load-balancing>.
- [1 Microsoft, "What is Azure Application Gateway?," Microsoft , 2023. [Online]. Available: 0] <https://learn.microsoft.com/en-us/azure/application-gateway/overview>.
- [1 Microsoft, "Top benefits of cloud computing," Microsoft, 2023. [Online]. Available: 1] <https://azure.microsoft.com/en-gb/resources/cloud-computing-dictionary/what-is-cloud-computing>.
- [1 kaspersky, "What is Cloud Security?," AO Kaspersky Lab, 2023. [Online]. Available: 2] <https://www.kaspersky.co.uk/resource-center/definitions/what-is-cloud-security>.
- [1 M. Shahrad, R. Fonseca, Í. Goiri, G. Chaudhry, P. Batum, J. Cooke, E. Laureano, C. Tresness, M. 3] Russinovich and R. Bianchini, "Serverless in the Wild: Characterizing and Optimizing the Serverless Workload at a Large Cloud Provider," 2020.
- [1 C. P. W. W. S. Jeremy Likness, Serverless apps: Architecture, patterns, and Azure implementation, 4] Washington: Microsoft Developer Division, .NET, and Visual Studio product teams , 2022.
- [1 A. AWS, "What is AWS Lambda?," Amazon Web Services, Inc., 2023. [Online]. Available: 5] <https://docs.aws.amazon.com/lambda/latest/dg/welcome.html>. [Accessed 2023 MARCH 04].

[1 G. N. Schenker, Learn Docker – Fundamentals of Docker 18.x, Birmingham: Packt Publishing Ltd., 6] 2018.

[1 T. K. Authors, “Kubernetes Components,” Kubernetes, 2023. [Online]. Available:
7] <https://kubernetes.io/docs/concepts/overview/components/>.

[1 G. Sayfan, Mastering Kubernetes , Automating container deployment and management,
8] Birmingham: Packt Publishing Ltd, 2017.

FINAL GRADE

86/100

GENERAL COMMENTS

Instructor

Docker Overview + Benefits (20): 16.5

Docker Examples (30): 27

Video (5): 5

Cloud Comp tech (25): 22.5

Cloud Comp functions (20): 15

Feedback:

=====

You could have provided a brief overview of containerisation to help clarify the benefits of the approach.

The cloud tech discussion was good, I would have liked a little more technical detail but overall the main concepts were covered very well. Likewise, the discussion of cloud drivers is good and covers the main points quite well. I would have liked more discussion of when containerisation vs serverless approaches are most appropriate.

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

PAGE 15

PAGE 16

PAGE 17

PAGE 18

PAGE 19

PAGE 20

PAGE 21

PAGE 22
