Lab 9
ICS423 - Internet of Things

Jayant Kolapkar - 2021BCS0132

Question

Task 1: Write golang-based services (2 numbers) on docker containers

Task 2: Write nodejs-based services (2 numbers) on docker containers

Task 1

1. Introduction

This project involves developing two microservices using Golang, containerizing them using Docker, and orchestrating them with Docker Compose. The microservices perform basic arithmetic operations:

- Service A: Addition (/add)
- Service B: Multiplication (/multiply)

Each service runs independently in a Docker container, ensuring scalability and ease of deployment.

---

2. Methodology

The following steps were followed:

1. Developed two Golang microservices using net/http.
2. Containerized each service using Docker.
3. Created a Docker Compose file to orchestrate the services.
4. Built and deployed the containers.
5. Tested the endpoints using curl.

3. Implementation

3.1. Service A: Addition Microservice

1. Create serviceA/go.mod

module serviceA

go 1.24

2. Create serviceA/main.go

```
package main

import (
    "encoding/json"
    "fmt"
    "net/http"
)
type Request struct {
    A float64 `json:"a"`
    B float64 `json:"b"`
}

type Response struct {
    Result float64 `json:"result"`
}

func addHandler(w http.ResponseWriter, r *http.Request) {
    var req Request
    err := json.NewDecoder(r.Body).Decode(&req)
    if err != nil {
        http.Error(w, "Invalid request", http.StatusBadRequest)
        return
```

```
    }

    result := req.A + req.B
    resp := Response{Result: result}

    w.Header().Set("Content-Type", "application/json")
    json.NewEncoder(w).Encode(resp)
}

func main() {
    http.HandleFunc("/add", addHandler)
    fmt.Println("Adder Service running on port 8081...")
    http.ListenAndServe(":8081", nil)
}
```

3. Create serviceA/Dockerfile

```
FROM golang:1.24

WORKDIR /app

COPY . .

RUN go mod tidy && go build -o main .

CMD ["/app/main"]
```

3.2. Service B: Multiplication Microservice

1. Create serviceB/go.mod

module serviceB

go 1.24

2. Create serviceB/main.go

```
package main

import (
    "encoding/json"
    "fmt"
    "net/http"
)

type Request struct {
    A float64 `json:"a"`
    B float64 `json:"b"`
}

type Response struct {
    Result float64 `json:"result"`
}

func multiplyHandler(w http.ResponseWriter, r *http.Request) {
    var req Request
    err := json.NewDecoder(r.Body).Decode(&req)
    if err != nil {
        http.Error(w, "Invalid request", http.StatusBadRequest)
        return
    }

    result := req.A * req.B
    resp := Response{Result: result}

    w.Header().Set("Content-Type", "application/json")
    json.NewEncoder(w).Encode(resp)
}

func main() {
    http.HandleFunc("/multiply", multiplyHandler)
    fmt.Println("Multiplier Service running on port 8082...")
```

```
    http.ListenAndServe(":8082", nil)
}
```

3. Create serviceB/Dockerfile

```
FROM golang:1.24

WORKDIR /app

COPY . .

RUN go mod tidy && go build -o main .

CMD ["/app/main"]
```

4. Deployment Steps

Step 1: Build and Start Containers

docker-compose up –build

```
WARN[0000] /home/jayant/IoTL9/Golang/docker-compose.yml: `version` is obsolete
[+] Building 27.5s (7/13)                                                                                  docker:default
 => [golang_service_2 internal] load build definition from Dockerfile                                                0.0s
 => => transferring dockerfile: 135B                                                                                 0.0s
 => [golang_service_1 internal] load metadata for docker.io/library/golang:1.20                                      4.2s
 => [golang_service_1 internal] load build definition from Dockerfile                                                0.0s
 => => transferring dockerfile: 135B                                                                                 0.0s
 => [golang_service_1 internal] load .dockerignore                                                                   0.0s
 => => transferring context: 2B                                                                                      0.0s
 => [golang_service_2 internal] load .dockerignore                                                                   0.0s
 => => transferring context: 2B                                                                                      0.0s
 => [golang_service_2 1/4] FROM docker.io/library/golang:1.20@sha256:8f9af7094d0cb27cc783c697ac5ba25efdc4da35f85    23.2s
 => => resolve docker.io/library/golang:1.20@sha256:8f9af7094d0cb27cc783c697ac5ba25efdc4da35f8526db21f7aebb0b0b4f    0.0s
 => => sha256:c1a446d8f0c5658cc9af7b207a521995cf26bcfe1c9e6a4e148ba4eb8ed2e8b5 1.79kB / 1.79kB                       0.0s
 => => sha256:d5beeac3653f7c94aeda867588172915ce848d8d49d4ca24d24245505949b64d 2.75kB / 2.75kB                       0.0s
 => => sha256:68e92d11b04ec0fe48e60d59964704aca234084f87af5d1a068c49456b37fe3d 15.73MB / 64.14MB                    23.2s
 => => sha256:8f9af7094d0cb27cc783c697ac5ba25efdc4da35f8526db21f7aebb0b0b4f18a 2.36kB / 2.36kB                       0.0s
 => => sha256:6a299ae9cfd996c1149a699d36cdaa76fa332c8e9d66d6678fa9a231d9ead04c 14.68MB / 49.58MB                    23.2s
 => => sha256:e08e8703b2fb5e50153f792f3192087d26970d262806b397049d61b9a14b3af5 9.44MB / 24.05MB                     23.2s
 => [golang_service_2 internal] load build context                                                                   0.0s
 => => transferring context: 459B                                                                                    0.0s
 => [golang_service_1 internal] load build context                                                                   0.0s
 => => transferring context: 459B                                                                                    0.0s
```

```
[+] Building 75.6s (15/15) FINISHED
 => [serviceb internal] load build definition from Dockerfile
 => => transferring dockerfile: 145B
 => [serviceb internal] load .dockerignore
 => => transferring context: 2B
 => [servicea internal] load metadata for docker.io/library/golang:1.24
 => [servicea internal] load build definition from Dockerfile
 => => transferring dockerfile: 147B
 => [servicea internal] load .dockerignore
 => => transferring context: 2B
 => [servicea 1/4] FROM docker.io/library/golang:1.24@sha256:c5adecdb7b3f8c5ca3c88648a861882849cc8b02fed68ece31e25de88ad13418
 => => resolve docker.io/library/golang:1.24@sha256:c5adecdb7b3f8c5ca3c88648a861882849cc8b02fed68ece31e25de88ad13418
 => => sha256:29616a01ff27428aaf681f7abd8439b6aca78cadb73fac0475196cb261a34b91 2.80kB / 2.80kB
 => => sha256:155ad54a8b2812a0ec559ff82c0c6f0f0dddb337a226b11879f09e15f67b69fc 48.48MB / 48.48MB
 => => sha256:1d281e50d3e435595c266df06531a7e8c2ebb0c185622c8ab2eed8d760e6576b 64.39MB / 64.39MB
 => => sha256:c5adecdb7b3f8c5ca3c88648a861882849cc8b02fed68ece31e25de88ad13418 10.06kB / 10.06kB
 => => sha256:7ebae3e990ad9a8406da7ec4cd127decc408c98f8a88d0f2bef629bcaff691cd 2.32kB / 2.32kB
 => => sha256:8031108f3cda87bb32f090262d0109c8a0db99168050967becefad502e9a681b 24.06MB / 24.06MB
 => => sha256:ec6bde4714ee6491f090f4367e5c540e43ac6f9b238b25b0838f2a9d1d10f577 92.33MB / 92.33MB
 => => extracting sha256:155ad54a8b2812a0ec559ff82c0c6f0f0dddb337a226b11879f09e15f67b69fc
 => => sha256:178cc98ff0842a2601bbc4e7db3db70a323469849a03684d1b9b21e7f825b7e4 78.93MB / 78.93MB
 => => extracting sha256:8031108f3cda87bb32f090262d0109c8a0db99168050967becefad502e9a681b
 => => sha256:c10ccacbd8ad4103e29b0a10e17fcfdbc768b1361d50b2c9222d457544de4cb1 126B / 126B
 => => extracting sha256:1d281e50d3e435595c266df06531a7e8c2ebb0c185622c8ab2eed8d760e6576b
```

```
 => [serviceb] exporting to image
 => => exporting layers
 => [servicea 4/4] RUN go mod tidy && go build -o main .
 => [serviceb] exporting to image
 => [servicea 4/4] RUN go mod tidy && go build -o main .
 => [servicea 4/4] RUN go mod tidy && go build -o main .
 => [serviceb] exporting to image
 => => exporting layers
 => => writing image sha256:7ef4a56d00356d335aff47b605d71c353899ccf21433b8a012c89249bf3c7bb4
 => => naming to docker.io/library/lab9-serviceb
 => [servicea] exporting to image
 => => exporting layers
 => => writing image sha256:69351b7e070abb7338dcb0d377b00c6e1391e1bb68f16e46bd2d03ca64a7b7f2
 => => naming to docker.io/library/lab9-servicea
[+] Running 3/3
 ✓ Network lab9_default       Created
 ✓ Container lab9-serviceb-1  Created
 ✓ Container lab9-servicea-1  Created
Attaching to servicea-1, serviceb-1
servicea-1  | Adder Service running on port 8081...
serviceb-1  | Multiplier Service running on port 8082...
```

5. Testing the Services

5.1. Test Service A (Addition)

curl -X POST http://localhost:8081/add -H "Content-Type: application/json" -d "{\"a\":5, \"b\":3}"

Expected Response:

{"result":8}

5.2. Test Service B (Multiplication)

curl -X POST http://localhost:8082/multiply -H "Content-Type: application/json" -d "{\"a\":5, \"b\":3}"

Expected Response:

{"result":15}

Task 2

1. Introduction

This project involves developing two microservices using Node.js, containerizing them using Docker, and orchestrating them with Docker Compose. The microservices perform basic arithmetic operations:

Service A: Addition (/add)

Service B: Multiplication (/multiply)

Each service runs independently in a Docker container, ensuring scalability and ease of deployment.

2. Methodology

The following steps were followed:

1. Developed two Node.js microservices using express.

2. Containerized each service using Docker.

3. Created a Docker Compose file to orchestrate the services.

4. Built and deployed the containers.

5. Tested the endpoints using curl.

3. Implementation

3.1. Service A: Addition Microservice

1. Create serviceA/package.json

```
{
    "name": "servicea",
    "version": "1.0.0",
    "description": "Addition service",
    "main": "index.js",
    "dependencies": {
      "express": "^4.18.2",
      "body-parser": "^1.20.2"
    }
  }
```

2. Create serviceA/index.js const

```
const express = require("express");
const bodyParser = require("body-parser");

const app = express();
app.use(bodyParser.json());

app.post("/add", (req, res) => {
  const { a, b } = req.body;
  if (typeof a !== "number" || typeof b !== "number") {
    return res.status(400).json({ error: "Invalid input. Please provide numbers."
});
  }
  res.json({ result: a + b });
});

const PORT = 8081;
app.listen(PORT, () => console.log(`ServiceA (Addition) running on port
${PORT}`));
```

3. Create serviceA/Dockerfile

```
FROM node:22

WORKDIR /app
COPY package.json .
RUN npm install
COPY . .

CMD ["node", "index.js"]
```

3.2. Service B: Multiplication Microservice

1. Create serviceB/package.json

```
{
    "name": "serviceb",
    "version": "1.0.0",
    "description": "Multiplication service",
    "main": "index.js",
    "dependencies": {
      "express": "^4.18.2",
      "body-parser": "^1.20.2"
    }
}
```

2. Create serviceB/index.js

```
const express = require("express");
const bodyParser = require("body-parser");

const app = express();
app.use(bodyParser.json());

app.post("/multiply", (req, res) => {
  const { a, b } = req.body;
  if (typeof a !== "number" || typeof b !== "number") {
    return res.status(400).json({ error: "Invalid input. Please provide numbers."
});
  }
  res.json({ result: a * b });
});

const PORT = 8082;
app.listen(PORT, () => console.log(`ServiceB (Multiplication) running on port
${PORT}`));
```

3.3. Docker Compose Configuration

Create docker-compose.yml

```
version: "3"
services:
  servicea:
    build: ./serviceA
    ports:
      - "8081:8081"

  serviceb:
    build: ./serviceB
    ports:
      - "8082:8082"
```

4. Deployment Steps

Step 1: Build and Start Containers

docker-compose up –build

```
2025/03/09 20:20:53 http2: server: error reading preface from client //./pipe/docker_engine: file has already been closed
[+] Building 91.5s (17/17) FINISHED
  docker:default
 => [servicea internal] load .dockerignore
           0.0s
 => => transferring context: 2B
           0.0s
 => [servicea internal] load build definition from Dockerfile
           0.0s
 => => transferring dockerfile: 143B
           0.0s
 => [servicea internal] load metadata for docker.io/library/node:22
           3.2s
 => [serviceb internal] load build definition from Dockerfile
           0.0s
 => => transferring dockerfile: 143B
           0.0s
 => [serviceb internal] load .dockerignore
           0.0s
 => => transferring context: 2B
           0.0s
 => [serviceb 1/5] FROM docker.io/library/node:22@sha256:f6b9c31ace05502dd98ef777aaa20464362435dcc5e312b0e213121dcf7d8b95
          82.9s
```

```
  0.2s
 => => writing image sha256:c292c8e40588ad1f33f9fc901149625eebc4d2afc56cbbef05ee952dc06d9af6
           0.0s
 => => naming to docker.io/library/nodejs-servicea
           0.0s
[+] Running 3/3
 ✓ Network nodejs_default        Created
           0.1s
 ✓ Container nodejs-serviceb-1   Created
           0.1s
 ✓ Container nodejs-servicea-1   Created
           0.1s
Attaching to servicea-1, serviceb-1
serviceb-1  | ServiceB (Multiplication) running on port 8082
servicea-1  | ServiceA (Addition) running on port 8081
```

5. Testing the Services

5.1. Test Service A (Addition)

curl -X POST http://localhost:8081/add -H "Content-Type: application/json" -d "{\"a\":5, \"b\":3}"

Expected Response:

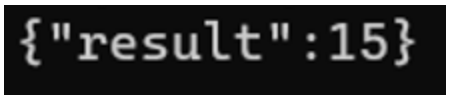
{"result":8}

5.2. Test Service B (Multiplication)

curl -X POST http://localhost:8082/multiply -H "Content-Type: application/json" -d "{\"a\":5, \"b\":3}"

Expected Response:

{"result":15}



Conclusion

This project successfully implemented two microservices using Node.js, deployed them in Docker containers, and managed them using Docker Compose. The services were tested using curl commands to verify their functionality.

Through this implementation, we demonstrated how to create, containerize, and orchestrate microservices efficiently using Docker. This approach can be extended to build scalable and distributed applications.