

# INTRO TO DEVOPS

Ivan Pancirov

University College Algebra, June 2024.

## Contents

<b>1. Abstract .....</b>	<b>3</b>
<b>2. Introduction.....</b>	<b>3</b>
<b>3. Explanation .....</b>	<b>4</b>
3.1. Cloning the project.....	4
3.2. Creating and pushing Docker images .....	4
3.3. Creating a network and deploying the application.....	6
3.4. Choosing container orchestration systems .....	8
3.5. Docker Swarm deployment.....	8
3.1. Kubernetes deployment .....	11
<b>4. Conclusion .....</b>	<b>17</b>
<b>5. References.....</b>	<b>18</b>

## 1. Abstract

This project focuses on the containerization and deployment of the TempConverter application using Docker and Kubernetes. The goal is to deploy this application using both a simple orchestrator, Docker Swarm, and a more complex one, Kubernetes, to evaluate their effectiveness. This comparison will highlight how each system manages the deployment, scaling, and ongoing management of containers, providing insights into their operational efficiency.

## 2. Introduction

The essence of this project is to explore the practical challenges and solutions associated with transforming a non-containerized web application into a scalable, containerized deployment. The TempConverter application, which was not originally designed for containerized environments, presents a typical scenario encountered in modern software deployment situations.

The project begins by defining the problem: the need to adapt and deploy an existing application in a manner that supports both scalability and ease of management using containerization technologies. Historically, such problems have been approached with various technologies, but Docker has emerged as a leading solution for containerization due to its simplicity and effectiveness. Additionally, orchestration systems like Docker Swarm and Kubernetes have become crucial in managing these containers, especially when scalability and resource optimization are paramount.

In this project, Docker is used to containerize the TempConverter application, preparing it for deployment in different orchestration environments. The first deployment will utilize Docker Swarm, known for its simplicity and less resource-intensive management capabilities. Following this, the project will employ Kubernetes, recognized for its robustness and suitability for larger, more complex deployments.

The comparative study of these two systems will document the deployment process, the challenges encountered, the scalability achieved, and the ease of administration provided by each system. This analysis will not only highlight the operational differences but also provide valuable insights into choosing the right orchestration tool based on specific project requirements.

Finally, the project will outline possible next steps, including recommendations for enhancing deployment strategies. These might involve incorporating advanced monitoring and management tools to further improve the operational efficiency of containerized applications, should the project be developed further, beyond the scope of this assignment.

### Important notes:

This project uses Docker Desktop on Windows 11 with WSL2 for managing Linux Docker images. Docker Swarm and Kubernetes are used for container orchestration.

All commands and scripts, unless otherwise specified, are written for and executed in Windows PowerShell. The Dockerfile is written in the Dockerfile format. The Docker Compose file and Kubernetes manifests are written in YAML.

## 3. Explanation

### 3.1. Cloning the project

First, the project needs to be cloned from the provided GitHub repository (1).

```
git clone https://github.com/jstanesic/tempconverter
```

*Figure 1 – Git clone command*

### 3.2. Creating and pushing Docker images

Next, a Dockerfile needs to be created, and placed in the root directory of the project (2).

The Dockerfile needs to comply with these requirements:

- Update all packages as part of the image build process
- Expose port 5000 TCP
- Install all required requirements
- Configure the correct command to start the flask application

```
FROM python:3.9-slim

WORKDIR /app

COPY . /app

RUN apt-get update && apt-get upgrade -y && apt-get install -y --no-install-
recommends \
    && rm -rf /var/lib/apt/lists/* \
    && pip install --no-cache-dir -U pip \
    && pip install --no-cache-dir -r requirements.txt

EXPOSE 5000

ENV FLASK_APP=app.py
ENV STUDENT="Ivan Pancirov"
ENV COLLEGE="Algebra University College"

CMD ["flask", "run", "--host=0.0.0.0"]
```

*Figure 2 – Dockerfile*

Now, a container image needs to be built using this Dockerfile (3).

```
docker build -t tempconverter .
```

*Figure 3 – Docker build command*

Once the container image has been successfully built, it needs to be pushed to a container registry and tagged as tempconverter:latest (4).

```
docker login
docker tag tempconverter:latest brownduck/tempconverter:latest
docker push brownduck/tempconverter:latest
```

Figure 4 – Docker login, tag and push image commands

To confirm that the image was successfully pushed to Dockerhub, the pushed images on Dockerhub are reviewed (5) (6).

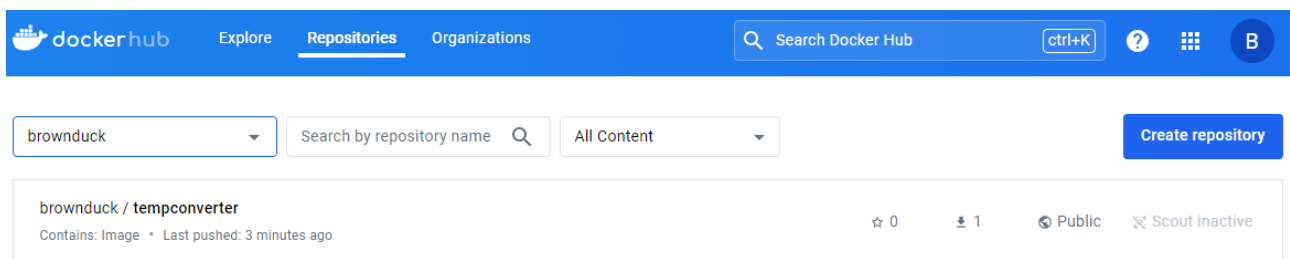


Figure 5 – Docker image on dockerhub

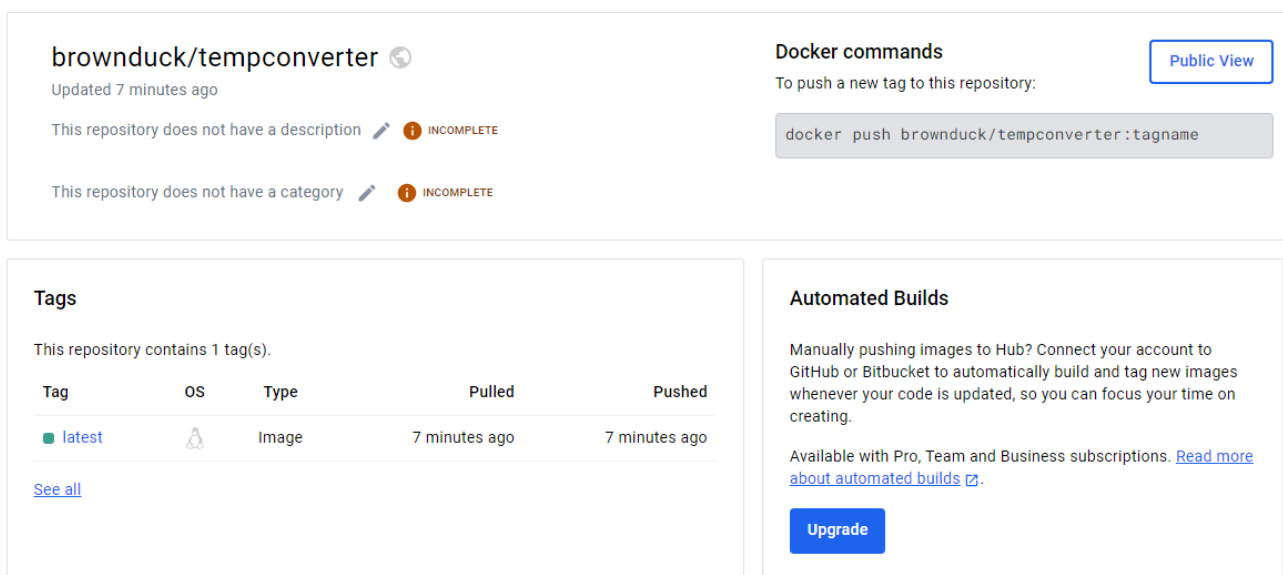


Figure 6 – Docker image details on Dockerhub

After this, the title in the index.html file needs to be changed (7).

```
<title>TempConverter</title>
```

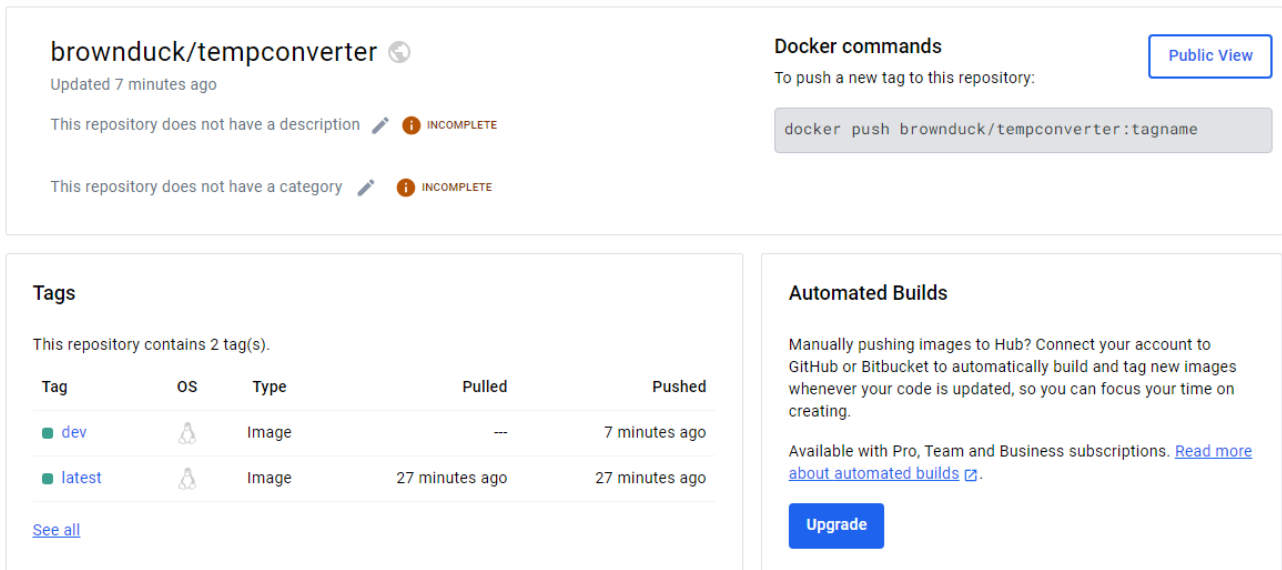
Figure 7 – Altered contents of html title

Then, a new container image needs to be created, tagged with tempconverter:dev and pushed to the container registry (8).

```
docker build -t brownduck/tempconverter:dev .
docker push brownduck/tempconverter:dev
```

Figure 8 – Docker build and push commands for the new image

It is now possible to confirm that this new image was successfully pushed to the container registry (9).



The screenshot shows the Docker Hub repository page for `brownduck/tempconverter`. The repository was updated 7 minutes ago. It notes that the repository does not have a description or category, both marked as 'INCOMPLETE'. The 'Tags' section shows two tags: `dev` (pushed 7 minutes ago) and `latest` (pushed 27 minutes ago). The 'Automated Builds' section provides information on connecting to GitHub or Bitbucket for automatic builds and includes an 'Upgrade' button.

Figure 9 – Updated docker image details on Dockerhub

### 3.3. Creating a network and deploying the application

To deploy the application locally, a network needs to be created to enable the application container to communicate with the required mysql container (10).

```
docker network create tempconverter_network
```

Figure 10 – Docker network creation command

The mysql container now needs to be started, specifying the network it will use for communication, and setting the required environment variables (11).

```
docker run --name mysql-db `
  --network tempconverter_network `
  -e MYSQL_ROOT_PASSWORD=root `
  -e MYSQL_USER=user `
  -e MYSQL_PASSWORD=pass `
  -e MYSQL_DATABASE=tempconverterdb `
  -d mysql:8
```

Figure 11 – Running the mysql container

After this container has been successfully started, the tempconverter container needs to be started. Again, the network is specified, and the required environment variables are set (12).

```
docker run --name tempconverter `
  --network tempconverter_network `
  -e DB_HOST=mysql-db `
  -e DB_USER=user `
  -e DB_PASS=pass `
  -e DB_NAME=tempconverterdb `
  -p 5000:5000 `
  -d brownduck/tempconverter:latest
```

*Figure 12 – Running the tempconverter container*

Having successfully started both container images, it is now possible to confirm the web application functions as expected (13).



# Celsius to Fahrenheit Converter

Hosted by student Ivan Pancirov attending Algebra University College.

Celsius

## Recent Conversions:

Date	Celsius	Fahrenheit	IP Address	User Agent
06/14/2024 03:49:50 PM	20.0	68.0	172.18.0.1	Mozilla/5.0

*Figure 13 – Deployed tempconverter application in use*

### 3.4. Choosing container orchestration systems

For this project, Docker Swarm was selected as the simpler container orchestration system due to its seamless integration with Docker, which offers an intuitive and straightforward method for managing containers. Its simplicity is ideal for smaller-scale applications or for those new to container orchestration [1] [2].

For more complex deployment needs, Kubernetes was chosen. It is recognized for its advanced capabilities and robust feature set, making it suitable for managing larger, more complex deployments. As the industry standard for container orchestration, Kubernetes supports high availability, scalability, and maintenance of containerized applications across multiple hosts, aligning with the project's requirements for a sophisticated orchestration solution [3].

### 3.5. Docker Swarm deployment

First, Docker Swarm needs to be initialized (14).

```
docker swarm init
```

*Figure 14 – Docker Swarm initialization command*

This command transforms the current Docker engine into a Swarm manager node.

The status of the Docker Swarm node is checked (15) (16).

```
docker node ls
```

*Figure 15 – Docker Swarm node status check command*

```
PS C:\Users\Ivan\Desktop\UD0Projekt-UnutarVM\ClonedApplication\tempconverter> docker node ls
ID                HOSTNAME          STATUS    AVAILABILITY  MANAGER STATUS  ENGINE VERSION
x694wi3gsgt6ec1vtrv7l42a6 *  docker-desktop  Ready     Active         Leader           26.1.4
```

*Figure 16 – Docker swarm node status*

In order to prepare this project for deployment using Docker Swarm, a Docker Compose configuration file first needs to be created and placed in the root directory of the project (17).

Since only one machine is used in this project, it is necessary to configure both services (mysql-db and tempconverter) to run on the manager node.

There is no need to set the environment variables STUDENT and COLLEGE in this configuration file, as they are already set in the Dockerfile used to build the tempconverter image used here and will be the same across all deployments.



```
version: '3.8'

services:
  mysql-db:
    image: mysql:8
    environment:
      MYSQL_ROOT_PASSWORD: root
      MYSQL_USER: user
      MYSQL_PASSWORD: pass
      MYSQL_DATABASE: tempconverterdb
    networks:
      - tempnet
    deploy:
      replicas: 1
      placement:
        constraints:
          - node.role == manager

  tempconverter:
    image: brownduck/tempconverter:latest
    ports:
      - "80:5000"
    environment:
      DB_HOST: mysql-db
      DB_USER: user
      DB_PASS: pass
      DB_NAME: tempconverterdb
    networks:
      - tempnet
    depends_on:
      - mysql-db
    deploy:
      replicas: 2
      placement:
        constraints:
          - node.role == manager

networks:
  tempnet:
```

*Figure 17 – Docker Compose file*

Next, the stack needs to be deployed (18).

```
docker stack deploy -c docker-compose.yml tempstack --detach=false
```

*Figure 18 – Docker Swarm stack deployment*

This command tells Docker to deploy the services, networks, and other resources described in the Docker Compose file as a stack in Swarm mode.

The deployment of the stack is checked (19) (20).

```
docker stack ls
```

Figure 19 – Docker Swarm stack check command

```
PS C:\Users\Ivan\Desktop\UDOProjekt-UnutarVM\ClonedApplication\tempconverter> docker stack ls
NAME          SERVICES
tempstack     2
```

Figure 20 – Docker Swarm stack status

Next, the deployment of the services themselves is checked (21) (22).

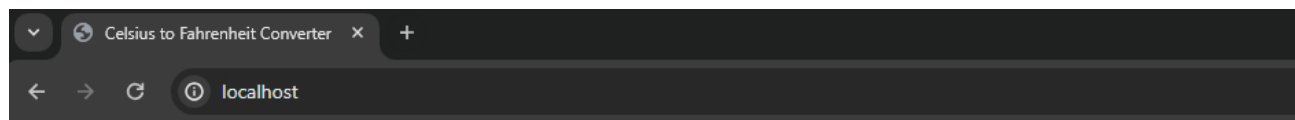
```
docker service ls
```

Figure 21 – Docker Swarm service check command

```
PS C:\Users\Ivan\Desktop\UDOProjekt-UnutarVM\ClonedApplication\tempconverter> docker service ls
ID                NAME                MODE                REPLICAS    IMAGE                PORTS
3j9jrceb2r13     tempstack_mysql-db  replicated          1/1         mysql:8
iwpmuiawpy16     tempstack_tempconverter  replicated          2/2         brownduck/tempconverter:latest  *:80->5000/tcp
```

Figure 22 – Docker Swarm service status

Finally, the web application itself is accessed at <http://localhost:80/>, to ensure it is functioning as expected (23).



# Celsius to Fahrenheit Converter

Hosted by student Ivan Pancirov attending Algebra University College.

Celsius

## Recent Conversions:

Date	Celsius	Fahrenheit	IP Address	User Agent
06/14/2024 06:27:14 PM	30.0	86.0	10.0.0.2	Mozilla/5.0

Figure 23 – Deployed tempconverter application using Docker Swarm in use

The application container can be scaled to 3 replicas using the scale command (24) (25).

```
docker service scale tempstack_tempconverter=3
```

*Figure 24 – Scale tempconverter to 3 replicas using Docker Swarm command*

```
PS C:\Users\Ivan\Desktop\UDOProjekt-UnutarVM\ClonedApplication\tempconverter> docker service scale tempstack_tempconverter=3
tempstack_tempconverter scaled to 3
overall progress: 3 out of 3 tasks
1/3: running [=====>]
2/3: running [=====>]
3/3: running [=====>]
verify: Service tempstack_tempconverter converged
```

*Figure 25 – Scale tempconverter to 3 replicas using Docker Swarm command output*

The scaling of the tempconverter container to 3 replicas is confirmed as before (26).

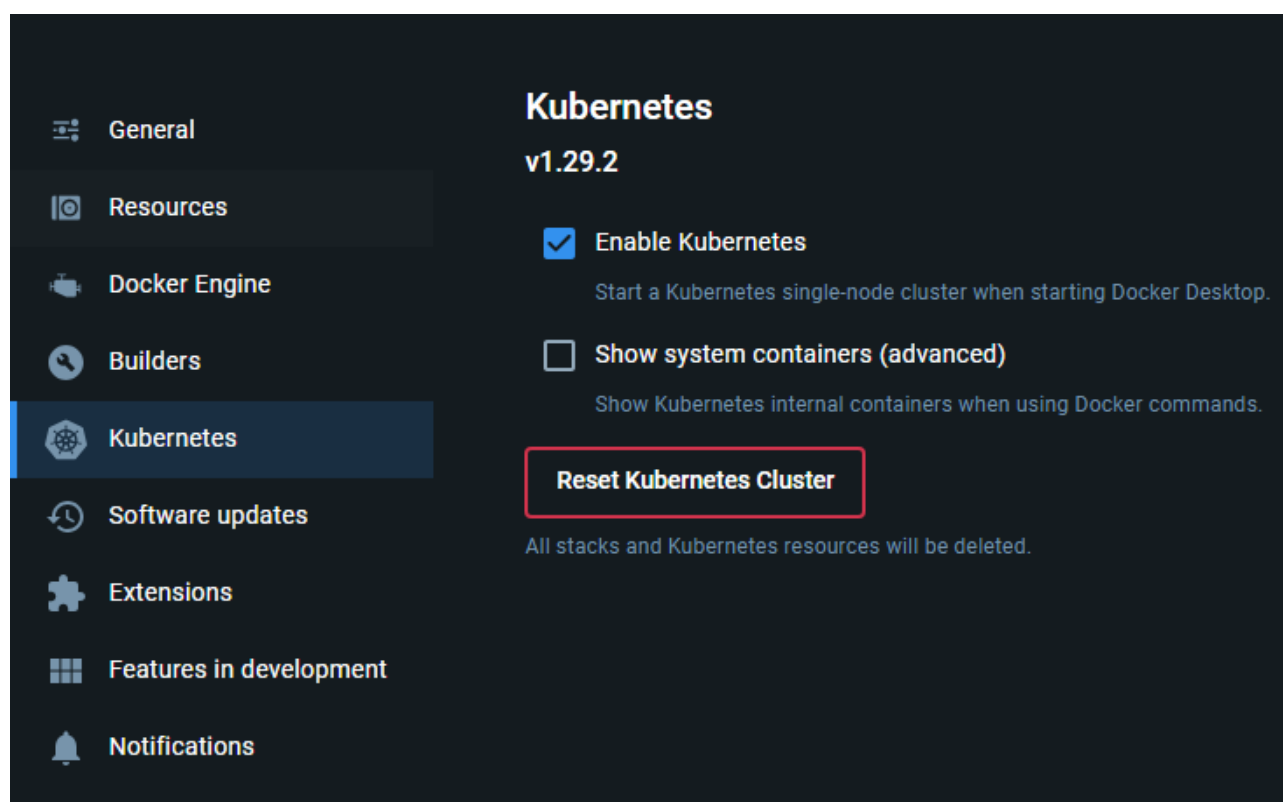
```
PS C:\Users\Ivan\Desktop\UDOProjekt-UnutarVM\ClonedApplication\tempconverter> docker service ls
```

ID	NAME	MODE	REPLICAS	IMAGE	PORTS
3j9jrceb2r13	tempstack_mysql-db	replicated	1/1	mysql:8	
iwpmuiawpy16	tempstack_tempconverter	replicated	3/3	brownduck/tempconverter:latest	*:80->5000/tcp

*Figure 26 – Docker Swarm service status after scaling tempconverter to 3 replicas*

### 3.1. Kubernetes deployment

This project uses Docker Desktop, therefore enabling and initializing kubernetes is done from the settings menu in the Docker Desktop GUI (27).



*Figure 27 – Docker Desktop Kubernetes options*

To ensure Kubernetes is running, the cluster-info command is used (28) (29).

```
kubectl cluster-info
```

Figure 28 – Kubernetes cluster status command

```
PS C:\Users\Ivan\Desktop\UDOProjekt-UnutarVM\ClonedApplication\tempconverter> kubectl cluster-info
Kubernetes control plane is running at https://kubernetes.docker.internal:6443
CoreDNS is running at https://kubernetes.docker.internal:6443/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy
To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
```

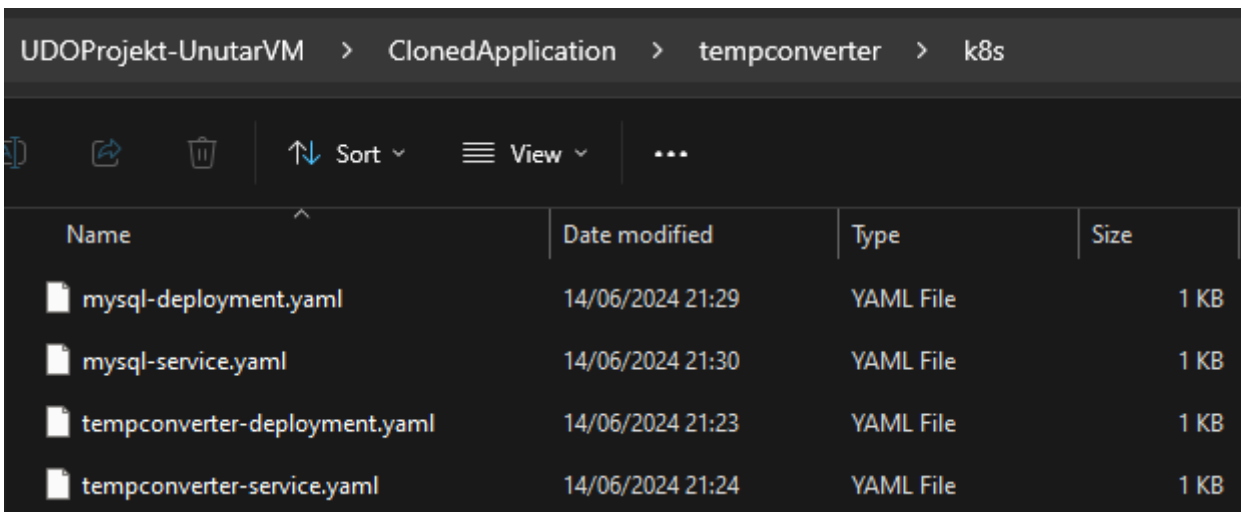
Figure 29 – Kubernetes cluster status

In order to prepare the project for deployment using Kubernetes, several configuration files, known as Kubernetes manifests, need to be created. These files are organized into a newly created subdirectory named k8s located in the project's root directory (30).

The configuration for Kubernetes deployment includes the following manifest files:

- tempconverter-deployment (31)
- tempconverter-service (32)
- mysql-deployment (33)
- mysql-service (34)

The deployment files focus on how Kubernetes orchestrates the application's pods, including their lifecycle, scalability, and resource requirements. In contrast, the service files ensure that these pods are network-accessible, defining how internal and external communications are handled.



Name	Date modified	Type	Size
mysql-deployment.yaml	14/06/2024 21:29	YAML File	1 KB
mysql-service.yaml	14/06/2024 21:30	YAML File	1 KB
tempconverter-deployment.yaml	14/06/2024 21:23	YAML File	1 KB
tempconverter-service.yaml	14/06/2024 21:24	YAML File	1 KB

Figure 30 – Kubernetes manifest directory and file structure

It is important to note that, as in the case of Docker Swarm, only one machine is used for Kubernetes deployment in this project. This means that all pods and services necessarily have to be run on one physical node. For this reason, no affinity or anti-affinity rules are included in the Kubernetes manifests (specifically the tempconverter-deployment manifest).

As in the case of Docker Swarm, there is no need to explicitly set the environment variables STUDENT and COLLEGE, as they are already set in the Dockerfile used to build the tempconverter image.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: tempconverter
  labels:
    app: tempconverter
spec:
  replicas: 2
  selector:
    matchLabels:
      app: tempconverter
  template:
    metadata:
      labels:
        app: tempconverter
    spec:
      containers:
        - name: tempconverter
          image: brownduck/tempconverter:latest
          ports:
            - containerPort: 5000
          env:
            - name: DB_HOST
              value: mysql-service
            - name: DB_USER
              value: "user"
            - name: DB_PASS
              value: "pass"
            - name: DB_NAME
              value: "tempconverterdb"
```

*Figure 31 – tempconverter-deployment Kubernetes manifest*

```
apiVersion: v1
kind: Service
metadata:
  name: tempconverter-service
spec:
  type: LoadBalancer
  ports:
    - port: 80
      targetPort: 5000
      protocol: TCP
  selector:
    app: tempconverter
```

*Figure 32 – tempconverter-service Kubernetes manifest*

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mysql
  labels:
    app: mysql
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mysql
  template:
    metadata:
      labels:
        app: mysql
    spec:
      containers:
        - name: mysql
          image: mysql:8
          env:
            - name: MYSQL_ROOT_PASSWORD
              value: root
            - name: MYSQL_USER
              value: user
            - name: MYSQL_PASSWORD
              value: pass
            - name: MYSQL_DATABASE
              value: tempconverterdb
          ports:
            - containerPort: 3306
```

*Figure 33 – mysql-deployment Kubernetes manifest*

```
apiVersion: v1
kind: Service
metadata:
  name: mysql-service
spec:
  ports:
    - port: 3306
      targetPort: 3306
      protocol: TCP
  selector:
    app: mysql
```

*Figure 34 – mysql-service Kubernetes manifest*

After navigating to the directory containing the Kubernetes manifests, the following commands are run in order, to ensure dependencies are correctly managed (35) (36).

```
kubectl apply -f mysql-deployment.yaml
kubectl apply -f mysql-service.yaml
```

*Figure 35 – Kubernetes deployment commands for mysql*

```
kubectl apply -f tempconverter-deployment.yaml
kubectl apply -f tempconverter-service.yaml
```

*Figure 36 – Kubernetes deployment commands for tempconverter*

Successful deployment of pods and services is verified (37) (38) (39).

```
kubectl get pods
kubectl get services
```

*Figure 37 – Kubernetes deployment verification commands*

```
PS C:\Users\Ivan\Desktop\UD0Projekt-UnutarVM\ClonedApplication\tempconverter\k8s> kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
mysql-6cc9b5f75d-5sz8d             1/1     Running   0           99s
tempconverter-7b66d8b6b8-gq9l8     1/1     Running   0           22s
tempconverter-7b66d8b6b8-mlkjg     1/1     Running   0           22s
```

*Figure 38 – Kubernetes pods*

```
PS C:\Users\Ivan\Desktop\UD0Projekt-UnutarVM\ClonedApplication\tempconverter\k8s> kubectl get services
NAME            TYPE        CLUSTER-IP    EXTERNAL-IP  PORT(S)          AGE
kubernetes      ClusterIP   10.96.0.1     <none>       443/TCP          21h
mysql-service   ClusterIP   10.109.177.234 <none>       3306/TCP         93s
tempconverter-service LoadBalancer 10.104.137.215 localhost     80:31821/TCP    8s
```

*Figure 39 – Kubernetes services*

The details on how to access the application, or more specifically the tempconverter-service are obtained (40) (41).

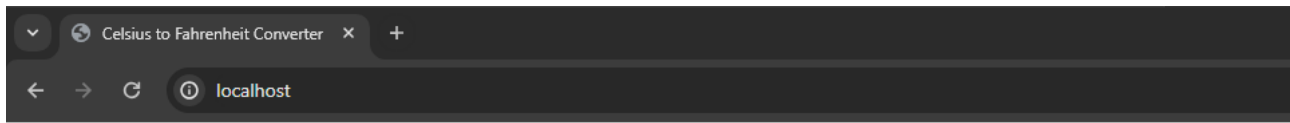
```
kubectl get service tempconverter-service
```

*Figure 40 – Kubernetes service information command*

```
PS C:\Users\Ivan\Desktop\UD0Projekt-UnutarVM\ClonedApplication\tempconverter\k8s> kubectl get service tempconverter-service
NAME            TYPE        CLUSTER-IP    EXTERNAL-IP  PORT(S)          AGE
tempconverter-service LoadBalancer 10.104.137.215 localhost     80:31821/TCP    17m
```

*Figure 41 – Kubernetes service information*

The web application is accessed at <http://localhost:80/>, to ensure that it is functioning as expected (42).



# Celsius to Fahrenheit Converter

Hosted by student Ivan Pancirov attending Algebra University College.

Celsius

## Recent Conversions:

Date	Celsius	Fahrenheit	IP Address	User Agent
06/14/2024 08:40:58 PM	10.0	50.0	192.168.65.3	Mozilla/5.0

Figure 42 – Deployed tempconverter application using Kubernetes

The application can be scaled, increasing the number of tempconverter replicas, using the following command (43) (44).

```
kubectl scale deployment/tempconverter --replicas=3
```

Figure 43 – Scale tempconverter to 3 replicas using Kubernetes command

```
PS C:\Users\Ivan\Desktop\UD0Projekt-UnutarVM\ClonedApplication\tempconverter\k8s> kubectl scale deployment/tempconverter --replicas=3
deployment.apps/tempconverter scaled
```

Figure 44 – Scale tempconverter to 3 replicas using Kubernetes command output

After scaling, it is important to verify that the new replicas are up and running without issues (45) (46).

```
kubectl get deployment tempconverter
```

Figure 45 – Status of tempconverter Kubernetes deployment command

```
PS C:\Users\Ivan\Desktop\UD0Projekt-UnutarVM\ClonedApplication\tempconverter\k8s> kubectl get deployment tempconverter
NAME          READY   UP-TO-DATE   AVAILABLE   AGE
tempconverter  3/3     3             3           42m
```

Figure 46 – Status of tempconverter Kubernetes deployment after scaling



## 4. Conclusion

This project embarked on the task of containerizing and deploying the TempConverter application using two prominent container orchestration systems: Docker Swarm and Kubernetes. Throughout this endeavor, the project successfully demonstrated the transition of a traditional application into a robust, scalable, containerized environment.

Using Docker Desktop on Windows 11 with WSL2, the project effectively utilized Docker to build and manage Linux Docker images, showcasing a seamless integration between development environments and production-like settings.

Docker Swarm was employed first, highlighting its simplicity and ease of use, which proved to be highly effective for straightforward deployments and smaller-scale applications. This phase underscored Docker Swarm's utility in environments where resource efficiency and straightforward management are prioritized.

Subsequently, the project utilized Kubernetes, widely recognized for its robust orchestration capabilities. The deployment in Kubernetes demonstrated its functionality in managing more complex deployments compared to Docker Swarm. This phase showcased Kubernetes' basic features, such as load balancing and manual scaling, which are essential for maintaining application availability and effectively handling increased loads, albeit on a modest scale.

The comparative analysis of Docker Swarm and Kubernetes provided valuable insights into their operational efficiencies, scalability, and ease of management. Docker Swarm was noted for its user-friendliness and lower resource demands, making it ideal for developers new to container orchestration or for projects with limited scalability requirements. On the other hand, Kubernetes, despite its steeper learning curve, offered unparalleled control and scalability, making it suitable for larger applications and enterprises requiring comprehensive management features [4].

## 5. References

- [1] Docker Documentation. Available on: <https://docs.docker.com/> (Accessed: 2024-06-14)
- [2] Docker Swarm Documentation. Available on: <https://docs.docker.com/engine/swarm/> (Accessed: 2024-06-14).
- [3] Kubernetes Documentation. Available on: <https://kubernetes.io/docs/home/> (Accessed: 2024-06-14).
- [4] What is Container Orchestration?, Red Hat. Available on: <https://www.redhat.com/en/topics/containers/what-is-container-orchestration> (Accessed: 2024-06-14)