Three-body Problem
By Runge-kutta Solver Infrastructure
(Earth, Sun and Jupiter)
(Project 1)

Aleksandar Mitic (2035177)
Dominik Wirsig (2020067)
Ravishankar Selvaraj (2036915)

Submitted to

Dr. Torsten Harenberg

Dr. Marisa Sandhoff

Virtualization 2 | Project Phase

TABLE OF CONTENTS

1. INTRODUCTION

This project gives an overview of the development of a simulation of the three-body problem using the Runge-Kutta method and the subsequent projection of our graphical representations (diagrams and animation) onto the front-end website. This is followed by an explanation of traffic routing and SSL certificates for encrypting the web pages.

1.1 Methodology

We created a Dockerfile based on a Python image, since we chose our main script in the Python language. It solves Three Body problem via Runge-Kutta method and stores the results on the local Volume. Then we created a Docker-compose file that runs three microservices. The first two microservices use the same volume and the third uses a separate volume.

The first microservice (rk_method) solves the three-body problem, the second microservice (webserver) communicates with both the first and third microservices to project our graphical results onto the website, and the third microservice (traefik) is used for routing and encrypting the website.

1.2 Alternative Method

The alternative method is the same as this method, but with small improvements. In the Dockerfile container we can use a server and a client so that the user can give input for the three-body problem. We can use Python sockets for this. Instead of storing our results on a local Volume, we can use Flask and SQLAlchemy to store our results in a database and publish them to a separate website, from which we then pull our results to the Frontend (main website).

If we choose this method, we will need to create another microservice in the docker-compose file for the database (like MySQL, MariaDB, MongoDB, etc..,) to store the values. In addition, we also need to implement reverse proxy (as the frontend communicates with the backend Docker container via user input) and load balancing in traefik microservice. We can buy SSL certificates from providers (like Cloudflare) to encrypt our frontend.

1.3 Why did we choose our method?

We specifically looked at Earth, Sun and Jupiter for the three-body problem. This is because we have the inputs (mass, time, distance) of all three bodies from the official NASA website. Users are not allowed to enter inputs in the frontend so we don't have to create a separate database, which takes up a lot of space. Users are only allowed to view the graphical results of our simulation, which already contains preset configurations.

The reasons why we did not opt for the alternative method

- We don't have much knowledge of Python to work with Flask and SQLAlchemy.
- Saving images in the database and using them later is not as easy as it seems.
- We don’t need to use the Python sockets (for server and client) because we already have inputs.

2. PHASE I

This chapter gives an overview of what is a docker and a container and how to build and run a Dockerfile. We briefly explain the problem we had in our first phase. We did not solve the Runge-Kutta method for the three-body problem and wrote the Python code for it. We have taken this from a research paper and a GitHub account given in the reference

2.1 Docker, Container and Image

Image: An Image is the Application we want to run

Container: A Container is an instance of that image running as a process. You can have many containers running off the same image.

Docker: Docker can build images automatically by reading the instructions from a Dockerfile

Dockerfile: A Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image

2.2 Dockerfile

We created our Dockerfile using 5 major commands.

FROM: initializes a new build stage and sets the base image for subsequent instructions (we choose python image)

WORKDIR: setting the working directory

COPY: Copying requirements.txt and Rkmethod.py scripts from local volume

RUN: installing the Python dependencies (libraries)

```
1   FROM python:latest
2
3   WORKDIR /usr/src/app/
4
5   COPY RKmethod.py ./
6   COPY requirements.txt ./
7
8   RUN pip install -r requirements.txt
9   RUN apt update &&apt -y install ffmpeg
10
11  CMD ["python" ,"RKmethod.py"]
```

CMD: Set the default command for the container to run python program

2.3 How to build and run a Docker container?

The command used for building the Docker container - **docker build -t (dockername)** .

for example, docker build -t rkmethod .

The command used for running the Docker container - **docker run -it (dockername)**

for example, docker run -it rkmethod

2.4 Problems we had in PHASE I

- ✓ Most of our problems at this stage relate to Python programming, such as finding the main program that solves the three-body problem specifically using the Runge-Kutta method. Then we got a working Python code from the GitHub website, which is quoted in the reference.
- ✓ While running Docker, we were confronted with many error messages regarding the Python libraries, which we then eliminate by adding a specific library in the Dockerfile (requirements.txt).
- ✓ Saving animation as video (mp4 format) requires a special command that we have added to the Dockerfile - **RUN apt update &&apt -y install ffmpeg**

3.PHASE II

This chapter gives an overview of the docker-compose file and how to build a docker via docker-compose. For this project we needed more than one container to achieve the final result. To run a multi-container in parallel, docker-compose is used

3.1 Docker-compose

Compose is a tool for defining and running multi-container Docker applications. With Compose, you use a YAML file to configure your application's services. Then, with a single command, you create and start all the services from your configuration.

The docker-compose file always starts by specifying the version and then the services (in which we have multi-containers)

Why it is used?

- To configure relationships between containers
- It saves our docker container run settings in easy-to-read file
- It creates one-liner developer environment startups

3.2 Microservices

In the docker-compose file, every application is broken into several smaller services each of which is often run in its own separate container

```
version: "3.7"

services:
  rk_method:
    image: three_body_problem
    build: ./Container
    volumes:
      - ./Container:/usr/src/app

  webserver:
    image: nginx
    restart: always
    volumes:
      - ./Container:/usr/share/nginx/html
    depends_on:
      - rk_method
```

We created our first microservice with previous Dockerfile (which is based on solving 3-body problem). We used the image as three_body_problem and created and executed the container with the build command. In volumes, we have specified the local directory where the Dockerfile is stored.

Then we created our second microservice called Webserver, which is used exactly to create a web server. This is the main microservice which helps you to project your results in website as localhost. For this container, we used nginx (engine-x) image because its an open-source reverse proxy server for HTTP, HTTPS as well as load balancer and a web server. The main advantage is low memory usage. We used the same volume as for the first container. Depends on command is used to make this container dependent on other containers (here it depends on first container).

3.3 Problems we had in PHASE II

- ✓ With the first container, there were problems because we tried to save our results in another directory. But in the end, we used the same Volume(directory) where the main scripts are located, rather than a different volume.
- ✓ While creating webserver, we had many discussions and suggestions about which image to use, but we finally decided on nginx. Because when we were looking for a web server image, most Docker-Compose experts suggested this (especially in Quora website).

  (We had problems with pulling images from volumes, which we discuss in next phase)

4.PHASE III

This chapter gives an overview of creating a frontend website with docker-compose and Traefik routing for this frontend. To create the website, we tried basic web design with HTML to project our plots and animations.

4.1 Frontend website

In the second container, we have an HTML page that is to graphically display our results on the website. For this we have created three HTML pages: Index, Plots and Animations, and we added some reference websites to our index page.

4.2 Traefik

Traefik is an open-source Edge Router that makes publishing your services a fun and easy experience. It receives requests on behalf of your system and finds out which component are responsible for handling them

- Traefik is used for routing, load balancing, proxy and reverse proxy, etc...
- Here we used Traefik for Routing and SSL encrypt our webserver service.

We created our third microservice with the Traefik image and used EntryPoints for routing.

EntryPoints: It defines the port which will receive the packets and whether to listen for TCP or UDP

```yaml
webserver:
  image: nginx
  restart: always
  volumes:
    - ./Container:/usr/share/nginx/html
  depends_on:
    - rk_method
    - traefik
  labels:
    - "traefik.enable=true"
    - "traefik.http.routers.nginxrouter.entrypoints=web,websecure"
    - "traefik.http.routers.nginxrouter.rule=Host(`localhost`) || Host(`compass27.physik.uni-wuppertal.de`)"
    - "traefik.http.services.nginxservice.loadbalancer.server.port=80"
    - "traefik.http.routers.nginxrouter.tls=true"

traefik:
  image: "traefik:v2.5"
  container_name: "traefik"
  ports:
    - 8990:8990
    - 443:443
    - 8080:8080
  volumes:
    - /etc/localtime:/etc/localtime:ro
    - /var/run/docker.sock:/var/run/docker.sock:ro
    - ./traefik/data/traefik.yml:/traefik.yml:ro
```

```yaml
global:
  checkNewVersion: true
  sendAnonymousUsage: false  # true by default
# Enable API and Dashboard
api:
  dashboard: true  # true by default
  insecure: true  # Don't do this in production!
# Entry Points configuration
entryPoints:
  web:
    address: :8990
    # (Optional) Redirect to HTTPS
    http:
      redirections:
        entryPoint:
          to: websecure
          scheme: https
  websecure:
    address: :443
providers:
  docker:
    exposedByDefault: false  # Default is true
```

EntryPoint :8080 Routes to dashboard (you must protect this with a password or disable it)
EntryPoint :443 Routes to Secured Website (HTTPS)
EntryPoint :8990 Routes to Frontend (which will redirect to secured EntryPoint 443)

To secure the frontend website, we need to put labels in the second container. The second container depends on both traefik (third container) and rk_method (first container).

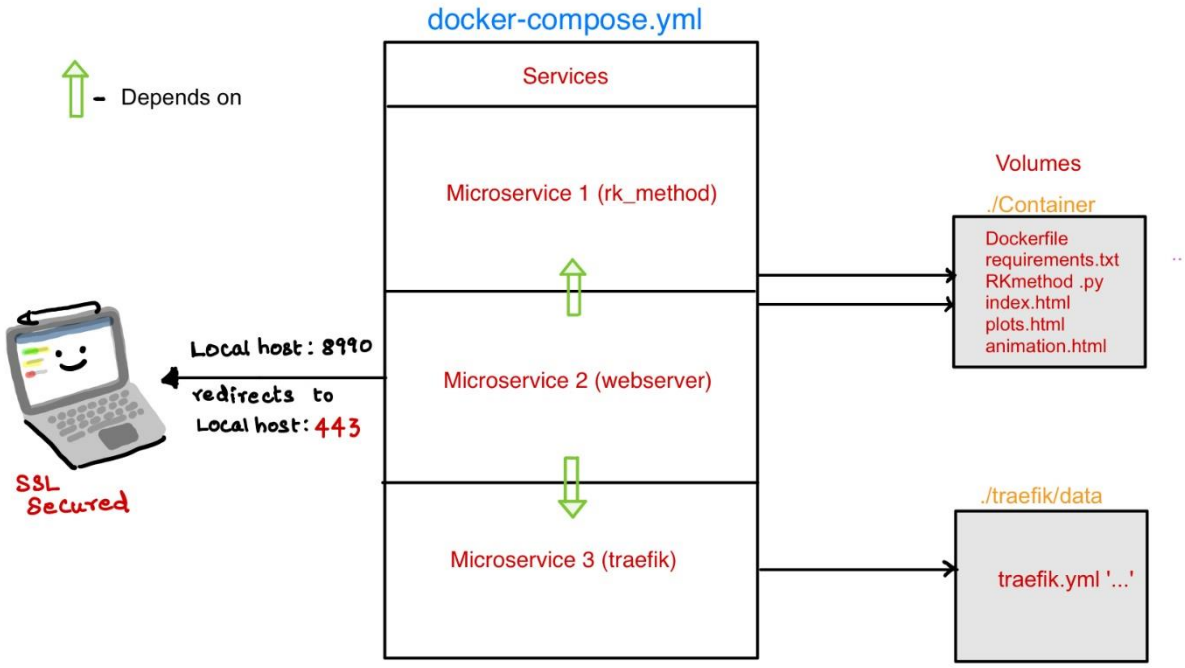
4.3 Problems we had in PHASE III

- ✓ Our plan so far is to store the plots in a separate database using Flask and SQLAlchemy and pull them from there directly to the frontend website. We have tried a lot, but we have decided on a simple solution to store the plots in a local volume and pull it out from there
- ✓ For a better understanding of the traefik routing and redirecting commands, we have created a separate traefik.yml file

5.PHASE IV

This chapter gives an overview of our whole project and how to run a docker-compose file. In the results we briefly explained about SSL Certificates and how frontend website works.

5.1 Graphical Representation of our Project



5.2 Executing docker-compose

Using the command docker-compose up -d to run the docker-compose file in the terminal

After Executing docker-compose file, the first container solves the three-body problem and stores the plots in the local volume, then the second container is started, which requests the output plots from the first container and projects them to the frontend website via the traefik routing that belongs to the third container.

Use the command docker-compose down to stop and remove the docker.

5.3 Results

Entering EntryPoint localhost:8990 in browser it redirects to the secured frontend website with the EntryPoint localhost:443

A warning message is displayed on this website: "Your connection is not private". If you click on the "Advanced" button located below this message, "proceed to localhost (unsafe)" is displayed.

When you click on it, you will be redirected to our frontend website. You can check the SSL certificate, but it shows that our certificate is not valid because we have not purchased SSL certificates from providers (like Cloudflare).

Finally, by clicking on plots you can view our various plots and by clicking on animation you can view our animation video, which represents three-body simulation between Earth, Sun and Jupiter.

6.REFERENCE

1. Earth Fact sheet: https://nssdc.gsfc.nasa.gov/planetary/factsheet/earthfact.html

2. Jupiter fact sheet: https://nssdc.gsfc.nasa.gov/planetary/factsheet/jupiterfact.html

3. Sun fact sheet: https://nssdc.gsfc.nasa.gov/planetary/factsheet/sunfact.html

4. The Restricted Three-Body Problem: Earth, Jupiter, Sun- Karla Boucher (March 2004) https://phas.ubc.ca/~berciu/TEACHING/PHYS349/karla.pdf

5. Three body problem solved by RK method (python script) https://github.com/zaman13/Three-Body-Problem-Gravitational-System/tree/master/Python%20script

6. Three Body Problem by Runge-Kutta solver Infrastructure (Our whole project code including presentation) https://github.com/RavishankarSelvaraj/Virtualization_2_Project

7. Traefik routing https://doc.traefik.io/traefik/