



Software framework for mobile
machine orchestration

DIY - Deployment of Autonomous Driving Applications

Dr. Carlos Viol Barbosa
Vehicle Control and Sensor Systems
Fraunhofer-Institute for Transportation and Infrastructure Systems IVI
Dresden, Germany



Logistic centers



Smart farms



Ports



Factories



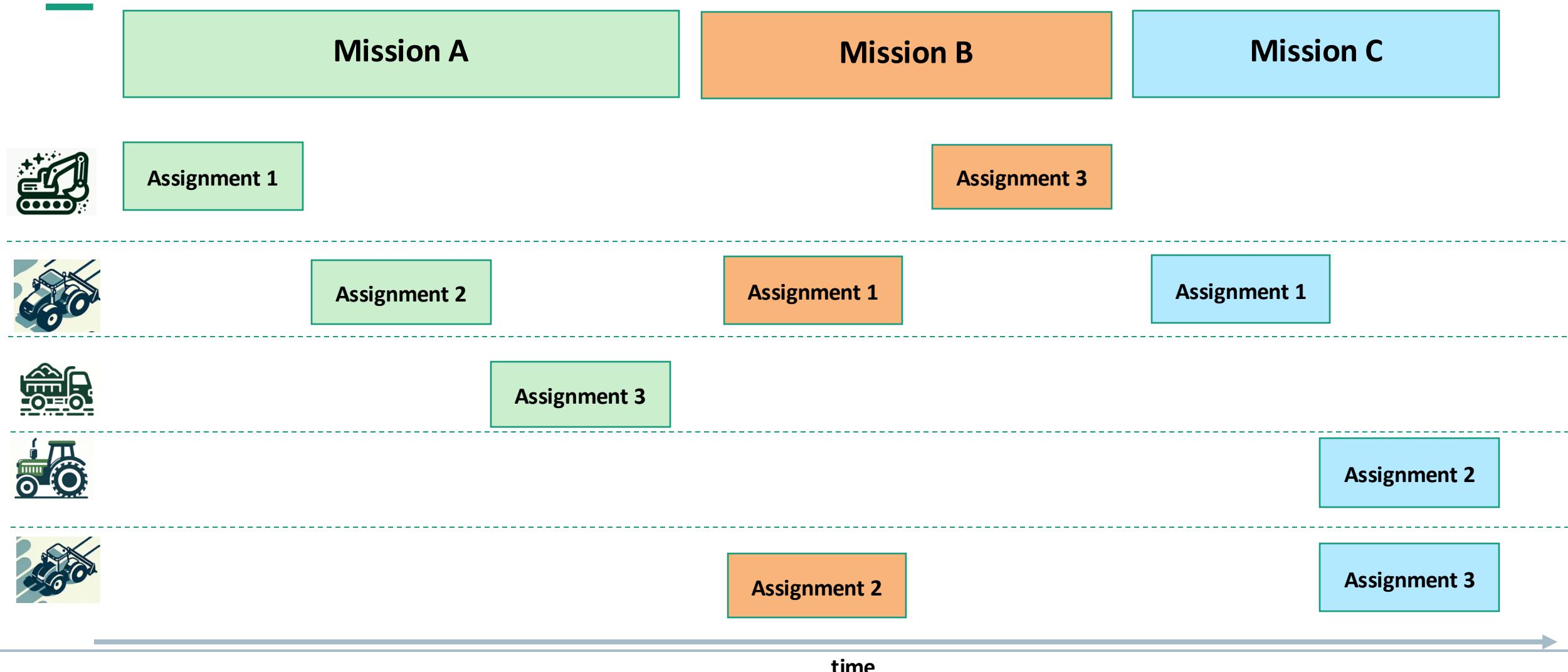
Warehouses



Mowers



helyOS solves the Orchestration of Assignments for You



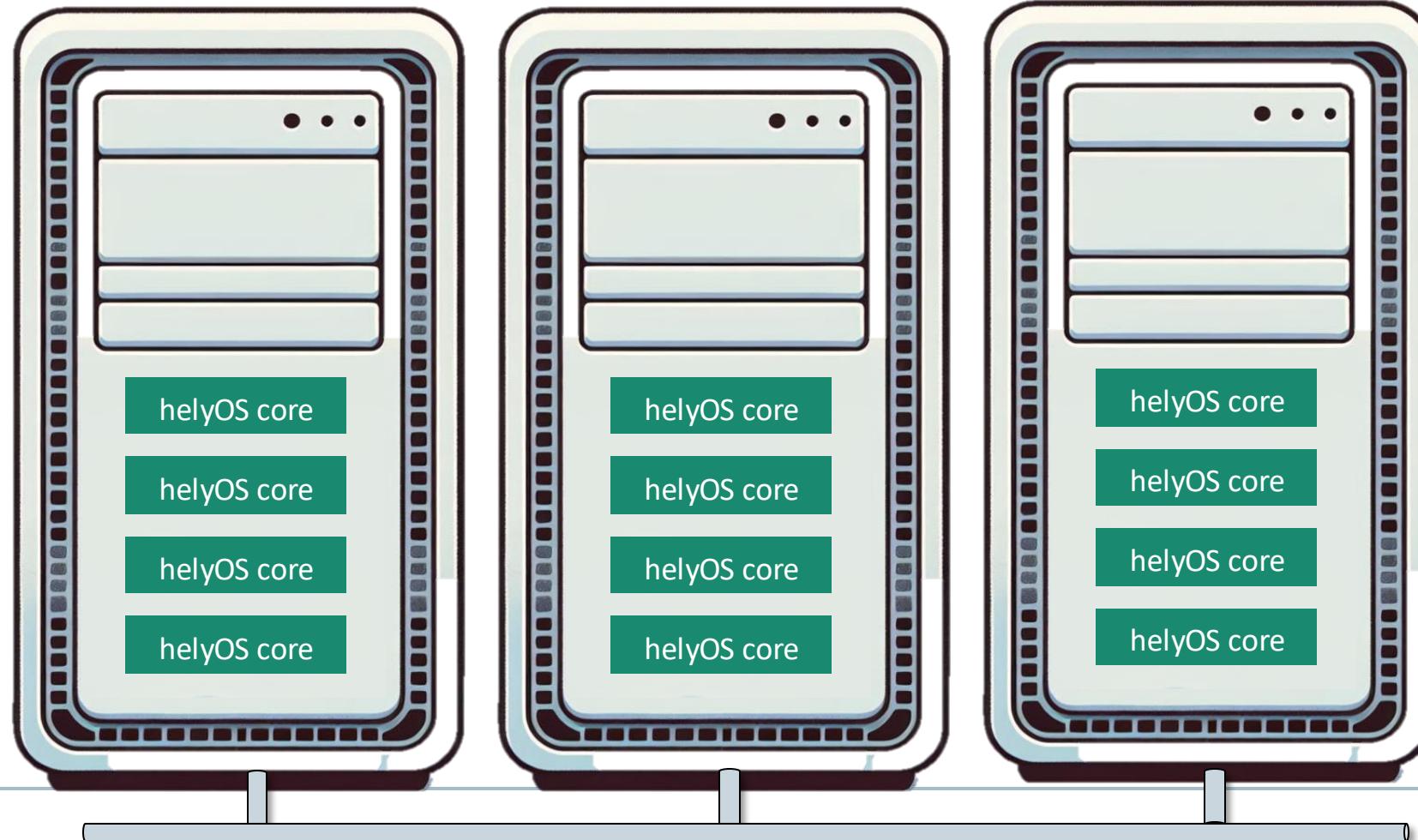
helyOS is Designed for Commercial Use

Vertically and Horizontally Scalable (version > 2.2.0)

helyos-server.com

workers

workers



Resource Optimization

Cost savings.

Reliability

Allow management tools like Kubernetes.

Future-Proofing

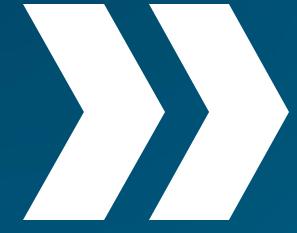
Growth with seamless scaling.

- **helyOS Framework Components**
- **Deploy Demo Applications**
- **Advancing Demo Applications Towards Real-World Scenarios.**

<https://github.com/FraunhoferIVI/diy-helyos-deployment>

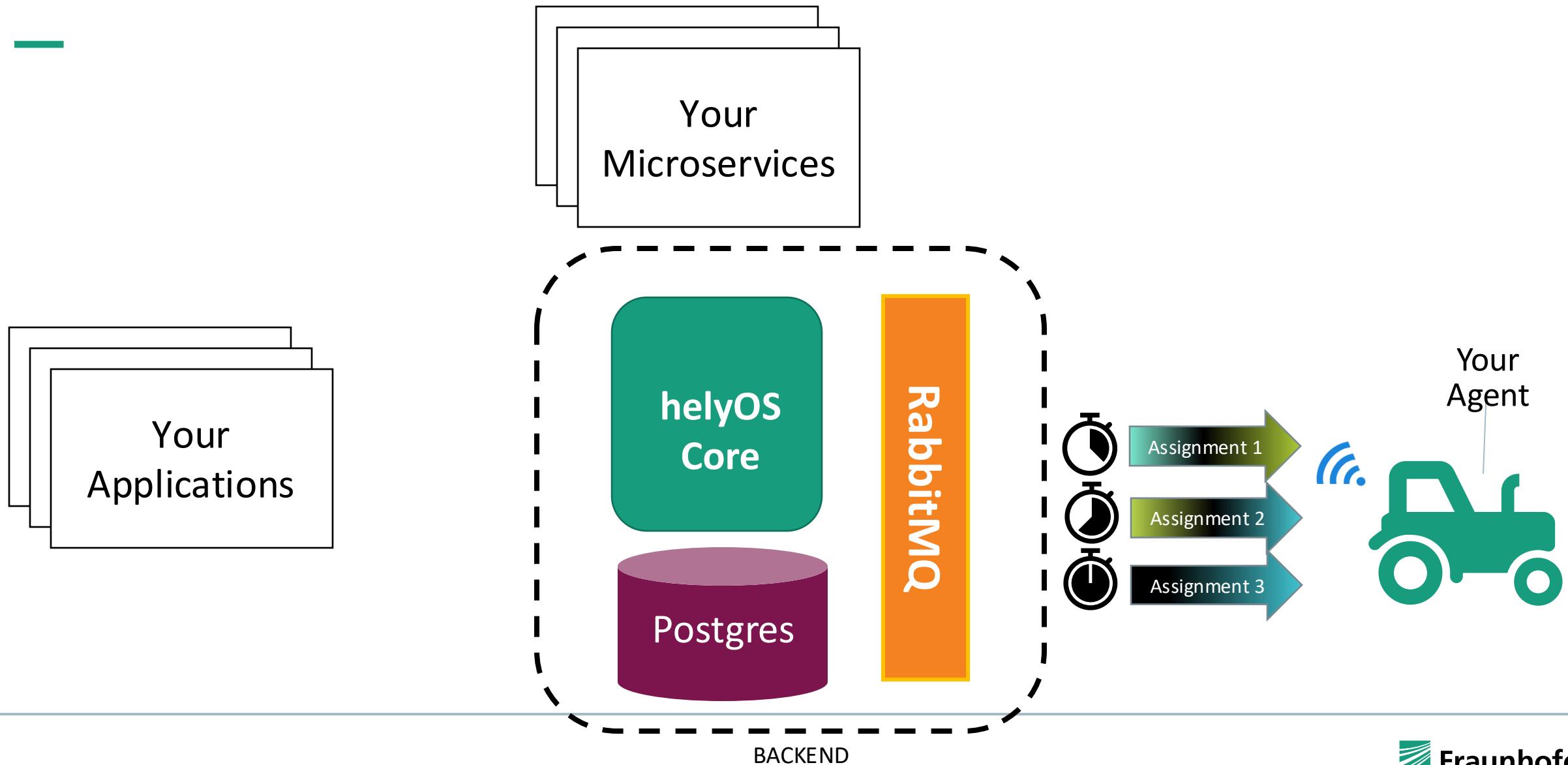
**Can't memorize everything?
No worries, no need to.
These slides will be your cheat-sheet later!**

<https://github.com/FraunhoferIVI/diy-helyos-deployment>



helyOS Framework Components

The helyOS framework components



The helyOS framework components



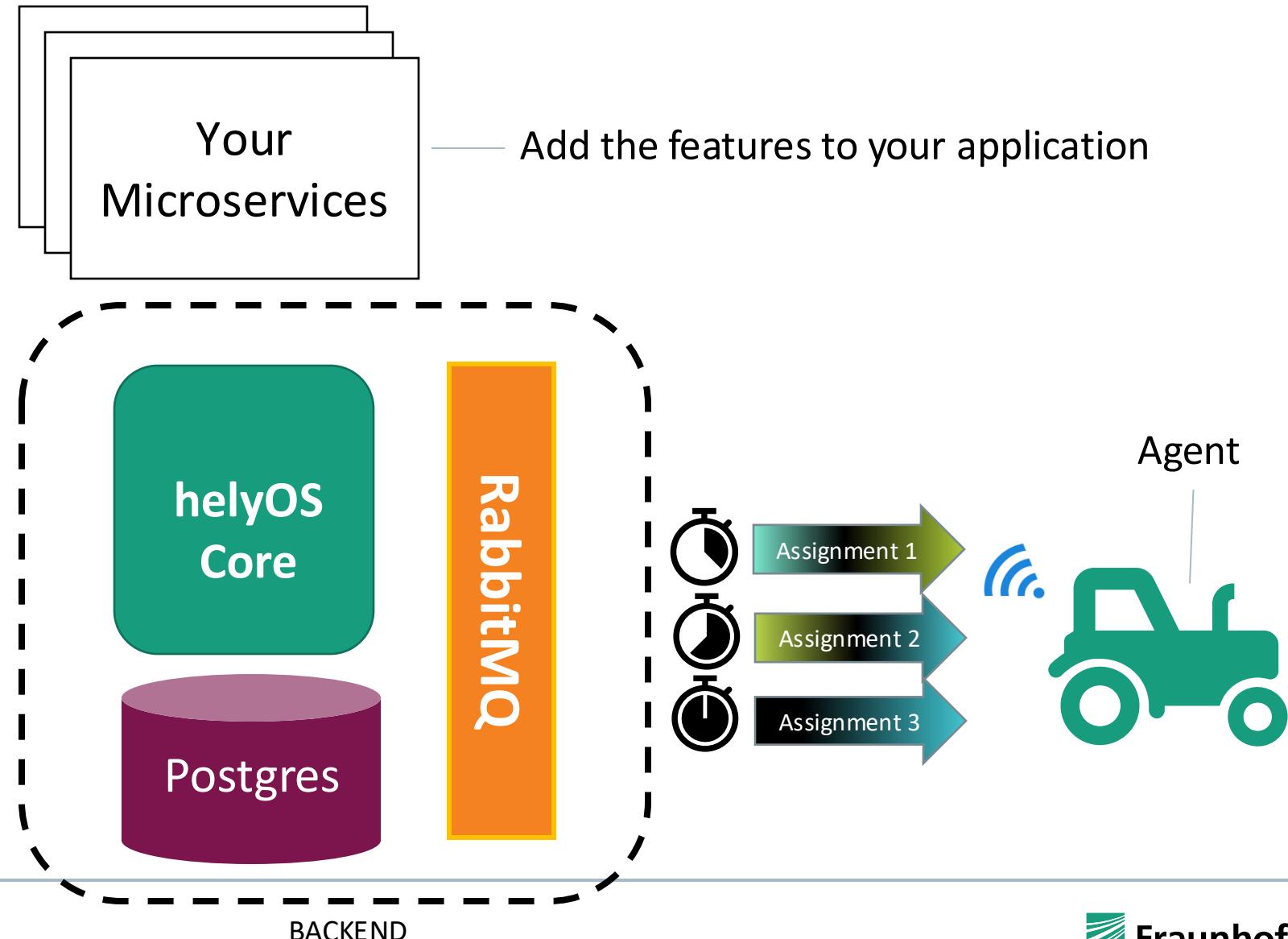
Your Applications

Agriculture platforms,
Logistic management



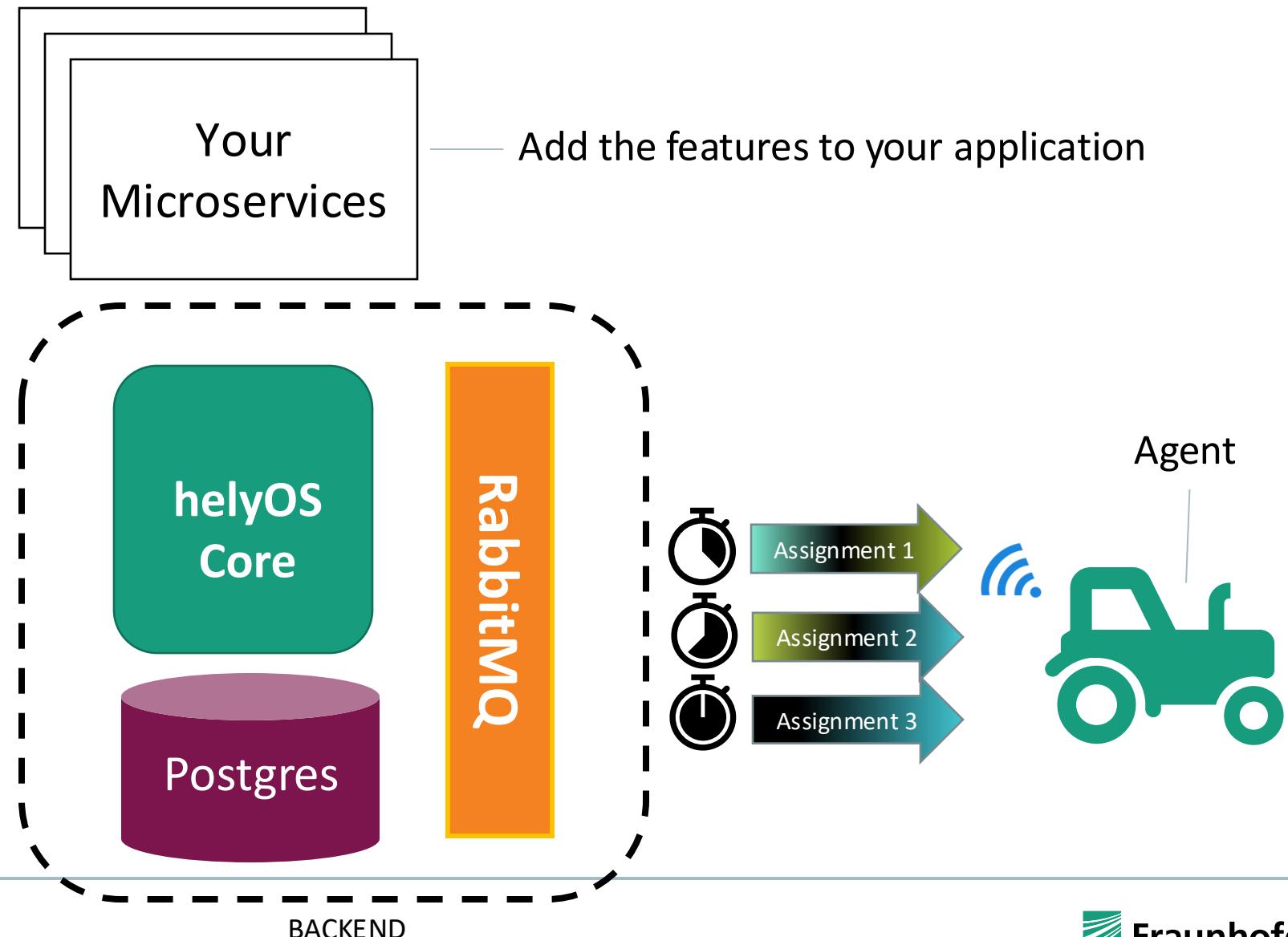
Auth &
Permissions

Web app
(visualization &
prototypes)



The helyOS framework components

Microservices



The main problem helyOS core is solving for you.



Microservices

Mission

Assignment 1



Assignment 2

Assignment3

The problem we are solving with helyOS

Microservices



Mission

Assignment 1

Assignment 2

Assignment3

```
from caculations import assigment_calculation

def my_mission(input_data):
    assignment1 = assigment_calculation(input_data, 'excavator')
    assignment2 = assigment_calculation(input_data, 'truck')
    assignment3 = assigment_calculation(input_data, 'tractor')

    results = [ {
        "uuid": 'excavator',
        "assignment": assignment1,
        "assignment_order": 1},
    {
        "uuid": 'truck',
        "assignment": assignment2,
        "assignment_order": 2},
    {
        "uuid": 'tractor',
        "assignment": assignment3,
        "assignment_order": 3}
    ]

    return results
```

The problem we are solving with helyOS



Microservices

Mission

Assignment 1

Assignment 2

Assignment3

```
from caculations import assigment_calculation

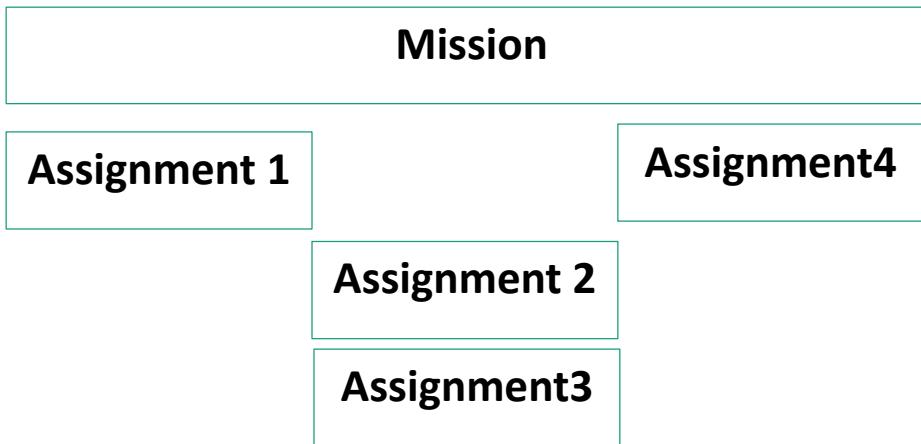
def my_mission(input_data):
    assignment1 = assigment_calculation(input_data, 'excavator')
    assignment2 = assigment_calculation(input_data, 'truck')
    assignment3 = assigment_calculation(input_data, 'tractor')

    results = [ {
        "uuid": 'excavator',
        "assignment": assignment1,
        "assignment_order": 1},
    {
        "uuid": 'truck',
        "assignment": assignment2,
        "assignment_order": 1},
    {
        "uuid": 'tractor',
        "assignment": assignment3,
        "assignment_order": 2}
    ]

    return results
```

The problem we are solving with helyOS

Microservices



```
from calculations import assigment_calculation

def my_mission(input_data):
    assignment1 = assigment_calculation(input_data, 'excavator')
    assignment2 = assigment_calculation(input_data, 'truck')
    assignment3 = assigment_calculation(input_data, 'tractor')
    assignment4 = assigment_calculation(input_data, 'excavator')

    results = [ {
        "uuid": 'excavator',
        "assignment": assignment1,
        "assignment_order": 1},
        {
        "uuid": 'truck',
        "assignment": assignment2,
        "assignment_order": 2},
        {
        "uuid": 'tractor',
        "assignment": assignment3,
        "assignment_order": 2},
        {
        "uuid": 'escavator',
        "assignment": assignment4,
        "assignment_order": 3}
    ]

    return results
```

The problem we are solving with helyOS

Microservices



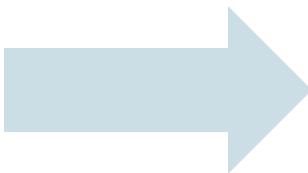
Function in a Module

```
from caculations import assigment_calculation

def my_mission(input_data):
    assignment1 = assigment_calculation(input_data, 'excavator')
    assignment2 = assigment_calculation(input_data, 'truck')
    assignment3 = assigment_calculation(input_data, 'tractor')

    results = [ {
        "uuid": 'excavator',
        "assignment": assignment1,
        "assignment_order": 1},
        {
        "uuid": 'truck',
        "assignment": assignment2,
        "assignment_order": 2},
        {
        "uuid": 'tractor',
        "assignment": assignment3,
        "assignment_order": 3}
    ]

    return results
```



Function Wrapped as a service

```
from flask import Flask, request, jsonify
from caculations import assigment_calculation

app = Flask(__name__)

@app.route('/plan_job', methods=['POST'])
def my_mission():
    input_data = request.json.get('input_data')
    assignment1 = assigment_calculation(input_data, 'excavator')
    assignment2 = assigment_calculation(input_data, 'truck')
    assignment3 = assigment_calculation(input_data, 'tractor')

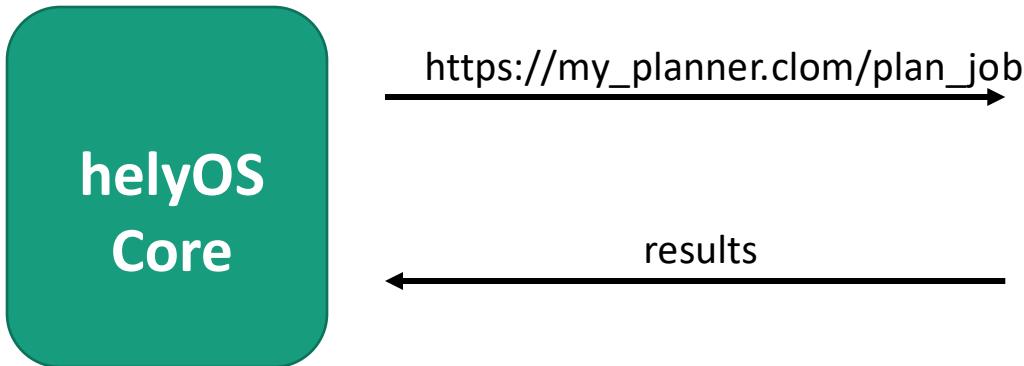
    results = [
        {
            "uuid": 'excavator',
            "assignment": assignment1,
            "assignment_order": 1
        },
        {
            "uuid": 'truck',
            "assignment": assignment2,
            "assignment_order": 1
        },
        {
            "uuid": 'tractor',
            "assignment": assignment3,
            "assignment_order": 2
        }
    ]

    return jsonify(results)

if __name__ == '__main__':
    app.run(debug=True)
```

The problem we are solving with helyOS

Microservices



Microservice

```
from flask import Flask, request, jsonify
from calculations import assignment_calculation

app = Flask(__name__)

@app.route('/plan_job', methods=['POST'])
def my_mission():
    input_data = request.json.get('input_data')
    assignment1 = assignment_calculation(input_data, 'excavator')
    assignment2 = assignment_calculation(input_data, 'truck')
    assignment3 = assignment_calculation(input_data, 'tractor')

    results = [
        {
            "uuid": 'excavator',
            "assignment": assignment1,
            "assignment_order": 1
        },
        {
            "uuid": 'truck',
            "assignment": assignment2,
            "assignment_order": 1
        },
        {
            "uuid": 'tractor',
            "assignment": assignment3,
            "assignment_order": 2
        }
    ]

    return jsonify(results)

if __name__ == '__main__':
    app.run(debug=True)
```

The helyOS framework components



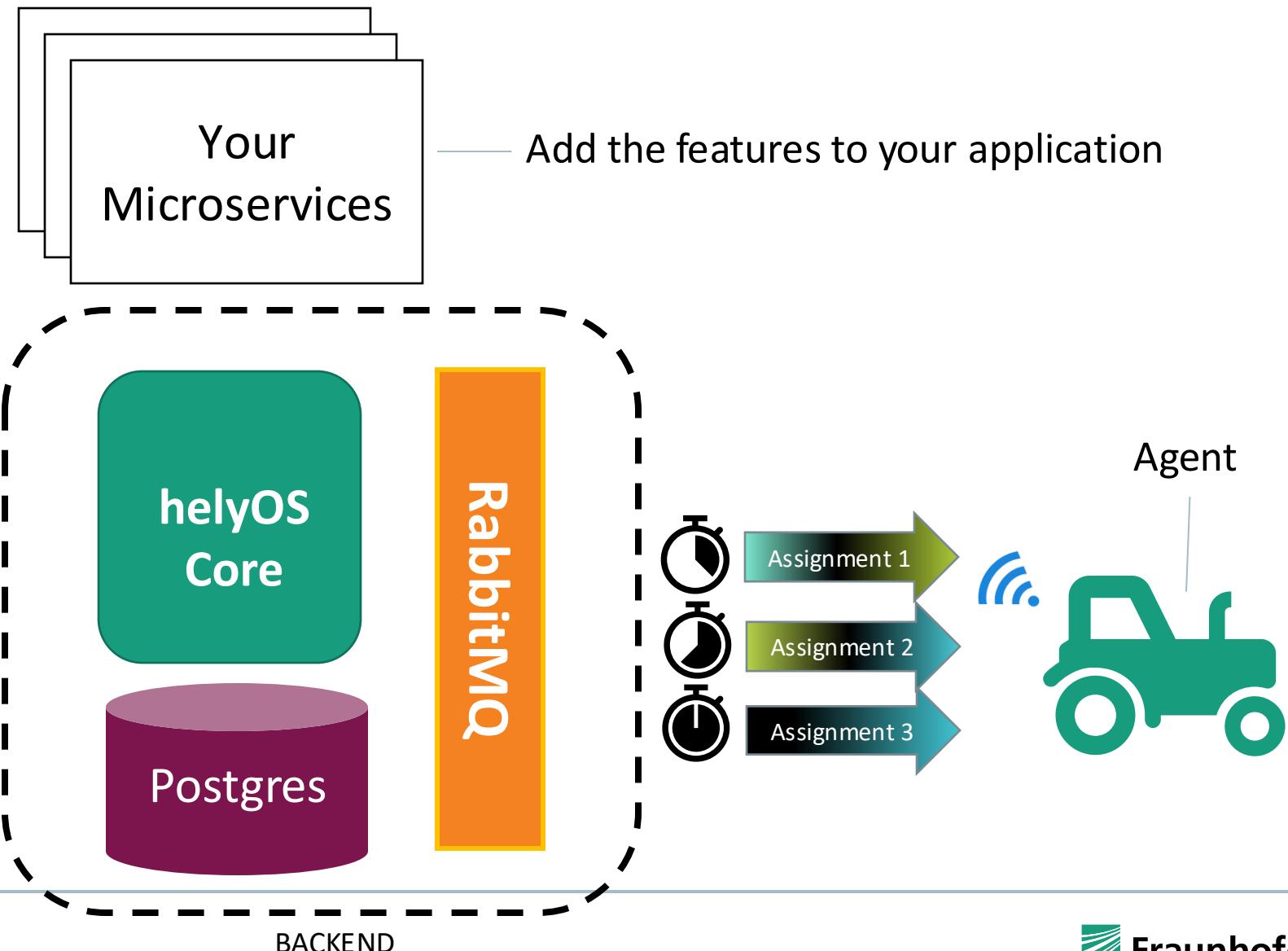
Your Applications

Agriculture platforms,
Logistic management

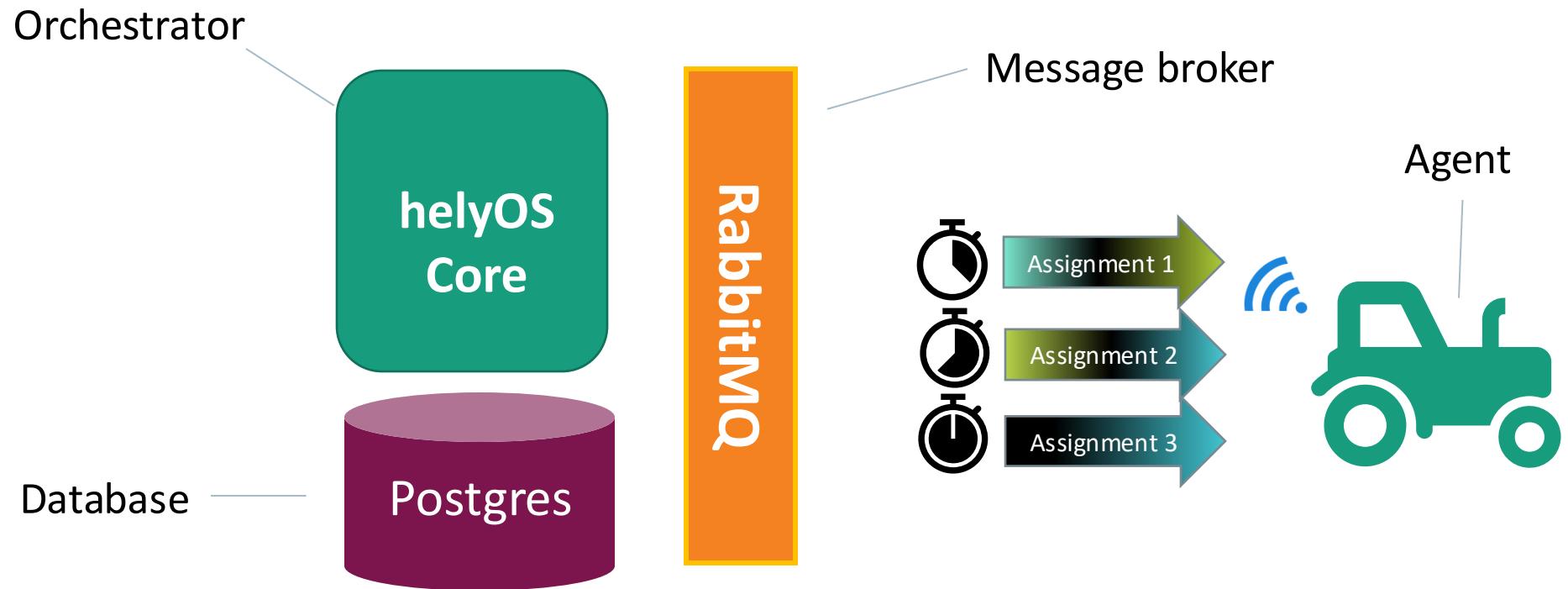


Auth &
Permissions

Web app
(visualization &
prototypes)

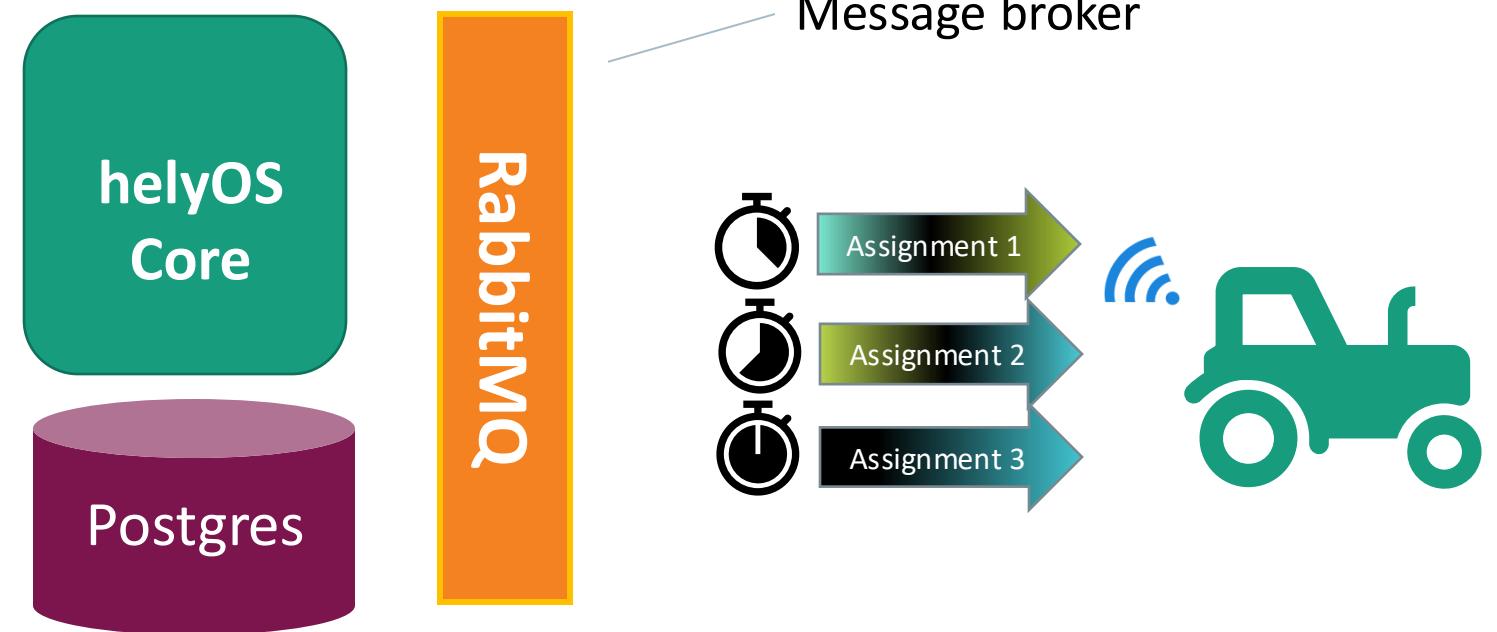


The helyOS framework components



The helyOS framework components

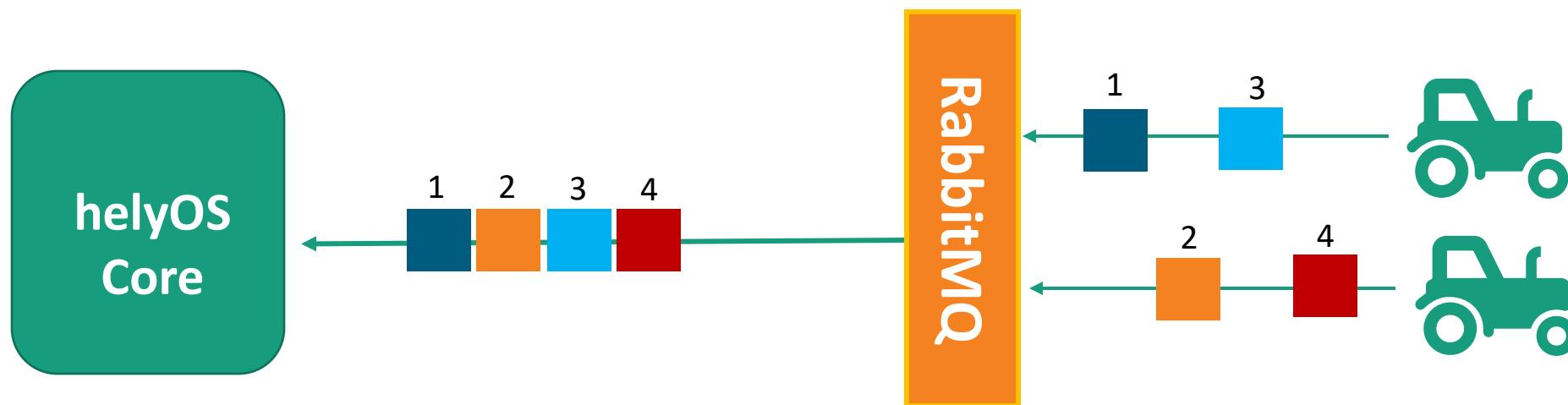
Message Broker

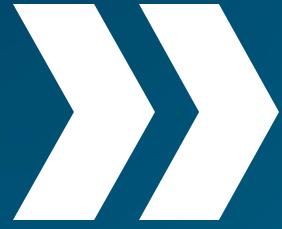


The helyOS framework components

Why RabbitMQ ?

- Queueing is crucial for the correct orchestration.
- Multiple Protocols including AMQP, MQTT, and STOMP
- Robust security features, which help prevent message forging.





Deploying Demo Application

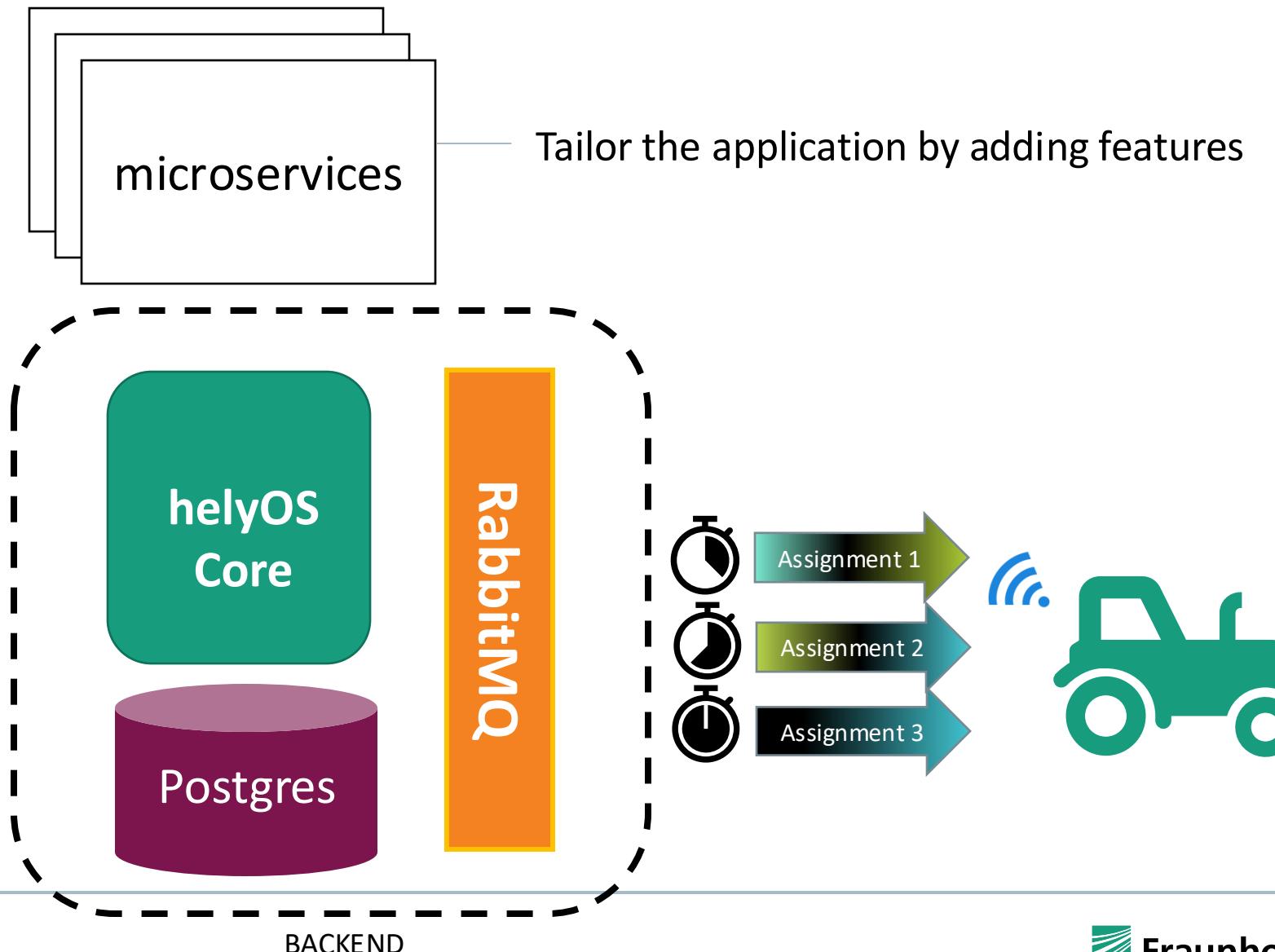
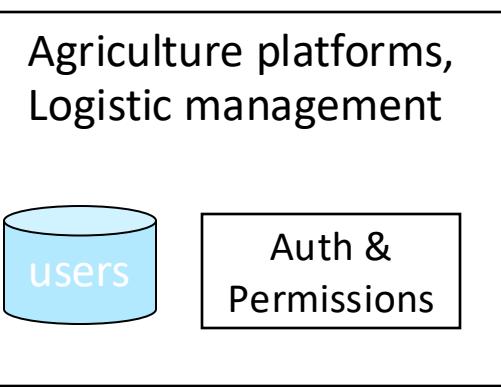
- Blazing-Fast Course on Docker
- Install helyOS Backend on Your Computer
- Creating a Microservice
- Running Your Application
- Migrate Your Installation to the Cloud



The helyOS framework components



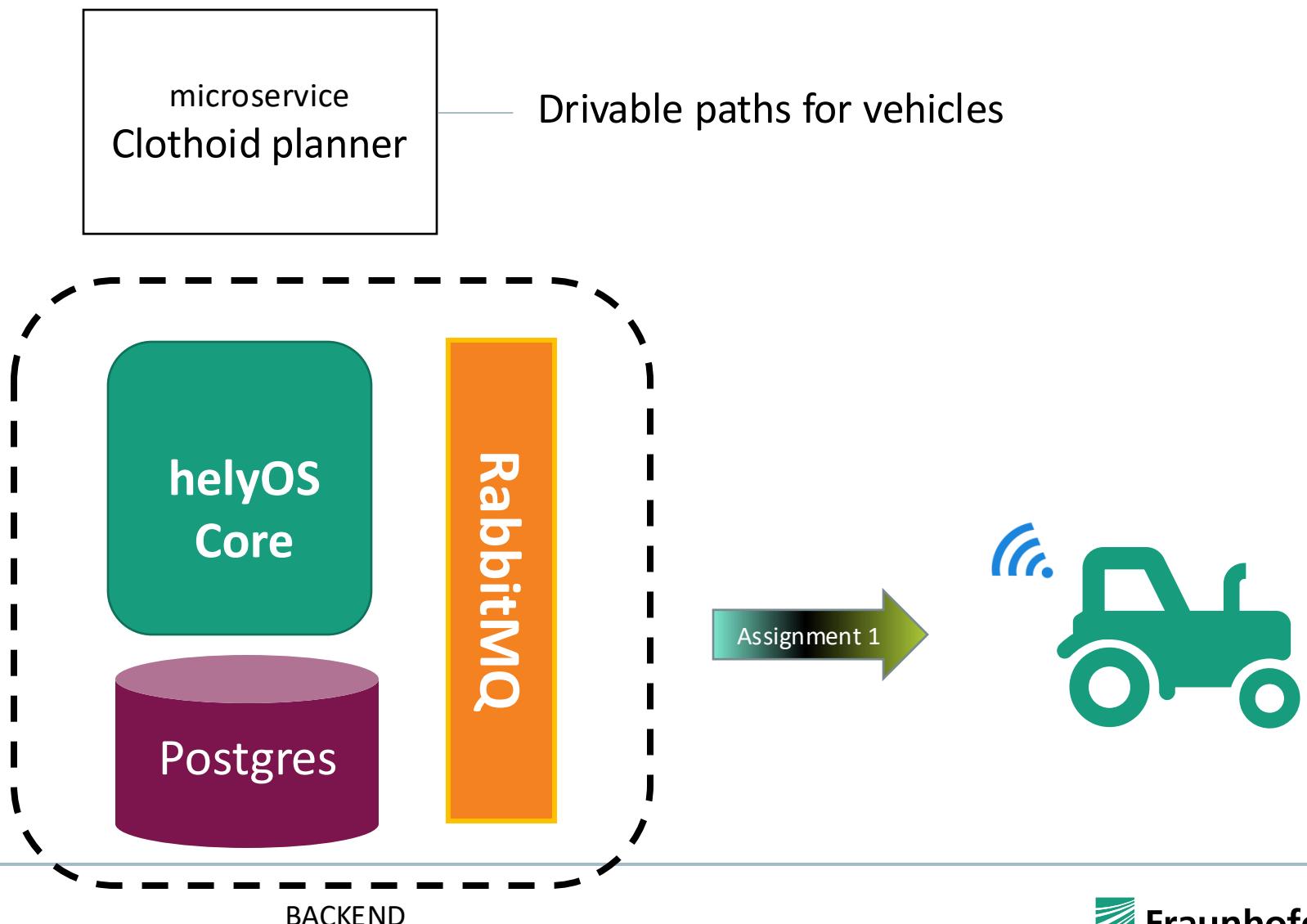
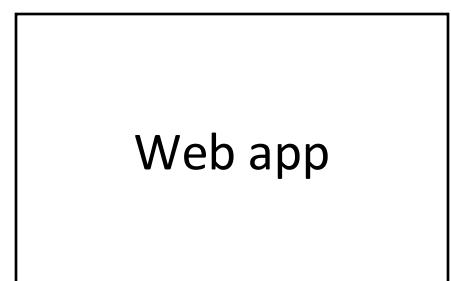
Your Applications



The helyOS framework components

Our Application

Your Applications



Blazing-Fast Course on Docker Install



<https://www.docker.com/>



The screenshot shows the Docker Desktop landing page. It features the Docker Desktop logo at the top left. Below it is a large, bold headline: "The #1 containerization software for developers and teams". A subtext below the headline reads: "Streamline development with Docker Desktop's powerful container tools." At the bottom, there are two blue call-to-action buttons: "Create an account" and "Download Docker Desktop".

<https://docs.docker.com/engine/install/>

Install Docker Engine

This section describes how to install Docker Engine on Linux, also known as Docker CE. Docker Engine is also available for Windows, macOS, and Linux, through Docker Desktop. For instructions on how to install Docker Desktop, see: [Overview of Docker Desktop](#).

Supported platforms

Platform	x86_64 / amd64	arm64 / aarch64	arm (32-bit)	ppc64le	s390x
CentOS	✓	✓		✓	
Debian	✓	✓	✓	✓	
Fedora	✓	✓		✓	
Raspberry Pi OS (32-bit)			✓		
RHEL	✓	✓		✓	
SLES				✓	
Ubuntu	✓	✓	✓	✓	✓
Binaries	✓	✓	✓		

- Docker **containers** are like a **super-light virtual machine**; they have their own “disk”, OS and network.
- **Host machine:** It is where you installed the Docker, where the containers run.
- The docker **image** encloses the base data of the container.
- **docker-compose.yml** is a file that describes which and how the containers run.

Blazing-Fast Course on Docker

docker-compose.yml

```
services:  
  local_rabbitmq:  
    image: rabbitmq:3-management  
    hostname: local_rabbitmq  
    volumes:  
      - rbmq_data:/var/lib/rabbitmq/  
  
    networks:  
      - my-docker-net  
    ports:  
      - 15672:15672  
  
  networks:  
    my-docker-net:  
      external: true  
  
  volumes: # For persistent data  
  rbmq_data:  
    external: false
```

→ Image name
→ Define a hostname for this container
→ Mapping internal folder into host file system
→ Open port to the host machine
→ Name of network
→ Storing the mapped folder on the host disk.

Blazing-Fast Course on Docker

Interacting with docker-compose.yml

We will be using four docker commands:

docker network create my-docker-net	Creates a virtual network.
docker compose up -d	Starts the containers.
docker compose down	Stops and removes all containers from the RAM.
docker compose down -v	Stops and removes all containers from the RAM and remove all volumes (permanent data).

Optional parameter: **-f my_docker-compose.yml**

Blazing-Fast Course on Docker

Interacting with docker-compose.yml



```
services:

  local_rabbitmq:
    image: rabbitmq:3-management
    hostname: local_rabbitmq
    volumes:
      - rbmq_data:/var/lib/rabbitmq/
    networks:
      - my-docker-net
    ports:
      - 15672:15672

  local_postgres:
    image: postgres:13
    hostname: local_postgres
    volumes:
      - postgres_data:/var/lib/postgresql/data/
    environment:
      POSTGRES_USER: helyos_db_admin
      POSTGRES_PASSWORD: helyos_secret
    networks:
      - my-docker-net

  networks:
    my-docker-net:
      external: true

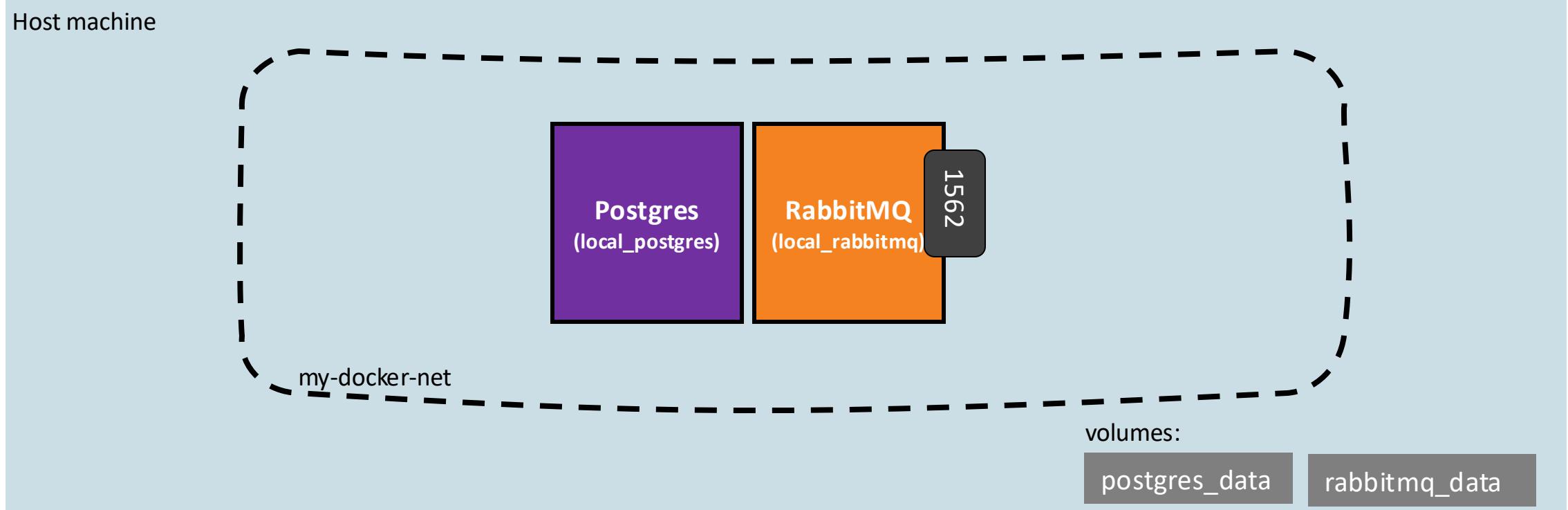
  volumes: # For persistent data
    postgres_data:
      external: false
    rbmq_data:
      external: false
```

Parameter: -f my_docker-compose-pg-rmbq.yml

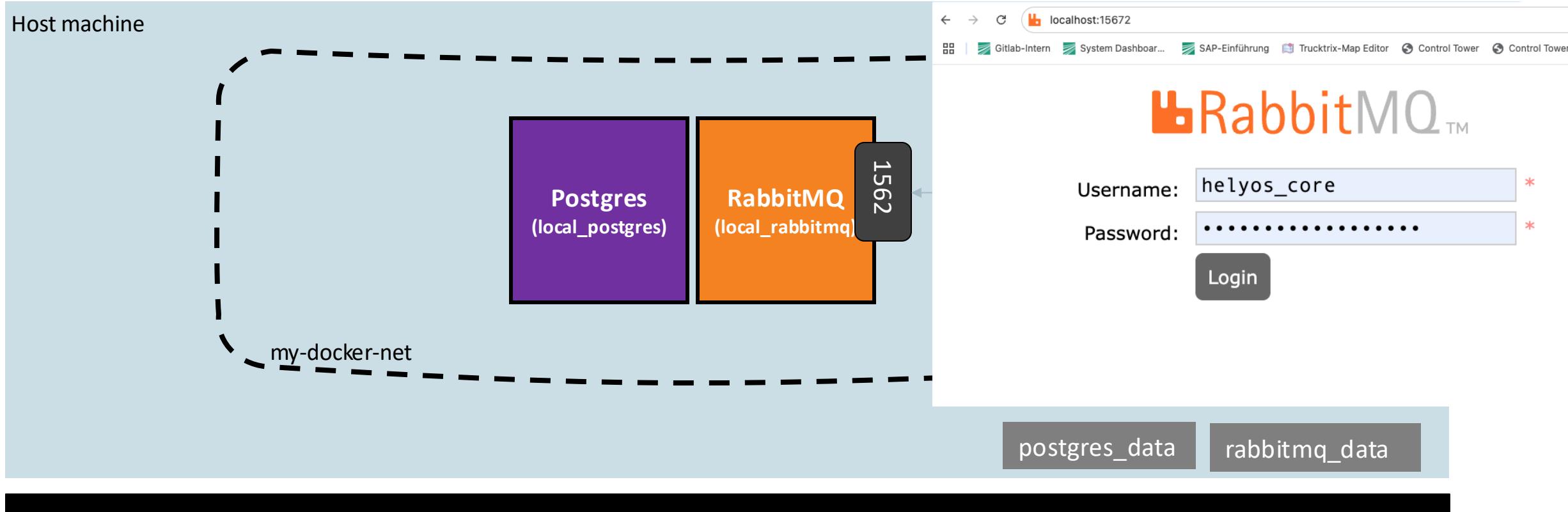
Host machine

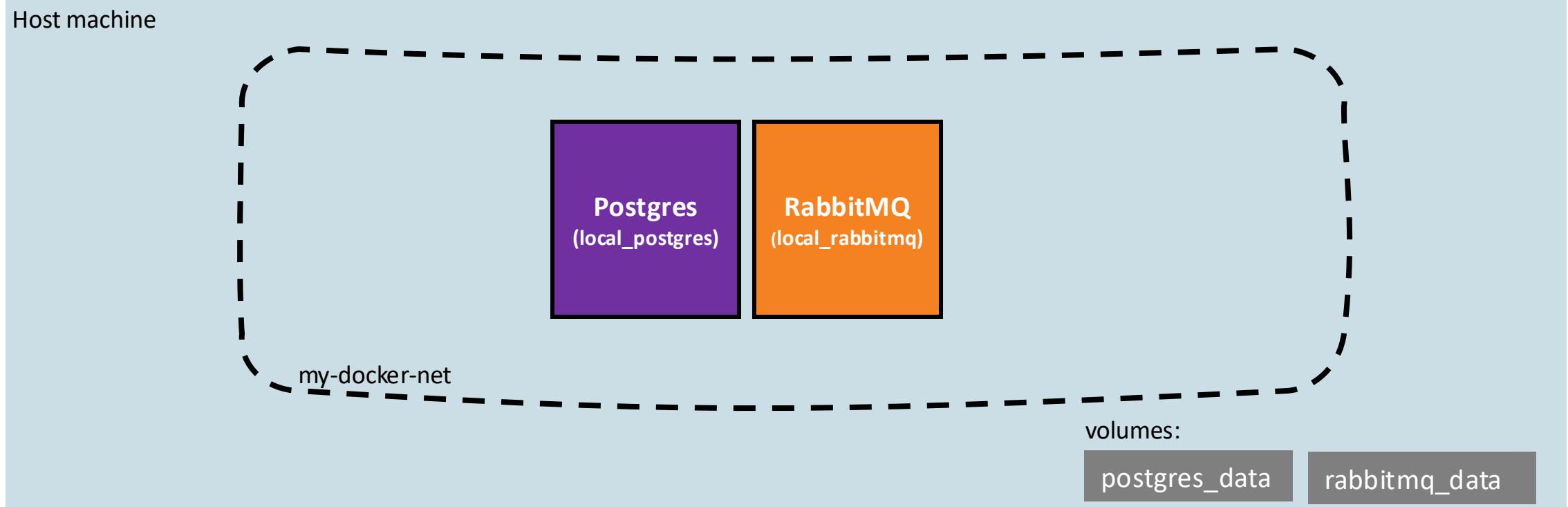


```
$ docker network create my-docker-net
```

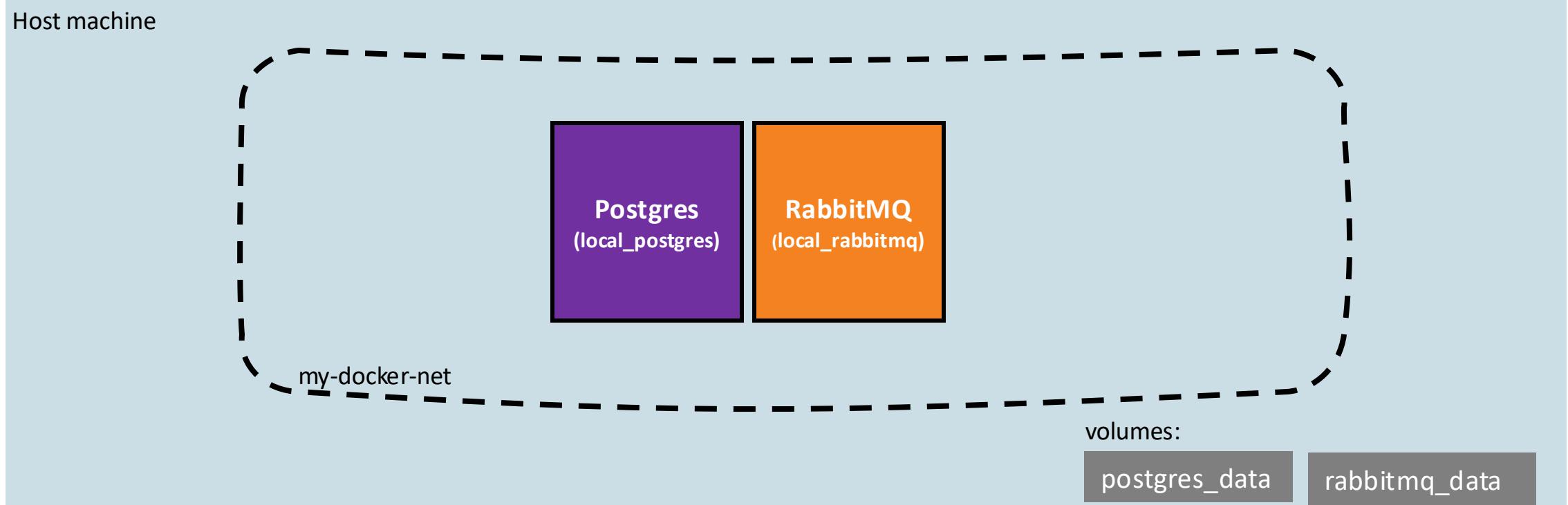


```
$ docker compose -f docker-compose-pg-rbmq.yml up -d
```

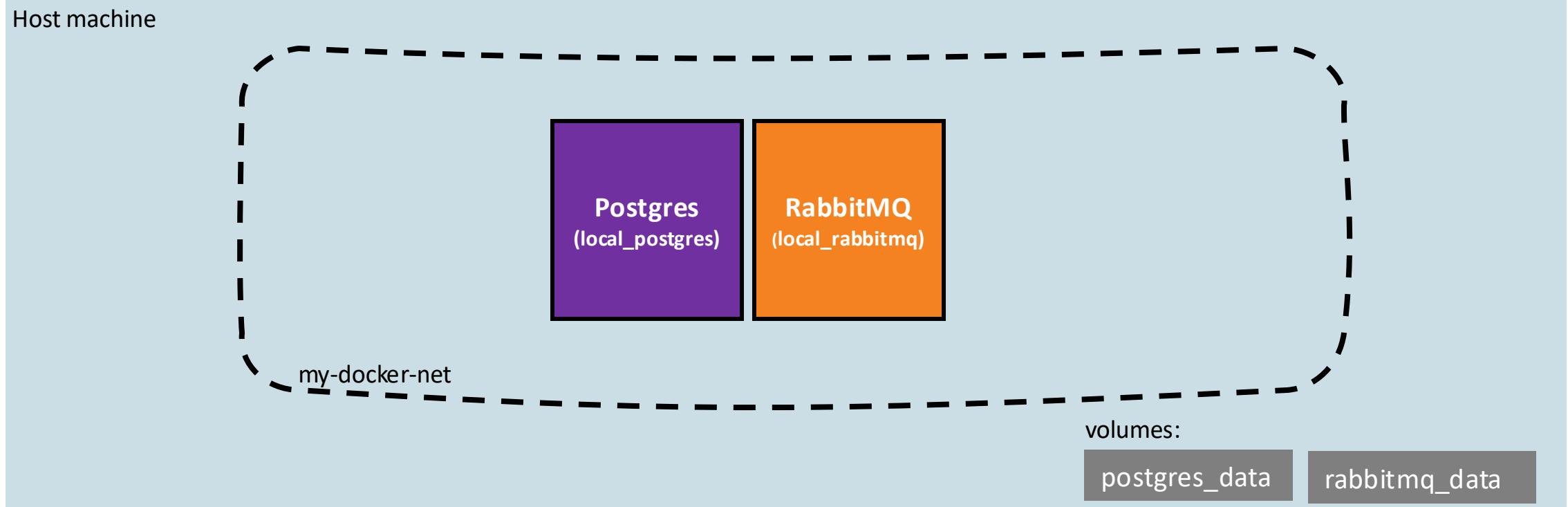




```
$ docker compose -f docker-compose-pg-rbmq.yml down
```



```
$ docker compose -f docker-compose-pg-rbmq.yml up
```



```
$ docker compose -f docker-compose-pg-rbmq.yml down -v
```

docker ps

List running containers.

docker system prune

Unexpected issues, it clean up unused containers, networks, images, and build cache.

docker compose –f my_docker-compose.yml logs

See the logs of all containers listed in the docker compose file.

Now that you are an expert in Docker ;)
let's start our helyOS project !

Install helyOS Backend on Your Computer



```
services:  
  
  helyos_core:  
    image: helyosframework/helyos_core:2.2.0-rc1  
  
    ports:  
      - 5002:5002 # Websocket: Push Notifications.  
      - 5000:5000 # GraphQL: Communicate with External Applications  
      - 8080:8080 # HelyOS Dashboard: Settings and Monitoring for Developers  
  
    volumes:  
      - ./etc_helyos:/etc/helyos:ro  
  
    environment:  
      # DATABASE  
      - PGUSER=helyos_db_admin  
      - PGPASSWORD=helyos_secret  
      - JWT_SECRET=keyboard_kitten  
      - PGHOST=local_postgres  
      - PGDATABASE=smartfarm_db  
      - PGPORT=5432  
      - GQLPORT=5000  
  
      # RABBITMQ  
      - RBMQ_HOST=local_rabbitmq  
      - RBMQ_PORT=5672  
      - RBMQ_API_PORT=15672  
      - RBMQ_SSL=False # For TLS support  
      - RBMQ_API_SSL=False # For TLS support  
  
      # RBMQ ACCOUNTS  
      - CREATE_RBMQ_ACCOUNTS=True  
      - RBMQ_ADMIN_USERNAME=helyos_rabbitmq_admin # Set it if CREATE_RBMQ_ACCOUNTS is True  
      - RBMQ_ADMIN_PASSWORD=helyos_secret # Set it if CREATE_RBMQ_ACCOUNTS is True  
      - AGENT_AUTO_REGISTER_TOKEN=0001-0002-0003-0000-0004 # Delete it for production  
  
    networks:  
      - my-docker-net  
  
networks:  
  my-docker-net:  
    external: true
```

helyOS official image

RabbitMQ hostname

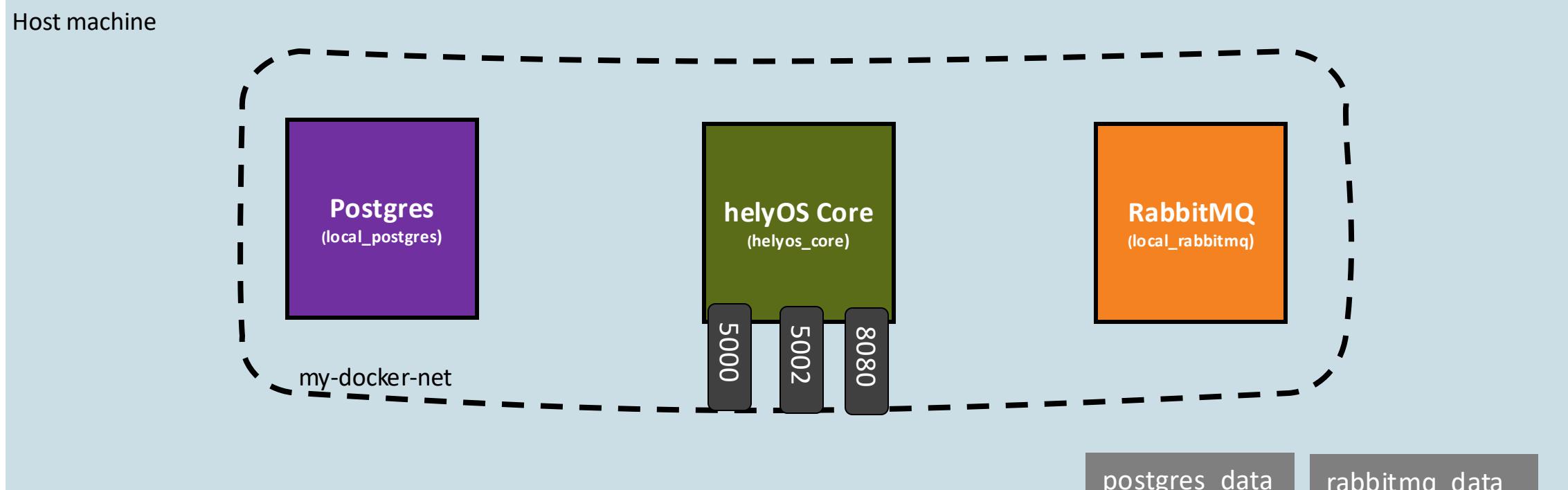
Postgres hostname

Only for development: any agent with this token can access helyOS.

Name	Size	Kind
.ssl_keys	--	Folder
helyos_private.key	916 bytes	Keynote
helyos_public.key	272 bytes	Keynote
config	--	Folder
microservices.yml	481 bytes	YAML
missions.yml	256 bytes	YAML
db_initial_data	--	Folder
load_initial_data.sql	967 bytes	Document

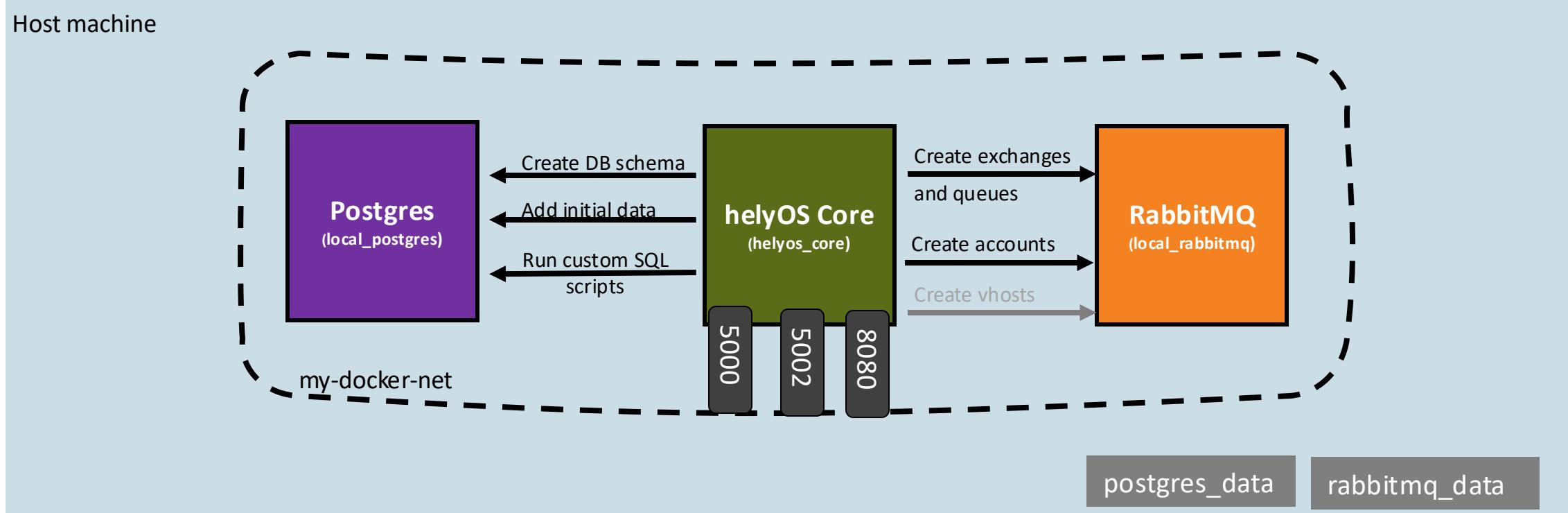
<https://helyos-manual.readthedocs.io/en/latest/2-helyos-configuration/getting-started.html#helyos-setting>

Install helyOS Backend on Your Computer



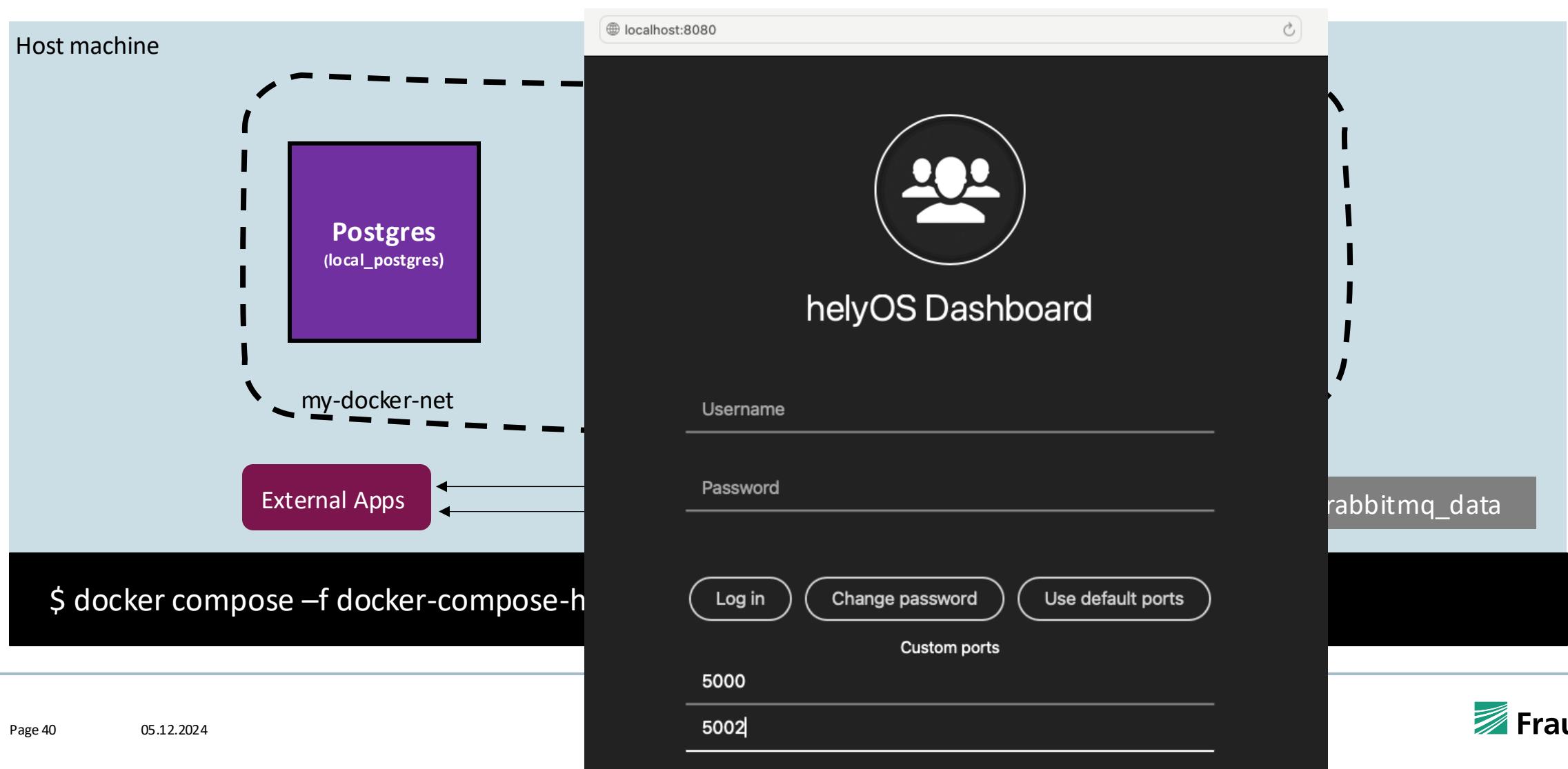
```
$ docker compose -f docker-compose-helyos_core up -d
```

Install helyOS Backend on Your Computer



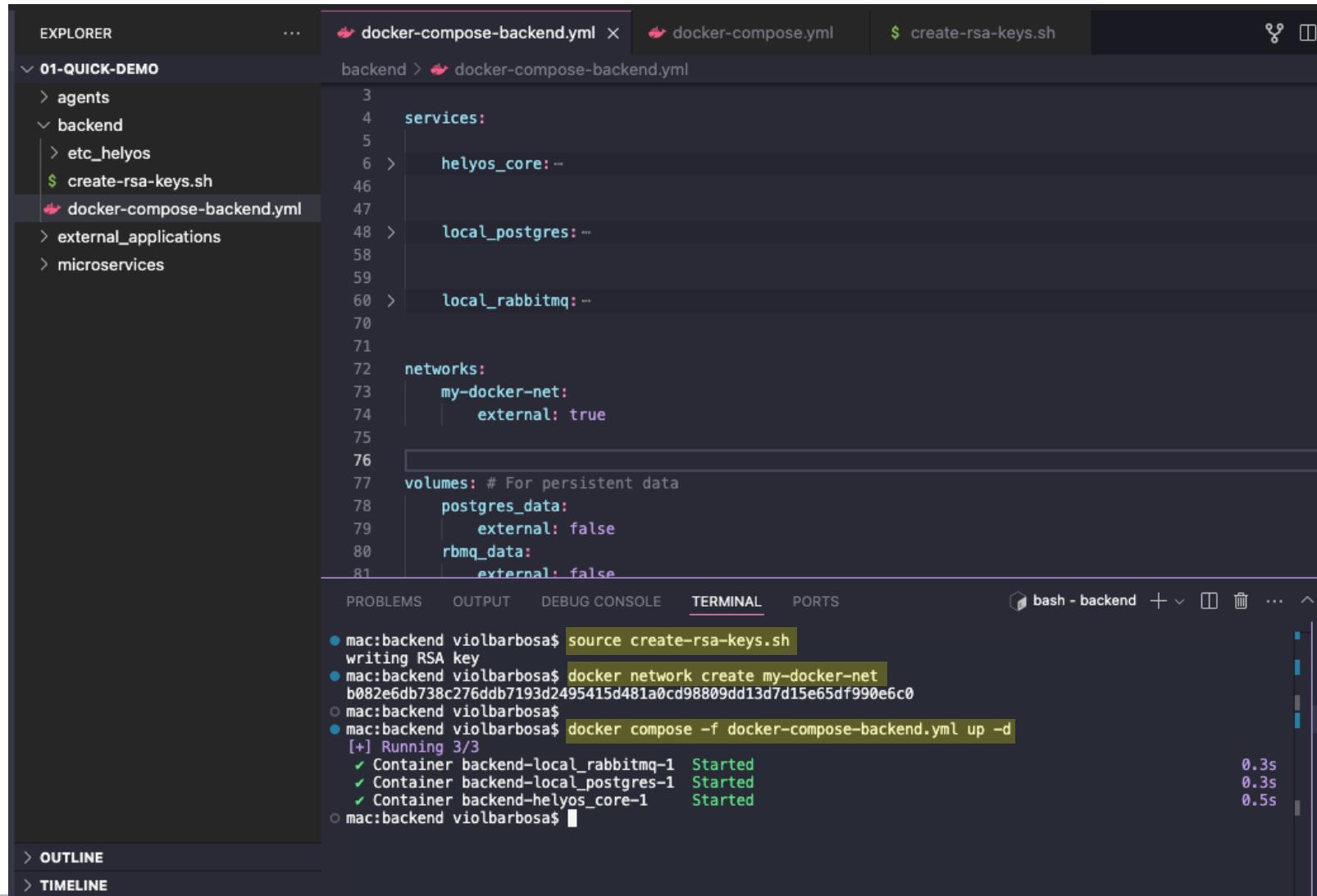
```
$ docker compose -f docker-compose-helyos_core up -d
```

Install helyOS Backend on Your Computer



Install helyOS Backend on Your Computer

Wrapping up



The screenshot shows a code editor interface with several tabs and panels. The tabs at the top include 'EXPLORER', 'docker-compose-backend.yml' (active), 'docker-compose.yml', and '\$ create-rsa-keys.sh'. The 'EXPLORER' panel on the left shows a project structure under '01-QUICK-DEMO' with folders like 'agents', 'backend', 'etc_helyos', and 'microservices', along with files such as 'create-rsa-keys.sh' and 'docker-compose-backend.yml'. The main code editor area displays a Docker Compose configuration file:

```
3
4 services:
5
6 > helyos_core: ...
46
47
48 > local_postgres: ...
58
59
60 > local_rabbitmq: ...
70
71
72 networks:
73   my_docker_net:
74     external: true
75
76
77 volumes: # For persistent data
78   postgres_data:
79     external: false
80   rbmq_data:
81     external: false
```

Below the code editor, there are tabs for 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', 'TERMINAL' (selected), and 'PORTS'. The 'TERMINAL' tab shows a command-line session:

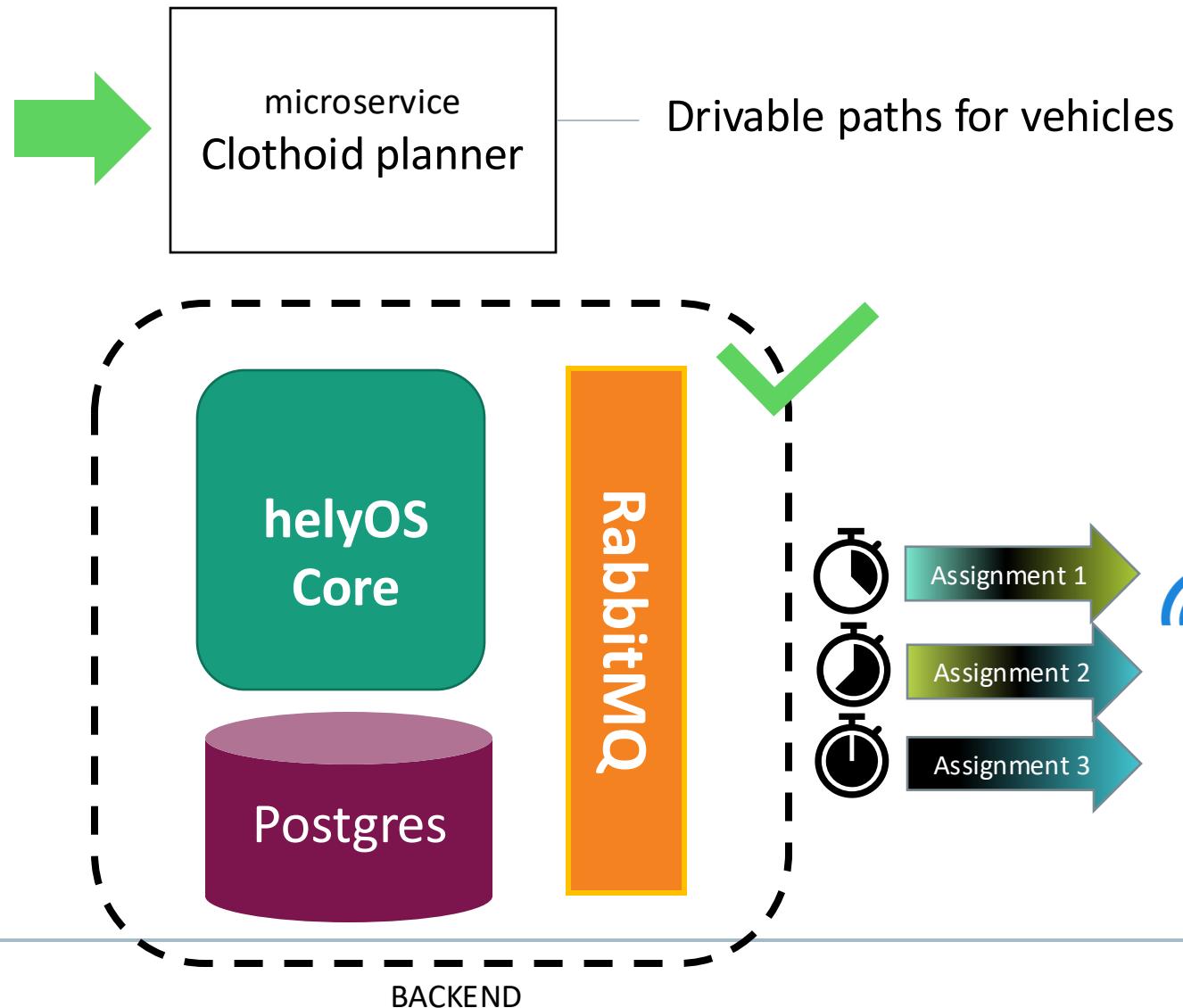
```
● mac:backend violbarbosa$ source create-rsa-keys.sh
writing RSA key
● mac:backend violbarbosa$ docker network create my-docker-net
b082e6db738c276ddb7193d2495415d481a0cd98809dd13d7d15e65df990e6c0
○ mac:backend violbarbosa$
● mac:backend violbarbosa$ docker compose -f docker-compose-backend.yml up -d
[+] Running 3/3
  ✓ Container backend-local_rabbitmq-1 Started
  ✓ Container backend-local_postgres-1 Started
  ✓ Container backend-helyos_core-1 Started
○ mac:backend violbarbosa$
```

At the bottom, there are 'OUTLINE' and 'TIMELINE' buttons.

The helyOS framework components



Your Applications.



Creating a Microservice

Code: service.py

```
from flask import Flask, jsonify, request
from path_calculation import calculate_path

app = Flask(__name__)

@app.post("/plan_job/")
def getPath():
    """
    Calculate path using a data format COMPATIBLE with the AGENT.
    """

    request_body = request.get_json()
    request_data = request_body['request'] # Input from external application.
    helyos_context = request_body['context'] # Snapshot from helyos database.

    # Parse input data
    agent_uuid = request_data.get('agent_uuid', None)
    initial_position = request_data.get('initial_position', None)
    destination = request_data.get('destination', initial_position)

    # Calculate path
    assignment = calculate_path(initial_position, destination)

    # Return response
    response = {
        "results": [
            {
                "agent_uuid": agent_uuid,
                "assignment": assignment
            }
        ]
    }

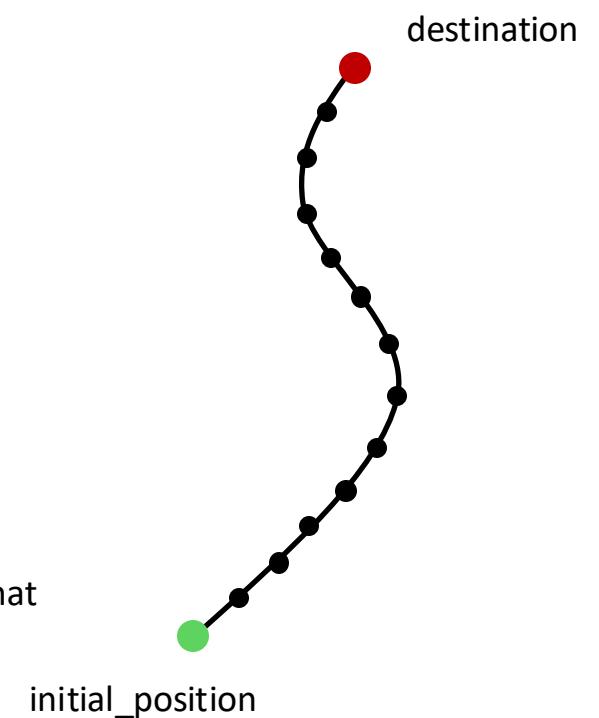
    return jsonify(response)

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=9002, debug=True)
```

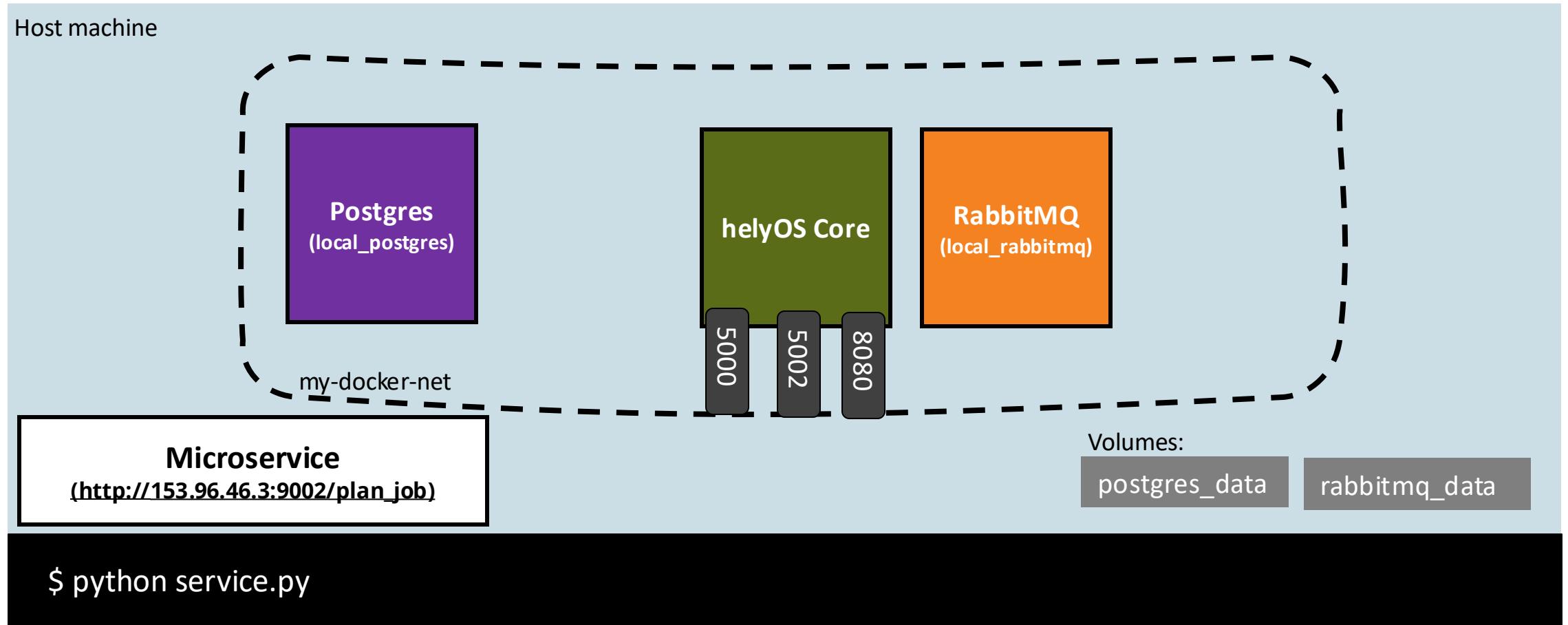
Example of request_data:

```
{
    "initial_position": {
        "x": 23234,
        "y": 54653,
        "orientations": [
            0
        ]
    },
    "destination": {
        "x": 11234,
        "y": 64653,
        "orientations": [
            1570.7
        ]
    }
}
```

Data format compatible with the vehicle that will receive this assignment.



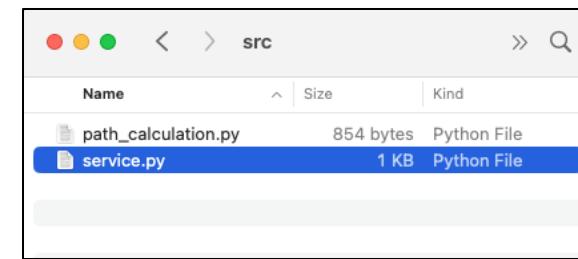
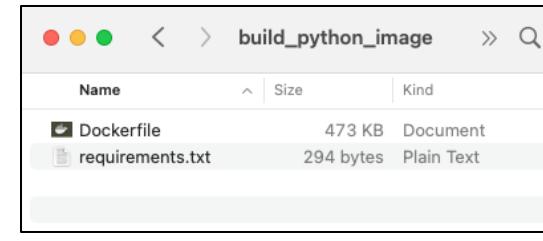
Running microservice On the host machine



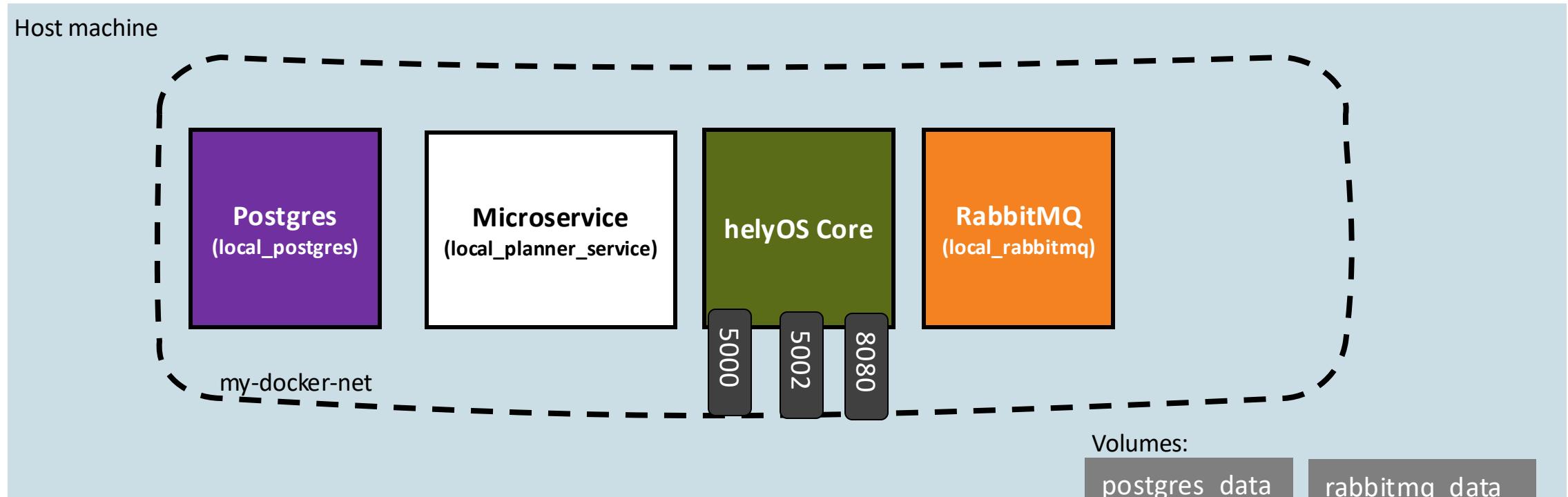
Creating microservice

Set the service in a container

```
services:  
  
  local_planner_service:  
    restart: always  
    container_name: local_planner_service  
  
    build:  
      dockerfile: ./build_python_image/Dockerfile ←  
  
    volumes:  
      - ./src/:/app/src ←  
  
    networks:  
      - my-docker-net  
  
    ports:  
      - "9002:9002" # optional  
  
  
  networks:  
    my-docker-net:  
      external: true
```



Running microservice As a Docker container

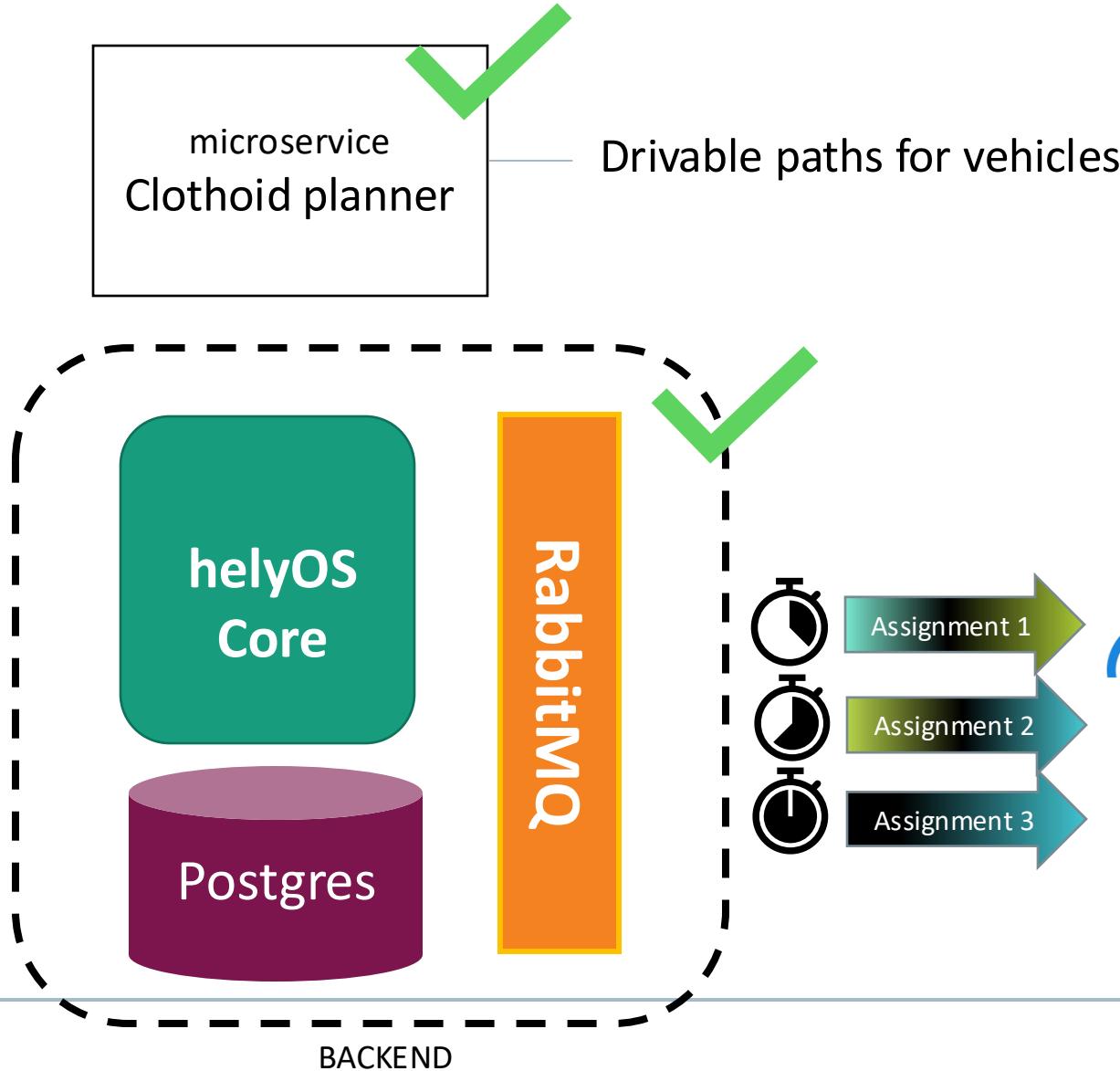
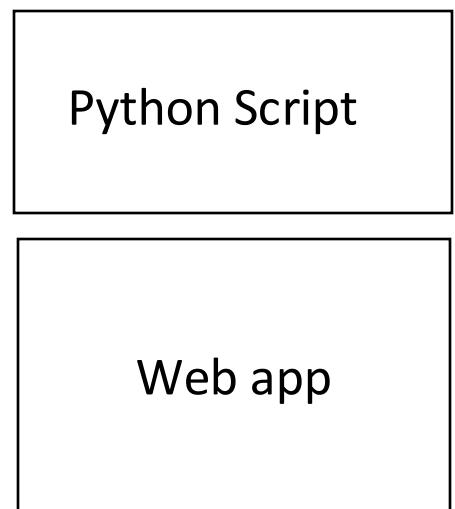


```
$ docker compose -f docker-compose-microservice.yml up -d
```

The helyOS framework components

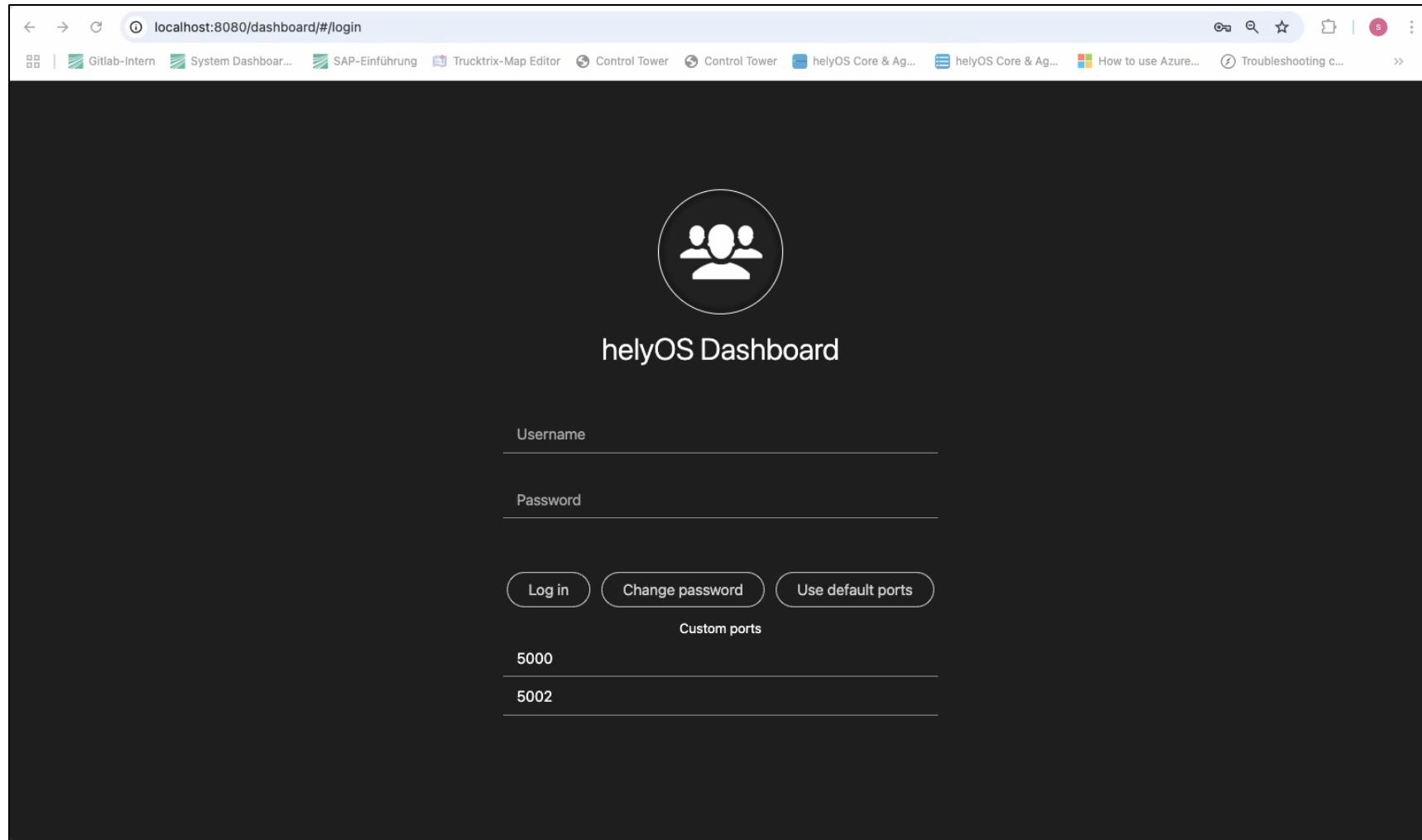


—
Your Applications.

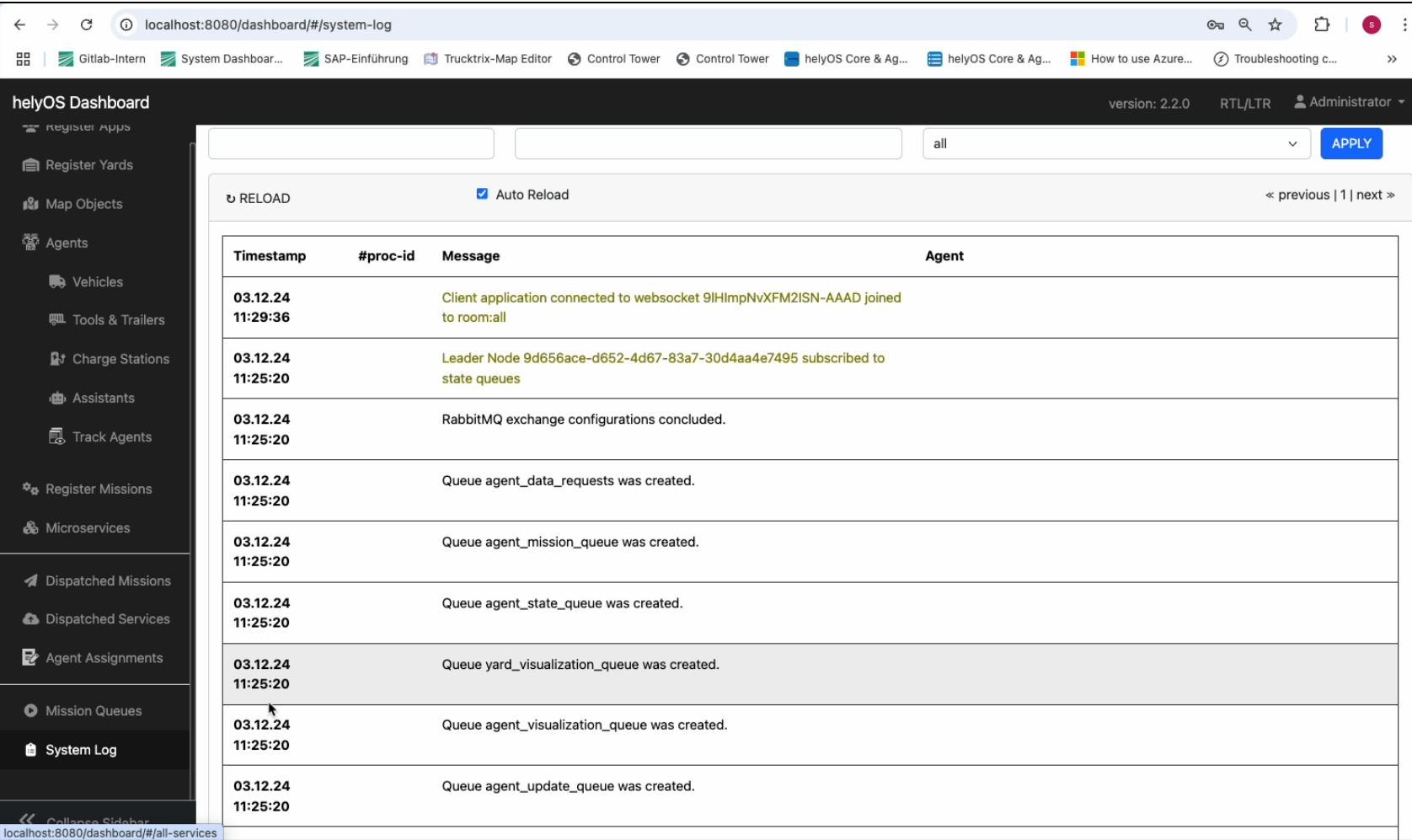


A look into helyOS Dashboard

Logs



A look into helyOS Dashboard Microservices



The screenshot shows the helyOS Dashboard interface with the 'System Log' page selected. The left sidebar contains navigation links for Register Apps, Register Yards, Map Objects, Agents (with sub-options for Vehicles, Tools & Trailers, Charge Stations, Assistants, Track Agents), Register Missions, Microservices (with sub-options for Dispatched Missions, Dispatched Services, Agent Assignments), Mission Queues, and System Log. The main content area displays a log table with columns: Timestamp, #proc-id, Message, and Agent. The log entries show the initial connection of a client application, the subscription of the Leader Node to state queues, and the creation of various message queues (agent_data_requests, agent_mission_queue, agent_state_queue, yard_visualization_queue, agent_visualization_queue, agent_update_queue) on March 12, 2024, at 11:25:20.

Timestamp	#proc-id	Message	Agent
03.12.24 11:29:36		Client application connected to websocket 9IHlmpNvXFM2ISN-AAAD joined to room:all	
03.12.24 11:25:20		Leader Node 9d656ace-d652-4d67-83a7-30d4aa4e7495 subscribed to state queues	
03.12.24 11:25:20		RabbitMQ exchange configurations concluded.	
03.12.24 11:25:20		Queue agent_data_requests was created.	
03.12.24 11:25:20		Queue agent_mission_queue was created.	
03.12.24 11:25:20		Queue agent_state_queue was created.	
03.12.24 11:25:20		Queue yard_visualization_queue was created.	
03.12.24 11:25:20		Queue agent_visualization_queue was created.	
03.12.24 11:25:20		Queue agent_update_queue was created.	

A look into helyOS Dashboard

External Applications

localhost:8080/dashboard/#/all-services

version: 2.2.0 RTL/LTR Administrator

Register Apps Apps are the building blocks used to create missions. These services can be employed to create paths, convert and update maps, and facilitate communication with cloud systems.
show more ...

Microservices

Service Type	Domain	Name	URL	API key	Enabled
drive	Assignment planner	my_path_planner	http://local_planner_service:9002/plan_job/	CN783V9SygdG0deHgfesdfsaeNuCqwbt	true

Add **Delete** **Enable/Disable**

my_path_planner

Dummy service

Name: my_path_planner URL: http://local_planner_service:9002/plan_job/

Domain: Assignment planner Type: drive Process time limit (sec): 300

API Key: CN783V9SygdG0deHgfesdfsaeNuCqwbt

Context data:

- Include data of the agents involved in the mission
- Include data of all agents in the yard
- Include yard and map objects data

Specify the types of map objects to include:

Creating Application

Python script run on the host machine

helyos_core url

Application token

See Python code examples in the documentation

<https://helyos-manual.readthedocs.io/en/latest/3-helyos-and-client-apps/examples.html>

```
import random, time
from helyos_requests import get_free_agents, create_work_process

# Define the GraphQL endpoint and headers
hostname = "http://localhost"
port = 5000
headers = {
    "Content-Type": "application/json; charset=utf-8",
    "Authorization": "Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJyb2xlIjoicm9sZV9hcHBsaWNhdGlvbiIsI
graphql_conf = { 'url': f'{hostname}:{port}/graphql', 'headers': headers }

# Main function to request a random mission for a random free agent
def request_random_mission():
    free_agents = get_free_agents(graphql_conf)
    if not free_agents:
        return

    # Randomly select a free agent
    for selected_agent in free_agents:
        agent_uuid = selected_agent['uuid']
        agent_yard_id = selected_agent['yardId']
        current_pos = {'x':selected_agent['x'],
                      'y':selected_agent['y'],
                      'orientations':selected_agent['orientations']}

        random_destination = { 'x':random.randint(-10000, 10000),
                               'y':random.randint(-20000, 20000),
                               'orientations':[random.randint(-6000, 6000)]}

        request_data = {'agent_uuid': agent_uuid,
                       'initial_position': current_pos,
                       'destination': random_destination}

        # Create a work process (mission) for the selected agent
        mission = create_work_process(graphql_conf,agent_uuid, agent_yard_id, "driving", request_data)
        print(f"Mission created with ID: {mission['id']} for Agent UUID: {agent_uuid}")

    # Execute the main function
try:
    print("This command will repeat until you press Control + C")
    while True:
        time.sleep(2)
        request_random_mission()
except KeyboardInterrupt:
    print("Loop interrupted by user.")
```

Creating Application

Index.html run on the host machine

helyos_core url

Application token

To learn how to make a complete web app able to send requests and Visualize agents, visit the tutorial by Jiapang Wang:

<https://fraunhofer-ivi-helyos-frontend-example.readthedocs.io/en/latest/index.html>

```
<!DOCTYPE html>
<html>
<head>
    <title>Minimalistic Map with Floating Images - Leaflet</title>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="https://unpkg.com/leaflet@1.7.1/dist/leaflet.css" />
    <style>
        #map { height: 100vh; }
        /* Set the rotation origin to the center of the icon */
        .leaflet-rotated-icon { transform-origin: 50% 50%; }
    </style>
</head>
<body>

<div id="map"></div>

<script src="https://unpkg.com/leaflet@1.7.1/dist/leaflet.js"></script>
<script src="Leaflet.Marker.SlideTo.js"></script>
<script src="leaflet.rotatedMarker.js"></script>
<script src="https://cdn.socket.io/4.0.0/socket.io.min.js"></script>
<script>

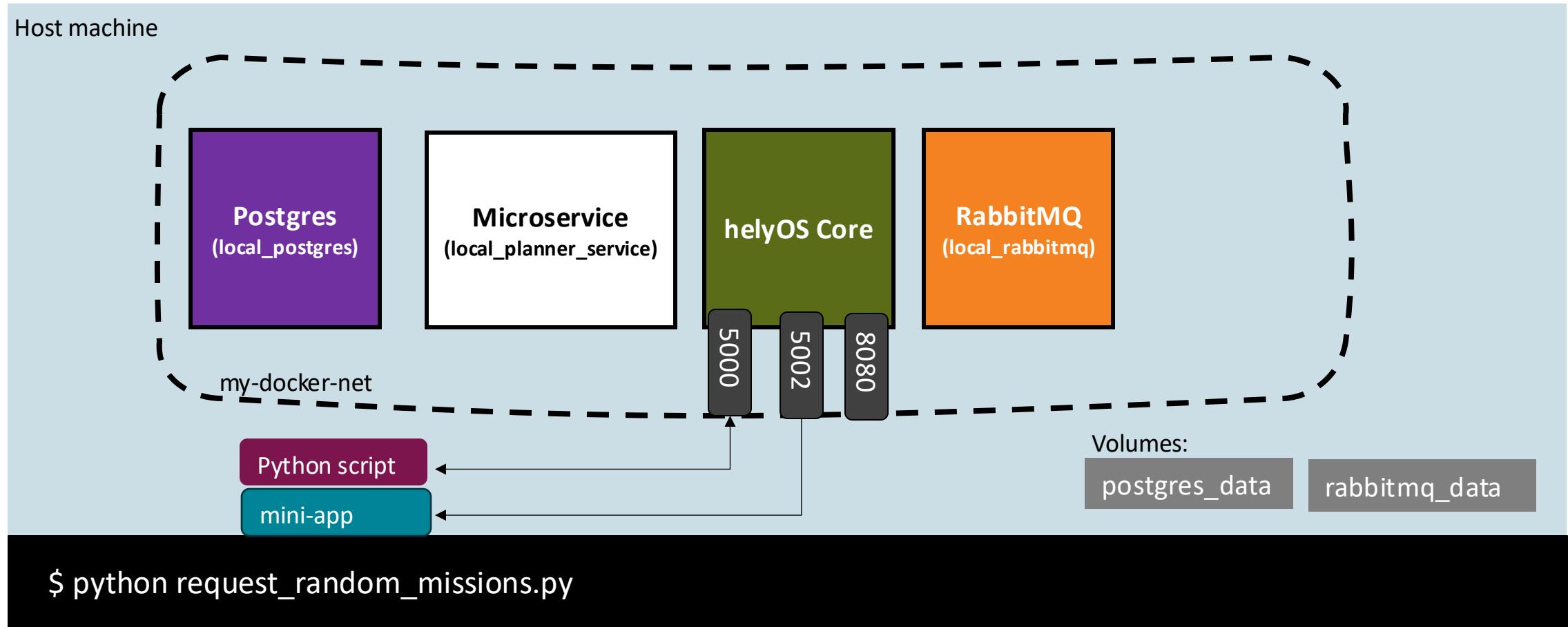
    ////////////// HELYOS CLIENT SETTINGS ///////////
    const socket = io('http://localhost:5002', [
        path: '/socket.io/',
        transports: ['websocket'],
        auth: {
            token: 'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJb2xlIjoicm9sZV9hcHBsaWNhdGlvbiIsInBlcnNvbl9pZC16bnVsbCwiZXhwIjoxNzY0NzYxNDU3LCJpYXQiOjE3MzM';
        }
    ]);

    ////////////// LEAFLET MAP SETTINGS ///////////
    const refPoint = [48.11398, 11.542205];
    const goalBlue = [-2013, 51588];
    const goalRed = [2499, -50177];
    const blueIcon = './tractor_blue.png';
    const redIcon = './tractor_red.png';
    const vehicleIconOption = (iconUrl, text) => ({
        iconUrl: iconUrl,
        iconSize: [15, 24], // size of the icon,
        className: 'leaflet-rotated-icon' // custom class for CSS rotation
    });
    // Create a label using L.divIcon
    var createLabelIcon = function(value) {
        return L.divIcon({
            className: 'label-icon',
            html: '<div style="position: relative; text-align: center;">' +
                '<div style="position: absolute; top: -25px; left: 50%; transform: translateX(-50%); font-size: 12px; color: black;">' + value + '</div>' +
                '</div>',
            iconSize: [0, 0] // Size of the icon, set to 0,0 to make it invisible
        });
    };

    const ballIconOption = {
        iconUrl: './ball_transp.png',
        iconSize: [20, 20], // size of the icon,
        className: 'leaflet-rotated-icon' // custom class for CSS rotation
    };
    const ballIcon = L.icon(ballIconOption);
    const vehicleRedIcon = L.icon(vehicleIconOption(redIcon));
    const vehicleBlueIcon = L.icon(vehicleIconOption(blueIcon));
```

Running the Application

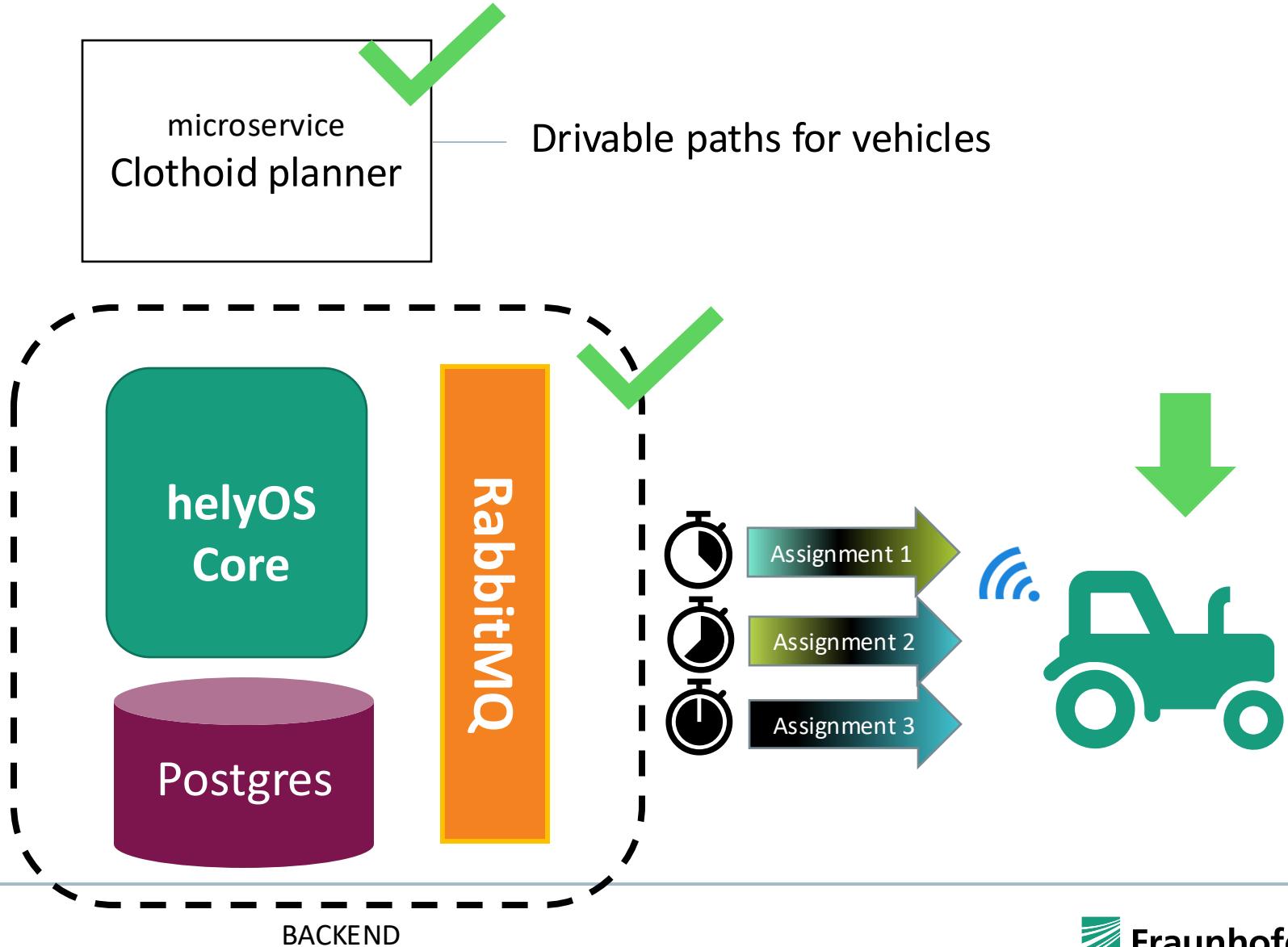
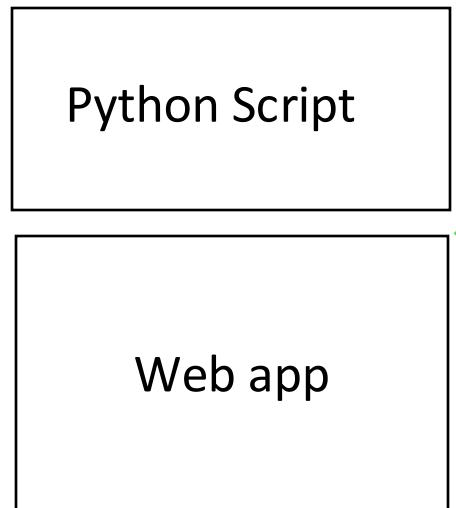
As a Docker container



The helyOS framework components



—
Your Applications.



Prepare an agent simulator

Use helyOS slim simulator!

```
services:

agent_simulator_1:
    image: helyosframework/helyos_agent_slim_simulator:0.8.2
    platform: linux/amd64
    environment:
        # AGENT ID
        - UUID=6b22b670-b185-11ef-8952-0242ac130008
        - ASSIGNMENT_FORMAT=trajectory
        - NAME=MY_TRUCK1
        - X0=-28000
        - Y0= 0
        - ORIENTATION=0.329
        - VELOCITY=10
        - YARD_UID=FTB
        - UPDATE_RATE=10
    # RABBITMQ
    - RBMQ_HOST=local_rabbitmq
    - RBMQ_PORT=5672
    - REGISTRATION_TOKEN=0001-0002-0003-0000-0004

networks:
    - my-docker-net

command: ["python", "-u", "main.py"]

networks:
    my-docker-net:
        external: true
```

helyOS Agent Slim Simulator

- Uses the *helyos_agent_sdk*
- Easy to customize.
- For development and tests.
- Run as docker container.

https://github.com/helyOSFramework/helyos_agent_slim_simulator

Prepare an agent simulator

Use helyOS slim simulator!

```
services:  
  
  agent_simulator:  
    image: helyosframework/helyos_agent_slim_simulator:0.8.2  
    platform: linux/amd64  
    environment:  
  
      # AGENT ID  
      - UUID=RANDOM_UUID ←  
      - ASSIGNMENT_FORMAT=trajectory  
      - NAME=MY_TRUCK2  
      - X0=28000  
      - Y0=0  
      - ORIENTATION=0.329  
      - VELOCITY=10  
      - YARD_UID=FTB  
      - UPDATE_RATE=10  
  
      # RABBITMQ  
      - RBMQ_HOST=local_rabbitmq  
      - RBMQ_PORT=5672  
      - REGISTRATION_TOKEN=0001-0002-0003-0000-0004 ←  
  
    networks:  
      - my-docker-net  
  
    command: ["python", "-u", "main.py"]  
    deploy:  
      replicas: 11 ←  
  
  networks:  
    my-docker-net:  
      external: true
```

Each container will create a random identifier.

Attention: Every time you start this docker, you will have more
11 agents registered in helyOS

Docker automatically creates replicas for you.

Running the Application

Index.html run on the host machine

```
1 import random, time
2 from helyos_requests import get_free_agents, create_work_process
3
4 # Define the GraphQL endpoint and headers
5 hostname = "http://localhost"
6 port = 5000
7 headers = {
8     "Content-Type": "application/json; charset=utf-8",
9     "Authorization": "Bearer eyJhbGciOiJIUzI1NiIsInR5cIi6IkpXVCJ9eyJyb2xlIjoicm9sZV9hcHBsaWNhdGlvbIisInBicnNvb19pZCI6bnVsbCwiZXhwIjoxNzY0NzYxNDU3LCj9YQjoi
10 }
11 graphql_conf = { 'url': f'{hostname}:{port}/graphql', 'headers': headers }
12
13 # Main function to request a random mission for a random free agent
14 def request_random_mission():
15     free_agents = get_free_agents(graphql_conf)
16     if not free_agents:
17         return
18
19     # Randomly select a free agent
20     for selected_agent in free_agents:
21         agent_uuid = selected_agent['uuid']
22         agent_yard_id = selected_agent['yardId']
23         current_pos = {'x':selected_agent['x'],
24                        'y':selected_agent['y'],
25                        'orientations':selected_agent['orientations']}
26
27         random_destination = { 'x':random.randint(-10000, 10000),
28                                'y':random.randint(-20000, 20000),
29                                'orientations':[random.randint(-6000, 6000)]}
30
31         request_data = {'agent_uuid': agent_uuid,
32                         'initial_position': current_pos,
33                         'destination': random_destination}
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

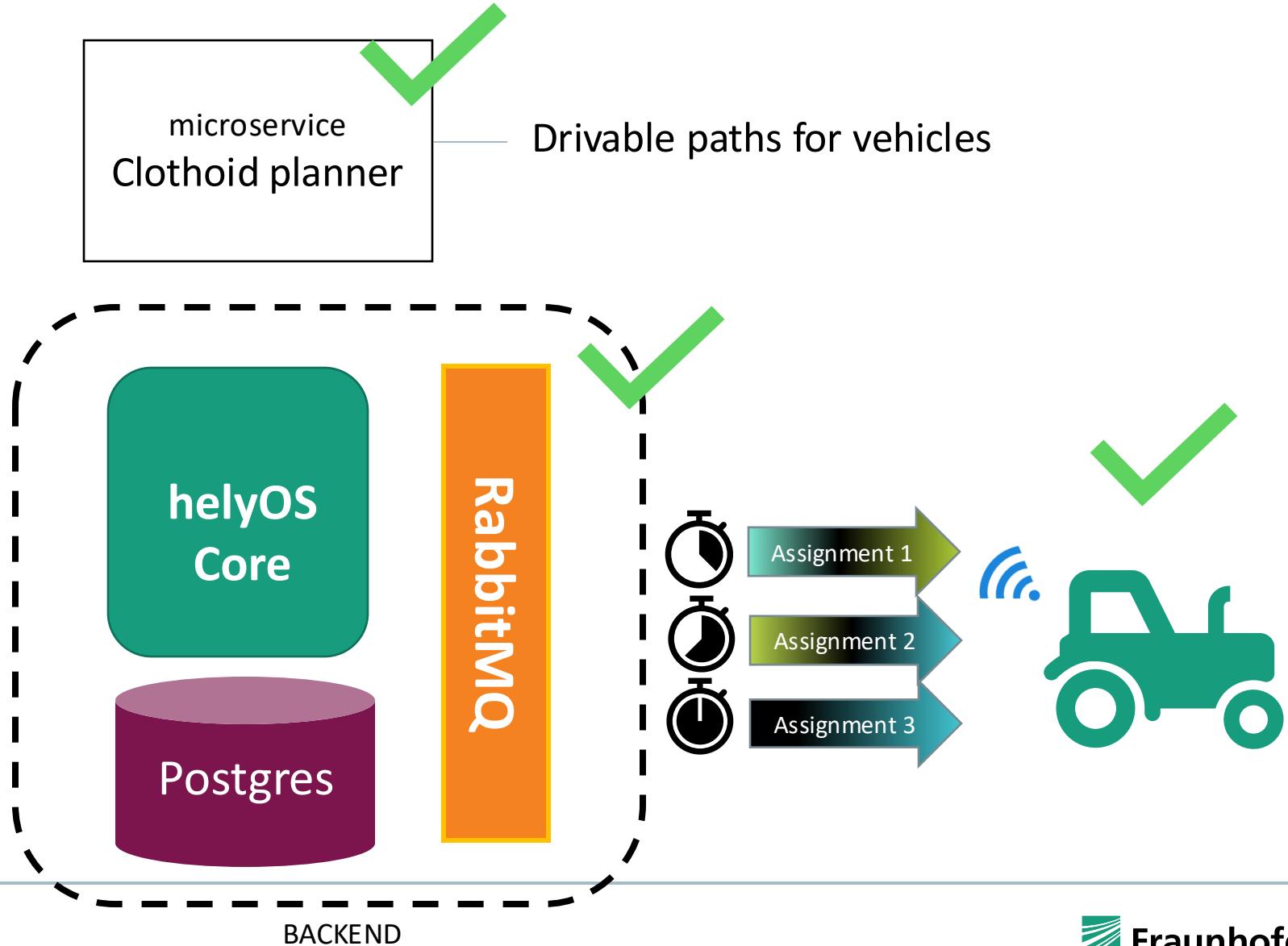
(myenvs) mac:python_script violbarbosas []



The helyOS framework components



—
Your Applications.



Migrate Your Installation to the Cloud



Prepare files

```
01-QUICK-DEMO
  agents
    docker-compose-agents.yml
  backend
    etc_helyos
    create-rsa-keys.sh
    docker-compose-backend.yml
  external_applications
    mini_app
    python_script
  microservices
    clothoid_path_planner
    build_python_image
    src
    docker-compose-microservice.yml
  README.md
```

Let's run this application in the computer
helyos-server.ivi.fraunhofer.de.

What should we change?

✓ In the **external_applications** and in the **agents**

HOSTNAME = "helyos-server.ivi.fraunhofer.de"
RBMQ_HOST = "helyos-server.ivi.fraunhofer.de"

✓ In the **docker-compose-backend.yml**

```
local_rabbitmq:
  image: rabbitmq:3-management
  hostname: local_rabbitmq
  volumes:
    - rbmq_data:/var/lib/rabbitmq/
  networks:
    - my-docker-net
  ports:
    - 15672:15672
    - 5672:5672
```

Accessible to external agents

Dev computer

```
└── 01-QUICK-DEMO
    ├── agents
    │   └── docker-compose-agents.yml
    ├── backend
    │   ├── etc_helyos
    │   ├── create-rsa-keys.sh
    │   └── docker-compose-backend.yml
    ├── external_applications
    │   ├── mini_app
    │   └── python_script
    └── microservices
        └── clothoid_path_planner
            ├── build_python_image
            ├── src
            └── docker-compose-microservice.yml
    └── README.md
```

Private Cloud computer

helyos-server.ivи.fraunhofer.de

User computer

Vehicle computer unit

Private Cloud computer

helyos-server.ivi.fraunhofer.de

```
└── backend
    ├── etc_helyos
    └── $ create-rsa-keys.sh
        └── docker-compose-backend.yml
```

```
└── microservices
    └── clothoid_path_planner
        ├── build_python_image
        ├── src
        └── docker-compose-microservice.yml
            └── README.md
```

5000 5002 8080 15672 5672

User computer

```
└── external_applications
    ├── mini_app
    └── python_script
```

Vehicle computer unit

```
└── agents
    └── docker-compose-agents.yml
```



Advancing Demo Applications Towards Real-World Scenarios

- Improving security
- Improving robustness

Improving security



Secure Dashboard and External Applications Connections

- Change the *admin* password.
- Use Nginx as a proxy to enable secure HTTPS connection

```
nginx_server:  
  image: nginx:stable-alpine  
  container_name: nginx_server  
  ports:  
    - "443:443"  
  volumes:  
    - ./etc_nginx/nginx.conf:/etc/nginx/nginx.conf:ro  
    - ./certificates:/etc/ssl:ro  
  networks:  
    - my-docker-net  
  
  depends_on:  
    - helyos_core
```

```
events {}  
http {  
  server {  
    listen 443 ssl;  
    server_name helyos-server.ivi.fraunhofer.de;  
  
    ssl_certificate /etc/ssl/server_certificate.pem;  
    ssl_certificate_key /etc/ssl/server_key.pem;  
  
    location /socket.io {  
      proxy_pass http://helyos_core:5002;  
      proxy_http_version 1.1;  
      proxy_set_header Upgrade $http_upgrade;  
      proxy_set_header Connection "Upgrade";  
      proxy_set_header Host $host;  
      proxy_set_header X-Real-IP $remote_addr;  
      proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;  
      proxy_set_header X-Forwarded-Proto $scheme;  
    }  
  
    location /graphiql {  
      proxy_pass http://helyos_core:5000/graphiql;  
      proxy_set_header Host $host;  
      proxy_set_header X-Real-IP $remote_addr;  
      proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;  
      proxy_set_header X-Forwarded-Proto $scheme;  
    }  
  
    location /graphql {  
      proxy_pass http://helyos_core:5000/graphql;  
      proxy_set_header Host $host;  
      proxy_set_header X-Real-IP $remote_addr;  
      proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;  
      proxy_set_header X-Forwarded-Proto $scheme;  
    }  
  
    location / {  
      proxy_pass http://helyos_core:8080;  
      proxy_set_header Host $host;  
      proxy_set_header X-Real-IP $remote_addr;  
      proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;  
      proxy_set_header X-Forwarded-Proto $scheme;  
    }  
  }  
}
```

Improving security

Secure Agents Connections

- Delete the RabbitMQ *guest* account.
- Delete the *AUTO-AGENT-REGISTRATION-TOKEN* in **helyos_core** environment variable
- Configure RabbitMQ to use TSL and use the port 15671 and 5673

```
local_rabbitmq:  
  # image: rabbitmq:3-management  
  build:  
    context: .  
    dockerfile: Dockerfile_RBMQ_with_certs ←  
  hostname: local_rabbitmq  
  volumes:  
    - ./etc_rabbitmq/enabled_plugins:/etc/rabbitmq/enabled_plugins:ro  
    - ./etc_rabbitmq/rabbitmq.conf:/etc/rabbitmq/rabbitmq.conf:ro ←  
    - rbmq_data:/var/lib/rabbitmq/  
  networks:  
    - my-docker-net  
  ports:  
    # - 15672:15672  
    # - 5672:5672  
    - 15671:15671 ←  
    - 5673:5673 ←
```

```
FROM rabbitmq:3-management  
# we cannot simply map the certificates in the docker-compose volumes  
# because rabbitmq is strict with the permissions.  
ADD ./certificates /etc/ssl/certs/  
RUN chown rabbitmq:rabbitmq /etc/ssl/certs \  
  && chown rabbitmq:rabbitmq /etc/ssl/certs/* \  
  && chmod 710 /etc/ssl/certs \  
  && chmod 610 /etc/ssl/certs/*
```

Name	Size	Kind
ssl	--	Folder
enabled_plugins	365 KB	Document
rabbitmq.conf	1 KB	Document

Private Cloud computer

helyos-server.ivi.fraunhofer.de

```
  > backend  
    > certificates  
    > etc_helyos  
    > etc_nginx  
    > etc_rabbitmq  
    $ create-rsa-keys.sh  
  ⚡ docker-compose-backend.yml  
  ⌂ Dockerfile_RBMQ_with_certs
```

15673

```
  > microservices  
    > clothoid_path_planner  
      > build_python_image  
      > src  
  ⚡ docker-compose-microservice.yml  
  ⓘ README.md
```

443

5000

5002

8080

15672

5673

```
  > external_applications  
    > mini_app  
    > python_script
```

User computer

```
  > agents  
  ⚡ docker-compose-agents.yml
```

Vehicle computer unit

Private Cloud computer

helyos-server.ivi.fraunhofer.de

15673

```
└── backend
    ├── certificates
    ├── etc_helyos
    ├── etc_nginx
    ├── etc_rabbitmq
    └── Dockerfile_RBMQ_with_certs
```

```
└── microservices
    └── clothoid_path_planner
        ├── build_python_image
        └── src
            ├── docker-compose-microservice.yml
            └── README.md
```

443

<https://helyos-server.ivi.fraunhofer.de>
<wss://helyos-server.ivi.fraunhofer.de>

```
└── external_applications
    ├── mini_app
    └── python_script
```

User computer

5673

<amqps://helyos-server.ivi.fraunhofer.de>

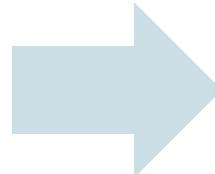
```
└── agents
    ├── docker-compose-agents.yml
```

Vehicle computer unit

Microservices

- Use a WSGI server and not the *development server*. I like *gunicorn* ☺

```
FROM python:3.8
WORKDIR /app
COPY ./build_python_image/requirements.txt .
RUN pip install -r requirements.txt
COPY src src
EXPOSE 9002:9002
CMD ["python", "./src/service.py"]
```



```
FROM python:3.8

WORKDIR /app
COPY ./build_python_image/requirements.txt .
RUN pip install -r requirements.txt
RUN pip install gunicorn
COPY src/ src/
EXPOSE 9002:9002

# Run the application using Gunicorn
WORKDIR /app/src
CMD ["gunicorn","-w","4","-b","0.0.0.0:9002", "service:app"]
```

Microservices

- Do not build your microservice in the server. Just pull its image from a docker registry.

helyos-server.ivи.fraunhofer.de

```
└── backend
    ├── certificates
    ├── etc_helyos
    ├── etc_nginx
    ├── etc_rabbitmq
    └── Dockerfile_RBMQ_with_certs
```

A large red X is overlaid on the directory structure of a microservice repository. The structure shown is:

```
└── microservices
    └── clothoid_path_planner
        ├── build_py
        ├── Dockerfile
        ├── src
        └── README.md
```

Microservices

- Do not build your microservice in the server. Just pull its image from a docker registry.

helyos-server.ivи.fraunhofer.de

```
└── backend
    ├── certificates
    ├── etc_helyos
    ├── etc_nginx
    ├── etc_rabbitmq
    └── $ create-rsa-keys.sh
        ↗ docker-compose-backend.yml
        ⇢ Dockerfile_RBMQ_with_certs
```

```
services:
  local_planner_service:
    image: my_registry/my_clotoid_planner:vs1.0
    restart: always
    container_name: local_planner_service
    networks:
      - my-docker-net
    # ports:
    #   - "9002:9002"  # optional
networks:
  my-docker-net:
    external: true
```

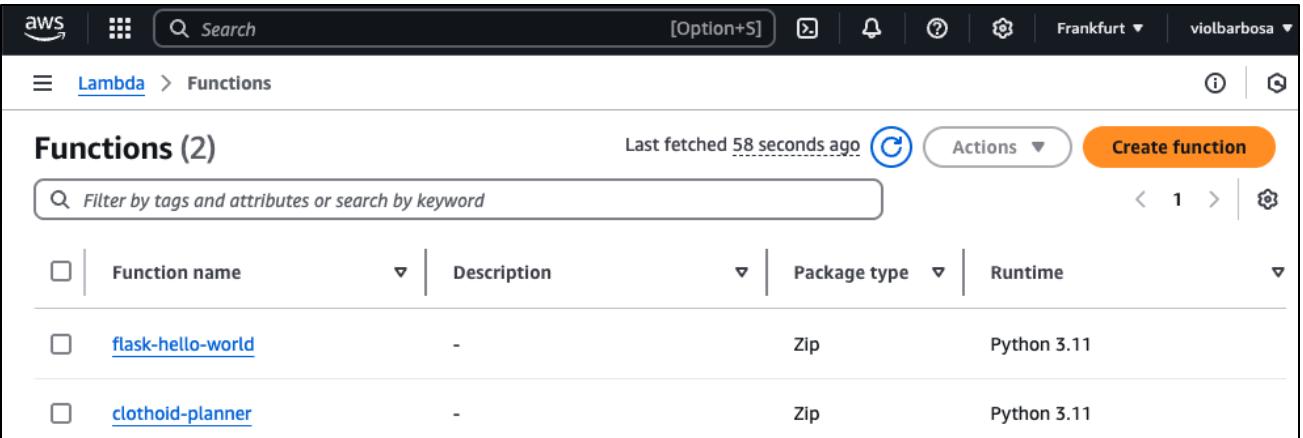


Improving robustness

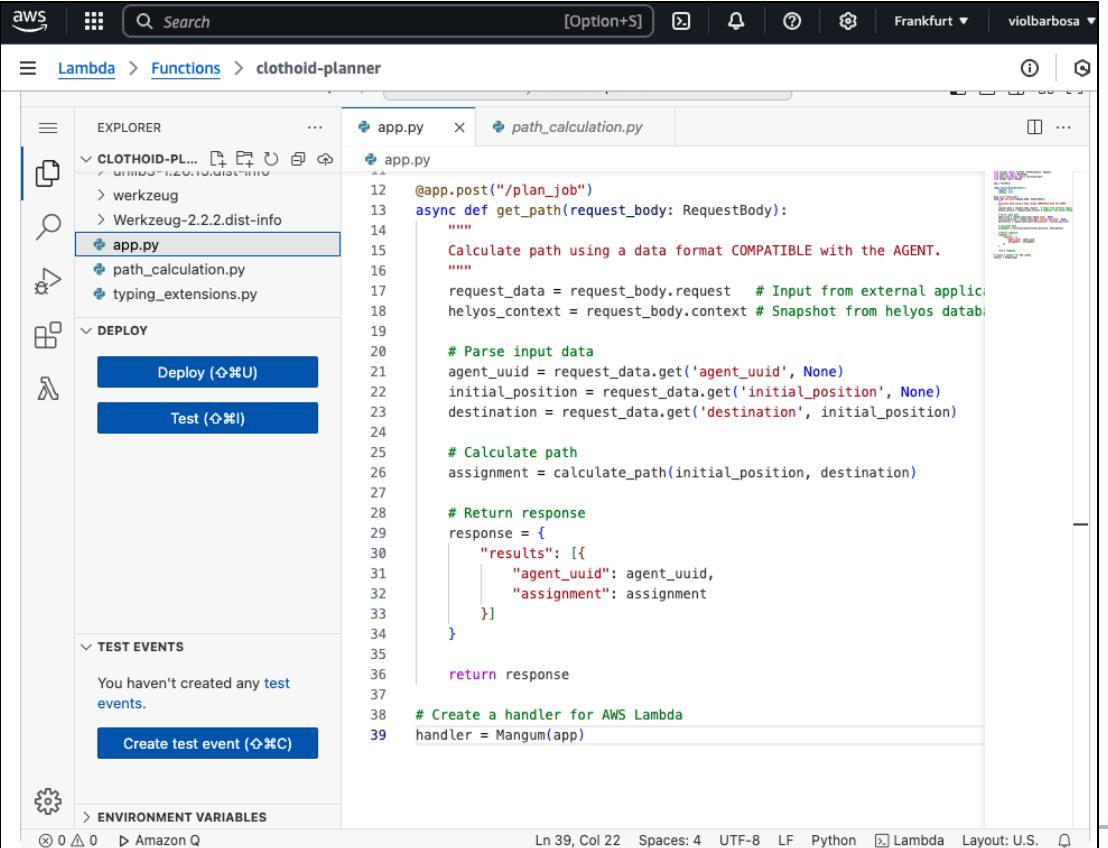


Microservices

- Create an extra server to host collections of shared microservices. If you do not want to host them, you can use a service like the AWS Lambda.



The screenshot shows the AWS Lambda Functions console. It displays two functions: "flask-hello-world" and "clothoid-planner". Both functions are in "Zip" package type and "Python 3.11" runtime. The "clothoid-planner" function was last fetched 58 seconds ago. There is a "Create function" button at the top right.



The screenshot shows the AWS Lambda Function code editor for the "clothoid-planner" function. The code is written in Python and uses the ASGI framework. It includes imports for Werkzeug, path calculation, and typing extensions. The main logic involves calculating a path based on input data and returning a response. A "Deploy" button is visible in the sidebar.

```
app.py
12 @app.post("/plan_job")
13 async def get_path(request_body: RequestBody):
14     """
15     Calculate path using a data format COMPATIBLE with the AGENT.
16     """
17     request_data = request_body.request # Input from external application
18     helyos_context = request_body.context # Snapshot from helyos database
19
20     # Parse input data
21     agent_uuid = request_data.get('agent_uuid', None)
22     initial_position = request_data.get('initial_position', None)
23     destination = request_data.get('destination', initial_position)
24
25     # Calculate path
26     assignment = calculate_path(initial_position, destination)
27
28     # Return response
29     response = {
30         "results": [
31             {
32                 "agent_uuid": agent_uuid,
33                 "assignment": assignment
34             }
35         ]
36     }
37
38     return response
39
# Create a handler for AWS Lambda
handler = Mangum(app)
```

Improving robustness

helyOS core

- Use Multi-threads to provide redundancy in case of failure.

```
helyos_core:  
  image: helyosframework/helyos_core:2.2.0-rc1  
  
  ports:  
    - 5002:5002 # WebSocket: Push Notifications.  
    - 5000:5000 # GraphQL: Communicate with External Applications  
    - 8080:8080 # HelyOS Dashboard: Settings and Monitoring  
  
  volumes:  
    - ./etc_helyos/:/etc/helyos/:ro  
  
  environment:  
    # MULTI-THREADING  
    - NUM_THREADS=4  
    - REDIS_HOST=redis.com  
    - REDIS_PORT=6379  
    - REDIS_PASSWORD=myp@ss,  
    # # #  
  
    # DATABASE  
    - PGUSER=helyos_db_admin
```

Configuration files

- Remove all passwords from the Docker Compose files and use .env files or other secure methods, such as secret management tools, to store and manage passwords.

```
helyos_core:  
  image: helyosframework/helyos_core:2.2.0-rc1  
  
  ports:  
    - 5002:5002 # WebSocket: Push Notifications.  
    - 5000:5000 # GraphQL: Communicate with External Applications  
    - 8080:8080 # HelyOS Dashboard: Settings and Monitoring for Developers  
  
  volumes:  
    - ./etc_helyos/:/etc/helyos/:ro  
  
  environment:  
    # DATABASE  
    - PGUSER=helyos_db_admin  
    - PGPASSWORD=${PGPASSWORD}  
    - JWT_SECRET=${PGPASSWORD}  
    - PGHOST=local_postgres  
    - PGDATABASE=smartfarm_db  
    - PGPORT=5432  
    - GQLPORT=5000  
  
    # RABBITMQ  
    - RBMQ_HOST=local_rabbitmq  
    - RBMQ_PORT=5672  
    - RBMQ_API_PORT=15672  
    - RBMQ_SSL=False      # For TLS support  
    - RBMQ_API_SSL=False # For TLS support  
  
    # RBMQ ACCOUNTS  
    - CREATE_RBMQ_ACCOUNTS=True  
    - RBMQ_ADMIN_USERNAME=helyos_rabbitmq_admin # Set it if CREATE_RBMQ_ACCOUNTS is True  
    - RBMQ_ADMIN_PASSWORD=${RBMQ_PASSWORD}      # Set it if CREATE_RBMQ_ACCOUNTS is True  
    # - AGENT_AUTO_REGISTER_TOKEN=0001-0002-0003-0000-0004 # Deleted for production  
  
  networks:  
    - my-docker-net
```

02-single-tenant > backend > ⚒ .env

- 1 PGPASSWORD=helyos_secret
- 2 RBMQ_PASSWORD=helyos_rb_secret
- 3

Configuration files

- Export save and version the configuration files *microservice.yml* and *mission.yml* every time you modify it in the dashboard.

microservice.yml

```
version: '2.0'

services:

  my_path_planner:
    is_dummy: false
    domain: "Assignment planner" # Assignment planner
    type: "drive"
    url: http://local_planner_service:9002/plan_job/
    enable: true
    apikey: "CN783V9SygdG0deHgfesdfsaeNuCqwbm"
    timeout: 300
    description: "Simple clothoid path planner"

    context:
      map_data: true
      all_agents_data: false
      mission_agents_data: true
```

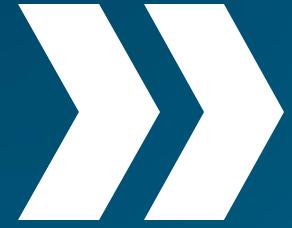
mission.yml

```
version: '2.0'

missions:

  driving:
    maxagents: 1
    description: "drive from a to b"

    recipe:
      steps:
        - step: "microservice1"
          service_type: "drive"
          request_order: 1
          apply_result: true
```



Conclusion

- We learned how to install helyOS and develop Demo Applications.
- We learned how to transfer our Demo Application to the cloud.
- We learned the steps to improve our application for a real-world scenario.

Useful links

- Official page: <https://www.helyosframework.org>
- helyOS Manual: <https://helyos-manual.readthedocs.io>
- helyOS WebApp Tutorial: <https://fraunhofer-ivi-helyos-frontend-example.readthedocs.io>
- helyOS AWS Deployment: <https://github.com/cviolbarbosa/aws-logistics-center-demo>
- Questions: https://github.com/helyOSFramework/helyos_core/discussions

Thank you.

violbarbosa@ivi.fraunhofer.de