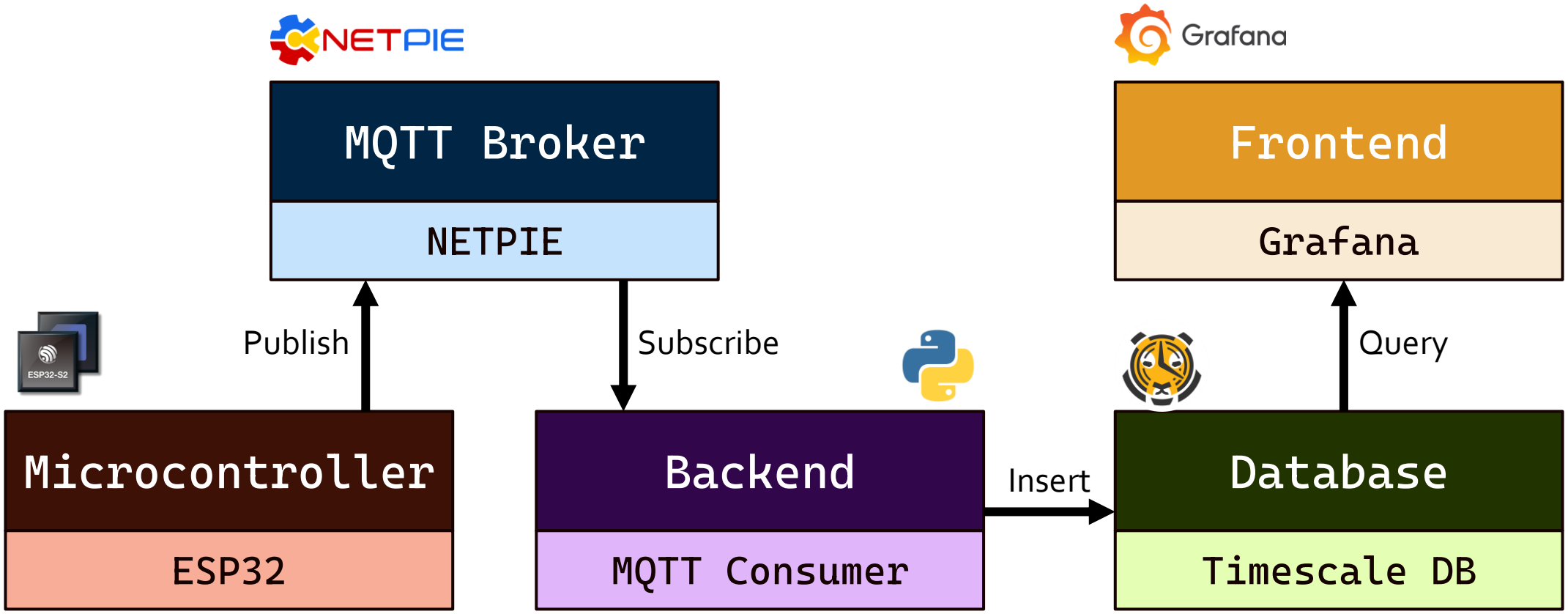
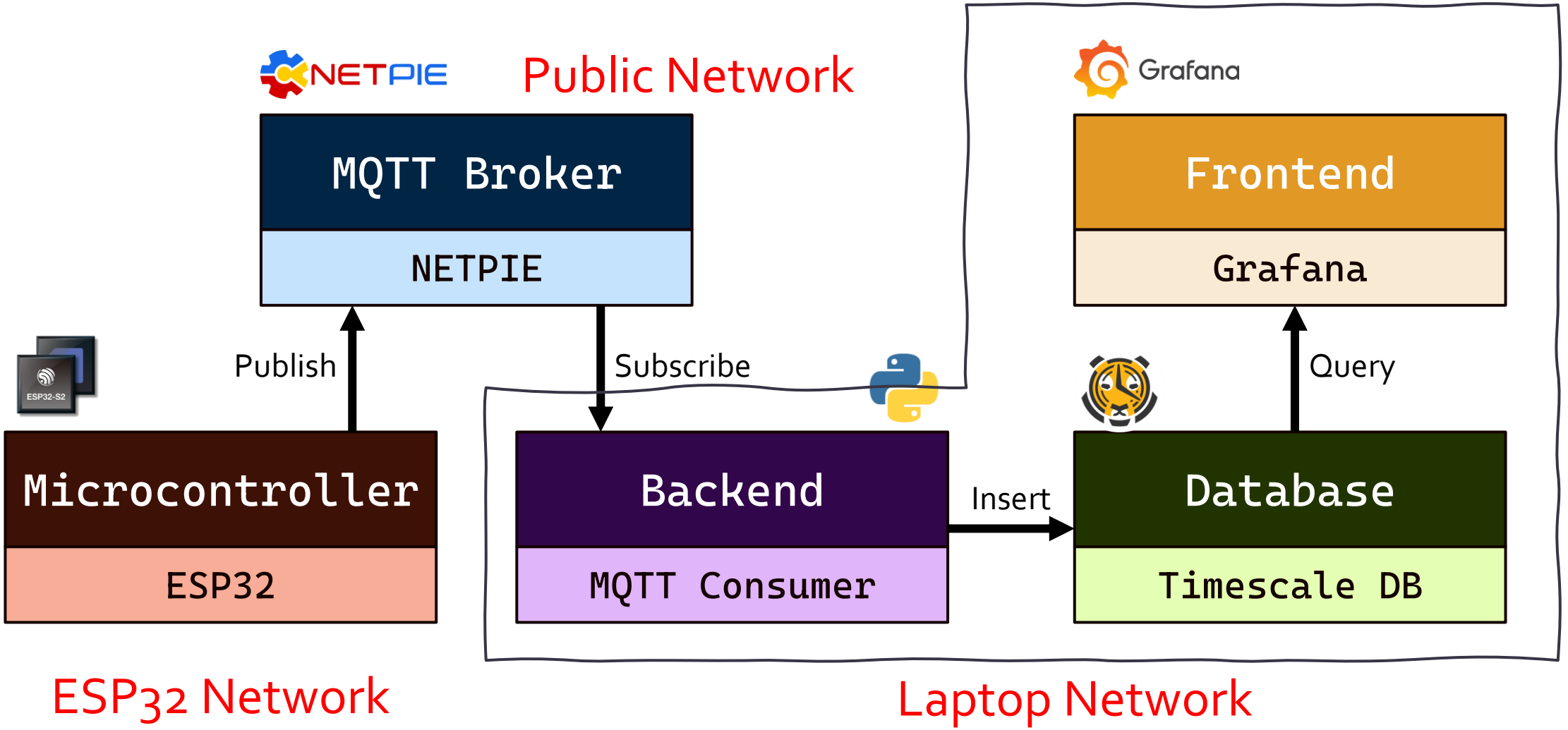
IoT Interns: Basic Networking

4 June 2025

When is networking required?

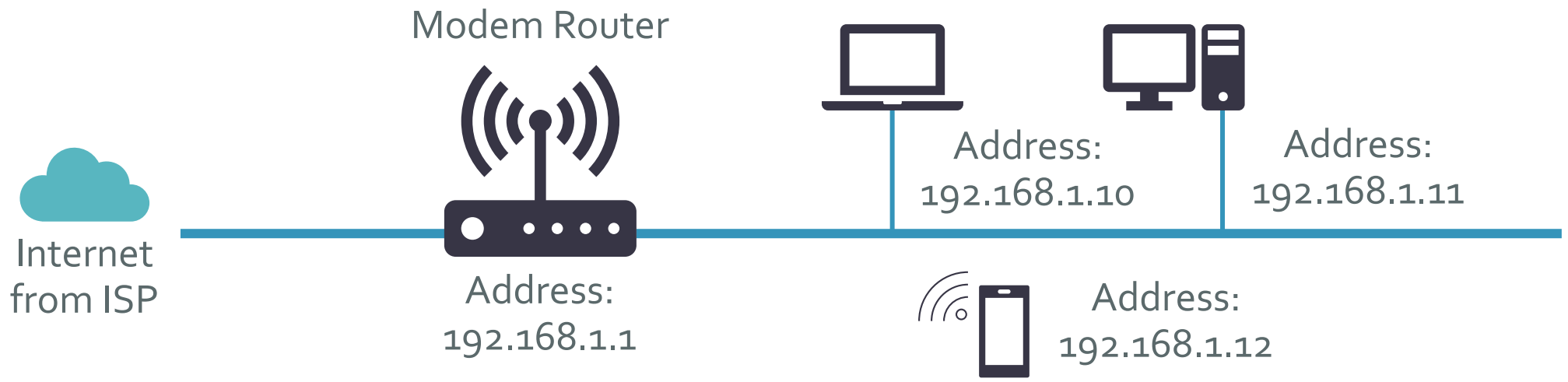


When is networking required?



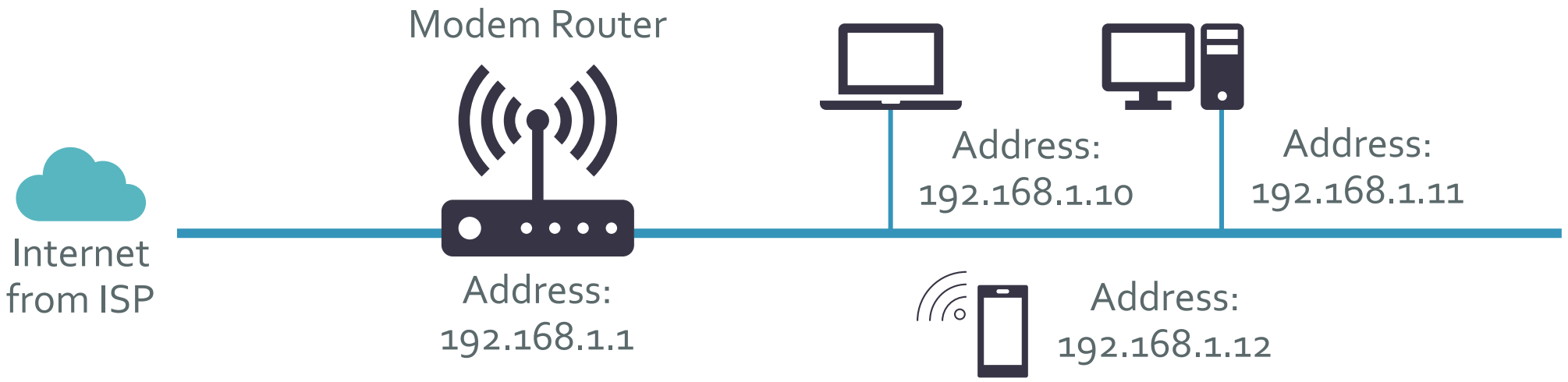
Network

- Group of connected devices that can communicate to each other
- Can be wire or wireless

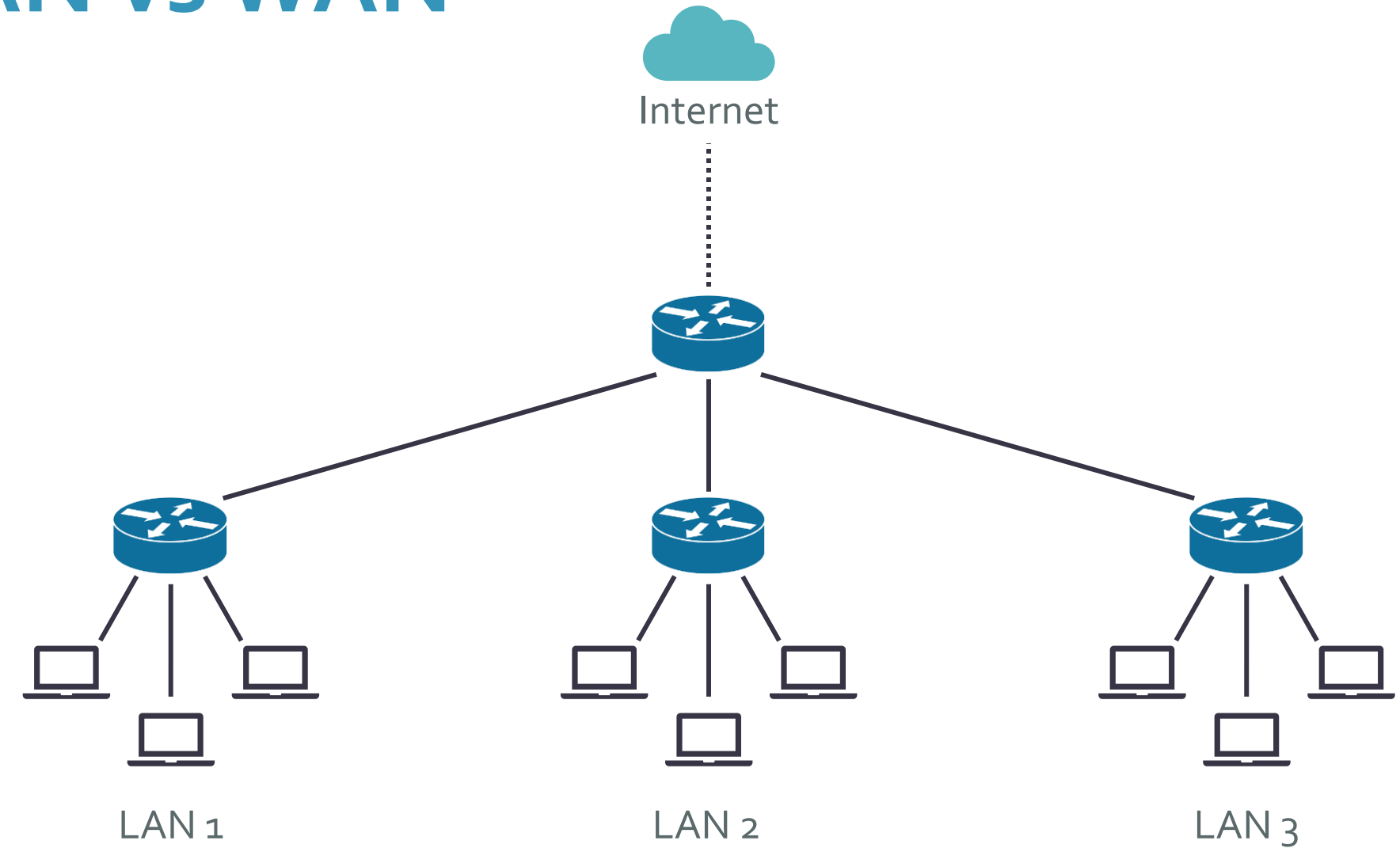


LAN vs WAN

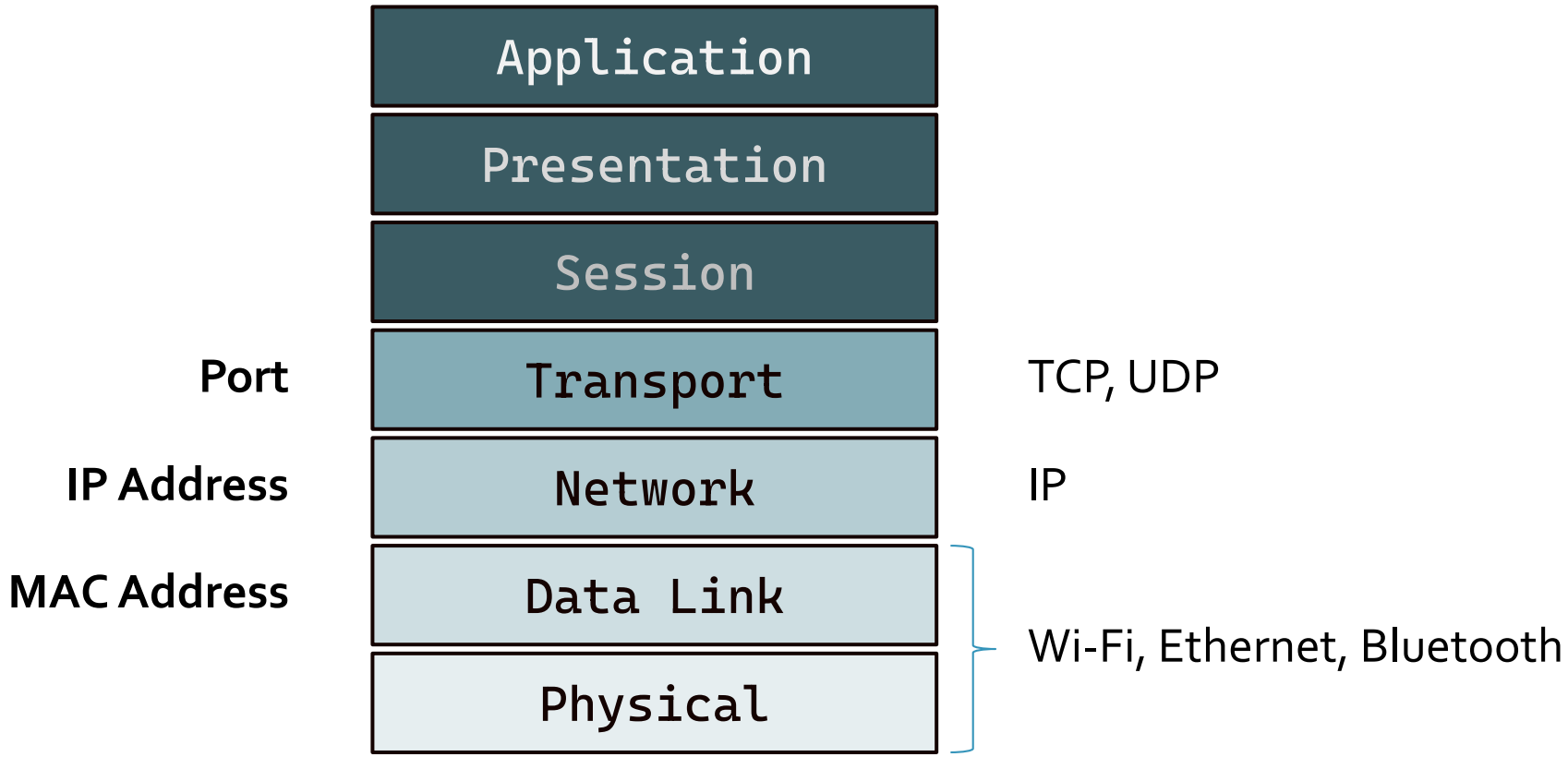
- **LAN** (**Local** Area Network) – Internal private network (e.g., home, office)
- **WAN** (**Wide** Area Network) – Network that connects LANs over long distances (e.g., the internet)



LAN vs WAN



OSI Layer

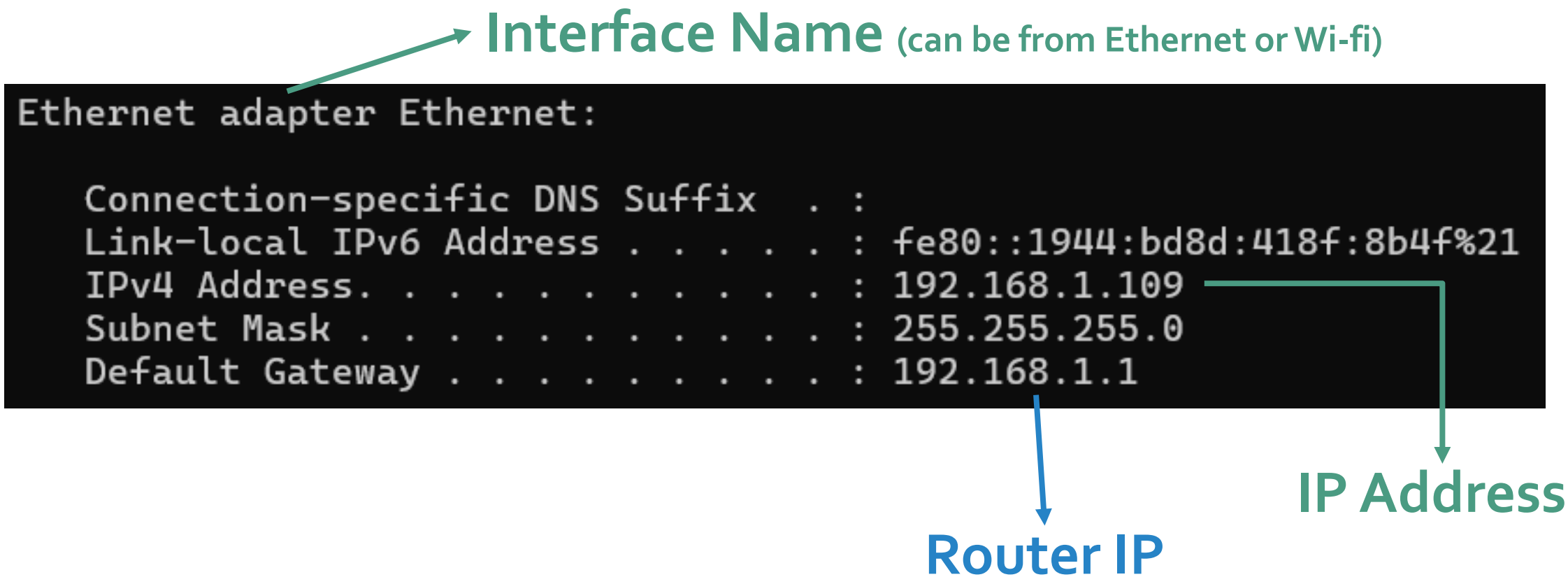

Application
Presentation
Session
Transport
Network
Data Link
Physical
Port
IP Address
MAC Address
TCP, UDP
IP
Wi-Fi, Ethernet, Bluetooth

IP Address

- Layer 3 Unique Identifier
- IPv4 consists of 4 bytes (e.g., 192.168.0.123)

- Check your own IP address
  - Window: open command prompt >> ***ipconfig***
  - MAC: open terminal >> ***ifconfig***

Check your IP



Public IP vs Private IP

Public IP

- Can be accessed from anywhere (if not firewalled)
- Globally unique and assigned by ISPs
- Example: 8.8.8.8 (Google DNS)

Private IP

- Not routable on the internet directly
- Used inside local networks (LAN)
- Must be translated (**NAT**) to a public IP to access the internet
- Common Range:
  - 192.168.xxx.xxx
  - 10.xxx.xxx.xxx
  - etc.

Test the reachability of connected devices with ping

- To check if you can connect to the internet

ping 8.8.8.8

- To verify if you are on the same LAN as my machine, ping the IP address:

ping 192.168.x.x

DNS (Domain Name System)

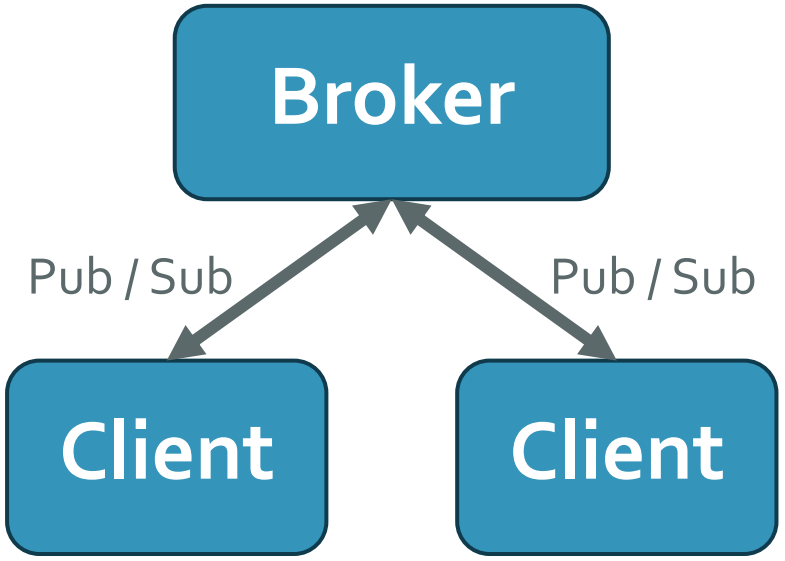
- www.google.com is domain name but behind it are actual IP addresses such as
  - 172.217.26.68
  - 142.250.199.46
  - 74.125.200.106
  - *In cmd:* *nslookup google.com*

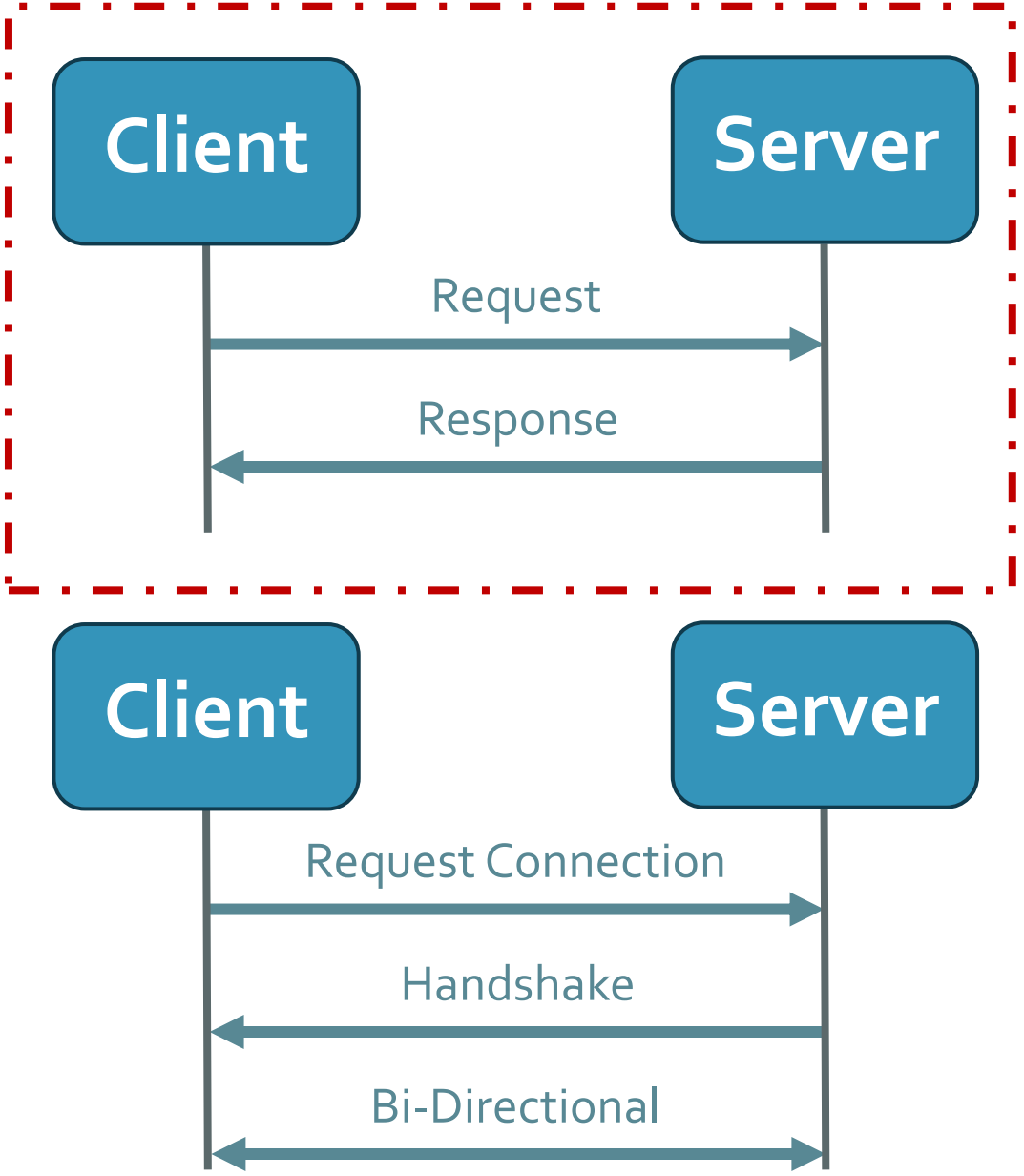
IoT Interns: API Server

4 June 2025

Previously: MQTT

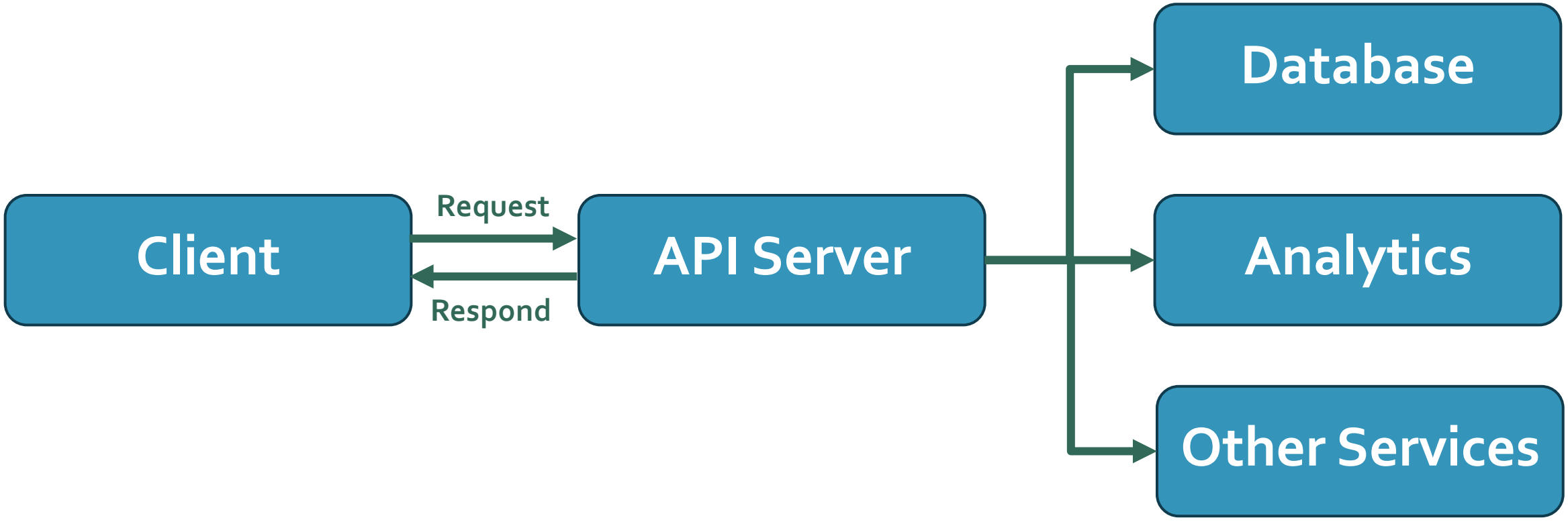
- **Publish / Subscribe model**
  What else can it be?



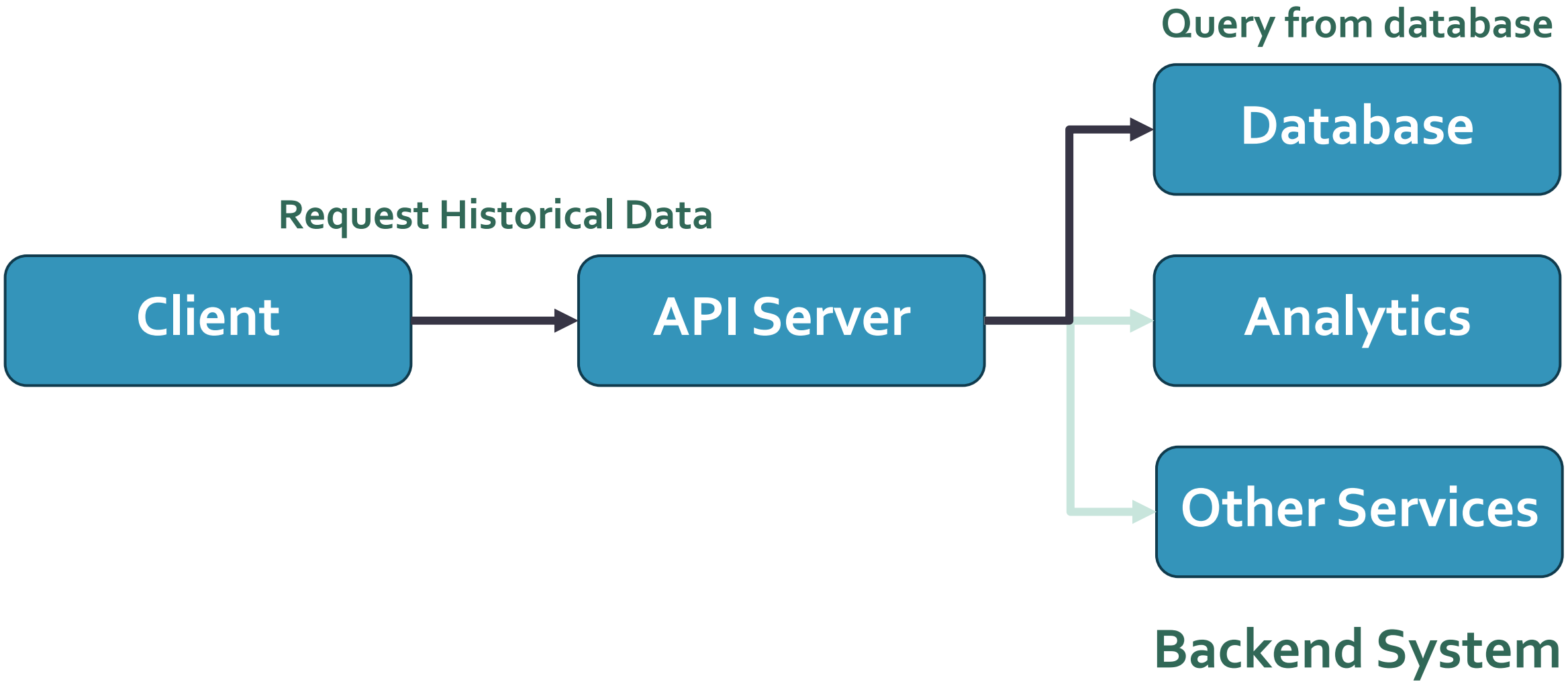


https://www.wallarm.com/what/websocket-vs-http-how-are-these-2-different

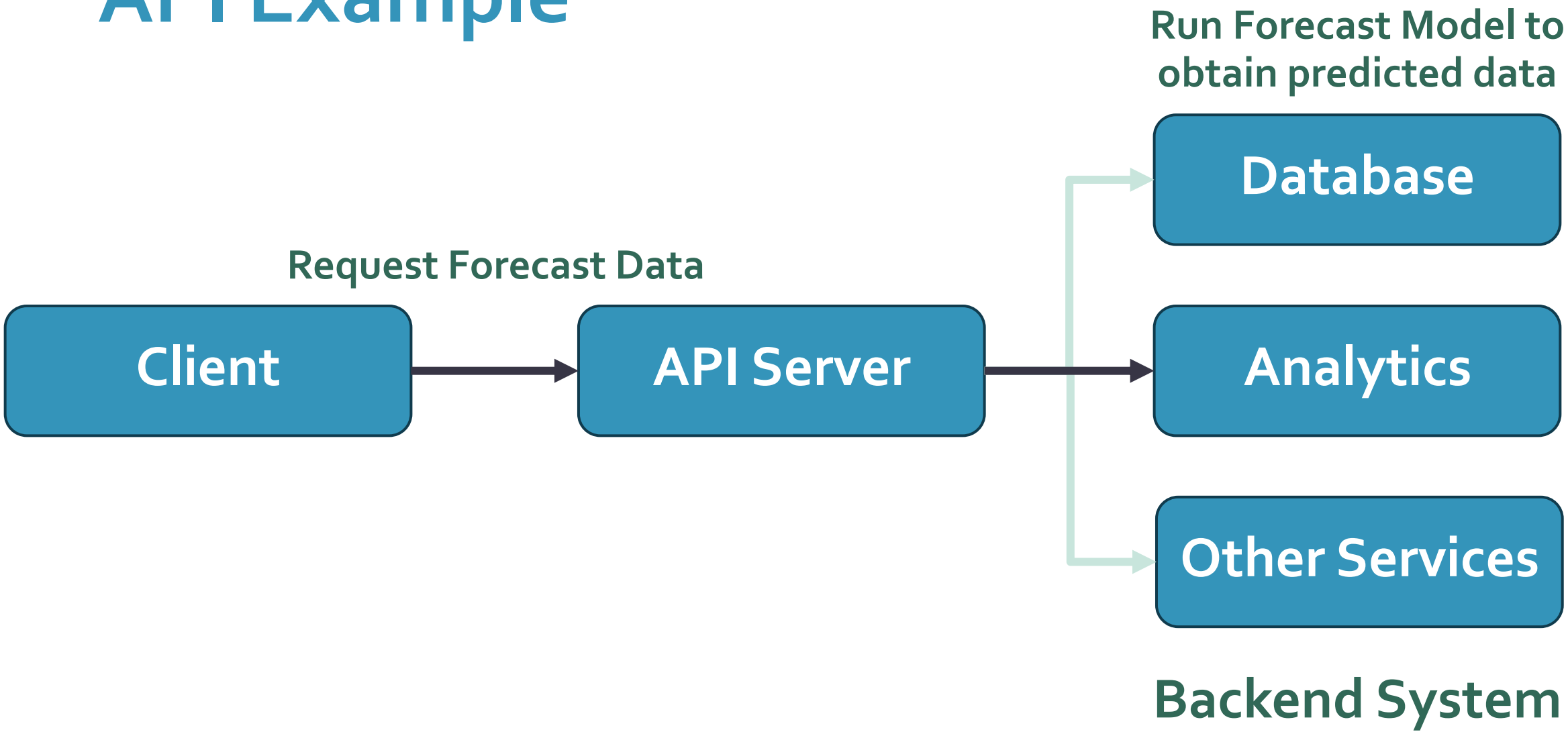
API (Application Programming Interface)



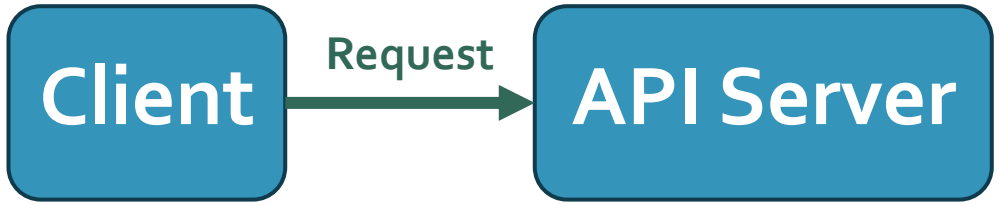
API Example


Query from database
Database
Request Historical Data
Client
API Server
Analytics
Other Services
Backend System

API Example


Run Forecast Model to obtain predicted data
Database
Request Forecast Data
Client
API Server
Analytics
Other Services
Backend System

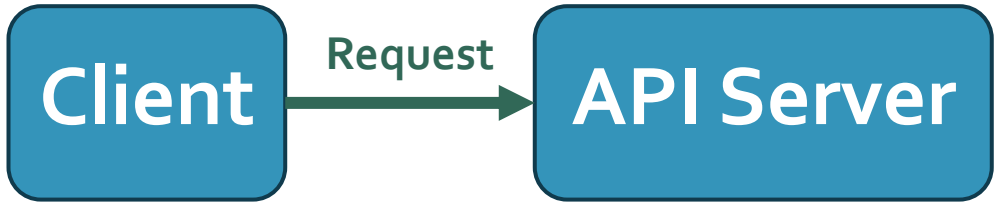
HTTP Requests



RESTful Convention

- GET – Retrieve Data idempotent
- POST – Create Data not idempotent
- PUT – Update Data idempotent
- DELETE – Remove Data idempotent

HTTP Requests



- **POST /users**
  Content-Type: application/json
  { "name" : "Film",
  "email" : "film@something.com" }

Not Idempotent

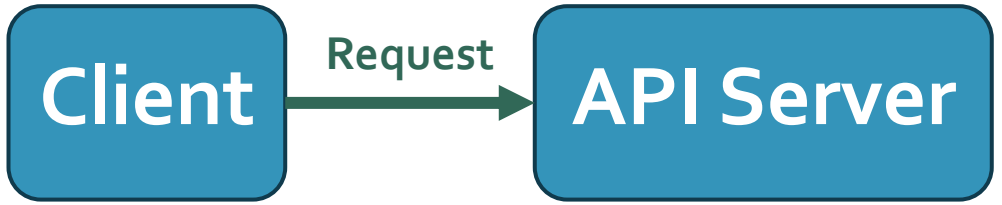
Create new user "Film" with user id: 1

- **GET /users/1**

Idempotent

Get the value of user "Film"

HTTP Requests



- **PUT /users/1**
  Content-Type: application/json
  { "name“ : “Film",
  "email“ : “patchapong@something.com" }

Idempotent

Update email of user “Film”

- **DELETE /users/1**

Idempotent

Delete the value of user “Film”

JSON format (JavaScript Object Notation)

- Like Dictionary in Python (Key-Value)
- Example

```
{
        "name":     "Patchapong",
        "age":      24,
        "email":    “film@something.com",
        "skills":   ["Python", "Flask", "IoT"],
        "is_active": true
}
```

Flask

Flask

- lightweight *Web Server Gateway Interface*(WSGI) web application framework
- Python-based

- It can be used to develop your own API Server

https://flask.palletsprojects.com/en/stable/

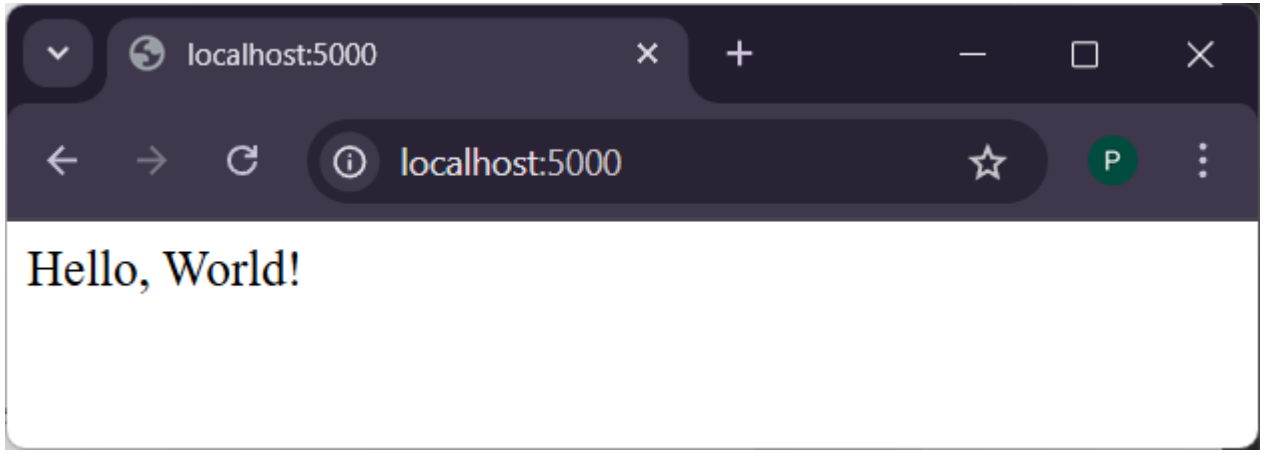
A Minimal Application

```
from flask import Flask
app = Flask(__name__)

@app.route("/")
def hello_world():
        return "Hello, World!"

if __name__ == '__main__':
        app.run(host='0.0.0.0', port=5000, debug=True)
```

https://flask.palletsprojects.com/en/stable/quickstart/#a-minimal-application

Run your first API server

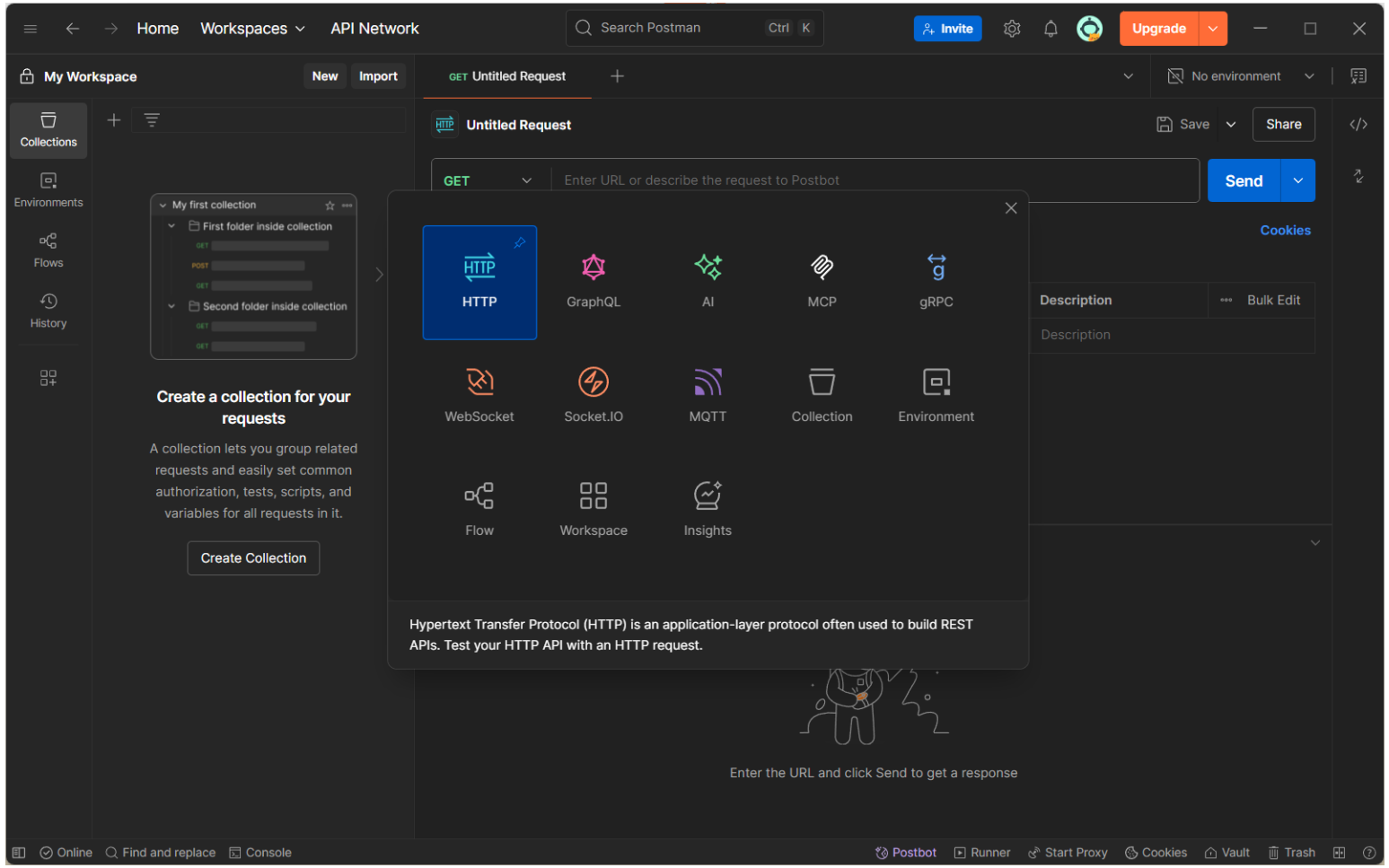
- Run the file: python [path of your python file]
- Check what's happening on http://localhost:5000



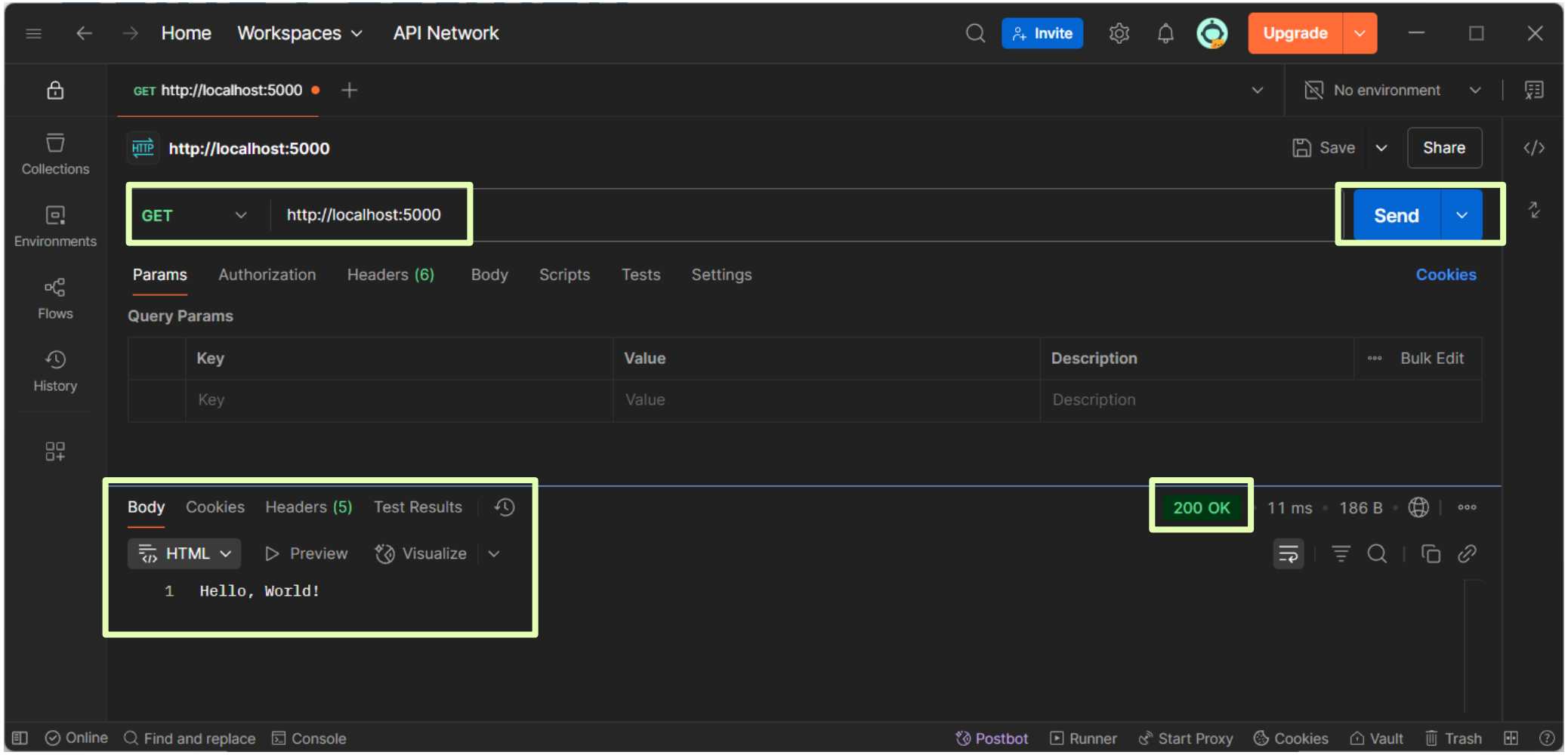

```
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://192.168.1.109:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 210-451-101
127.0.0.1 - - [03/Jun/2025 09:11:38] "GET / HTTP/1.1" 200 -
```

Opening a URL in the browser is equivalent to making a GET request.
Alternatively, you can use **Postman** to send HTTP requests

Using Postman

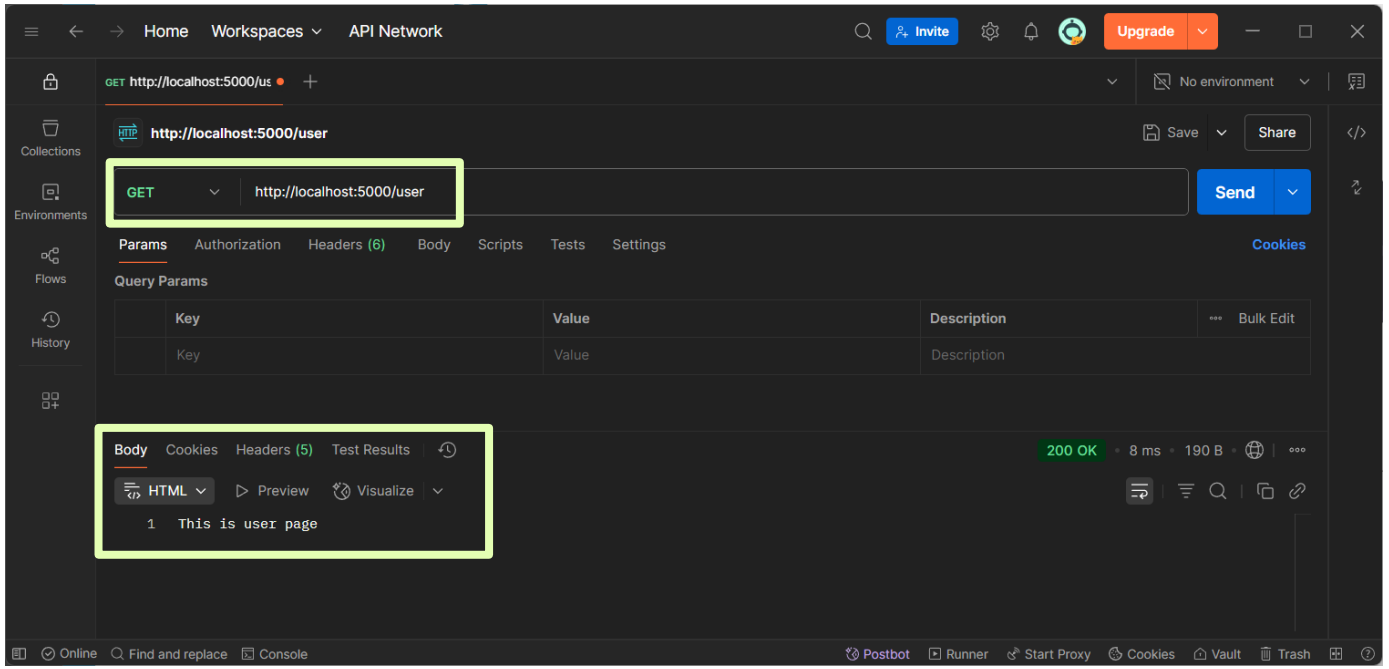


Using Postman



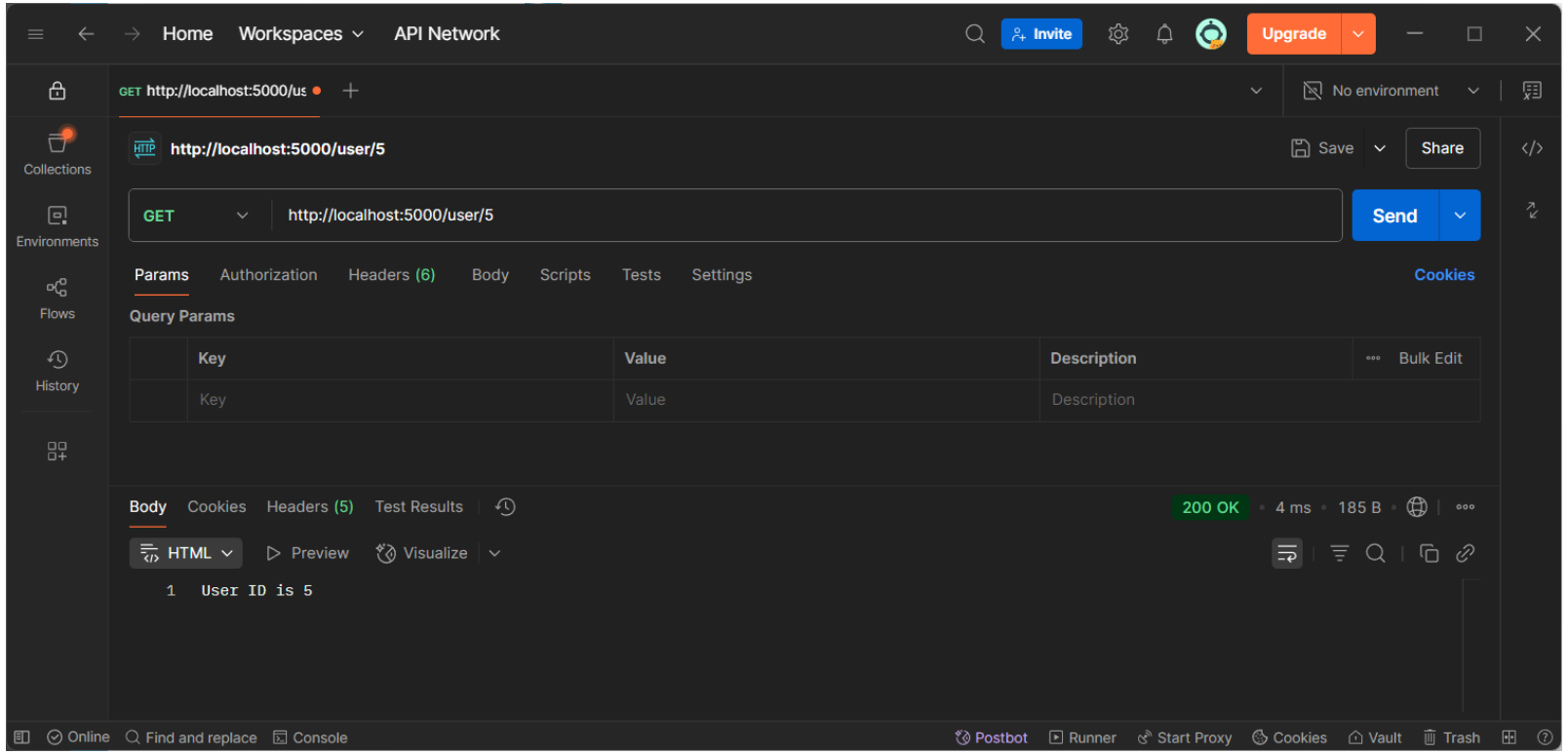
Adding more route

```python
from flask import Flask
app = Flask(__name__)

@app.route("/")
def hello_world():
  return "Hello, World!"

@app.route("/user")
def user_page():
  return "This is user page"

if __name__ == '__main__':
  app.run(host='0.0.0.0', port=5000, debug=True)
```



Parsing variable via route

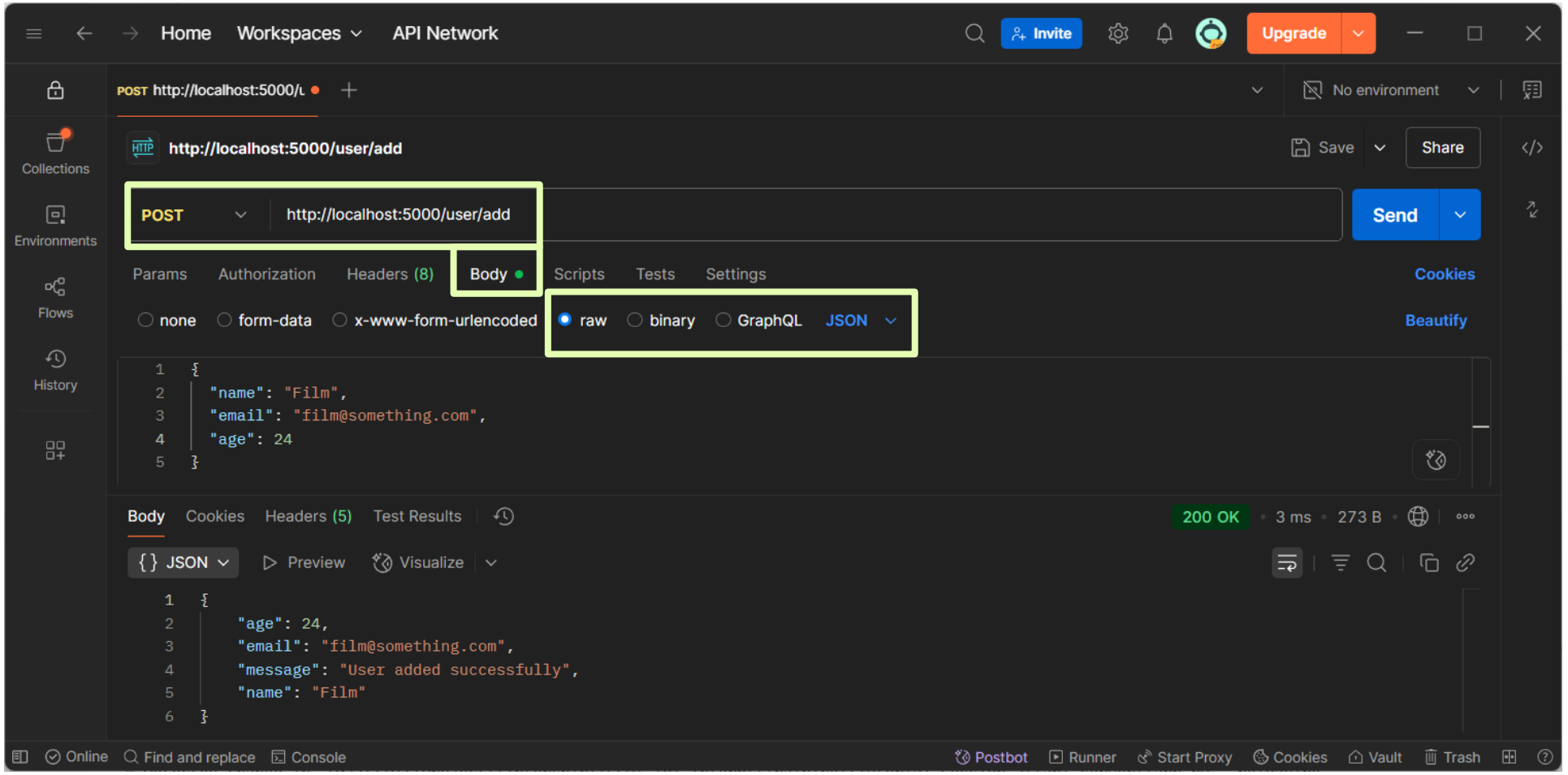
```
@app.route('/user/<int:user_id>')
def show_user(user_id):
    return f"User ID is {user_id}"
```



Add POST Request

```
from flask import Flask, request, jsonify

@app.route('/user/add', methods=['POST'])
def add_user():
  data = request.get_json() # Get JSON data from request
  name = data.get('name')
  email = data.get('email')
  age = data.get('age')

  return jsonify({
         'message': 'User added successfully',
         'name': name,
         'email': email,
         'age': age
  })
```

Use Postman to send POST request



More API Example

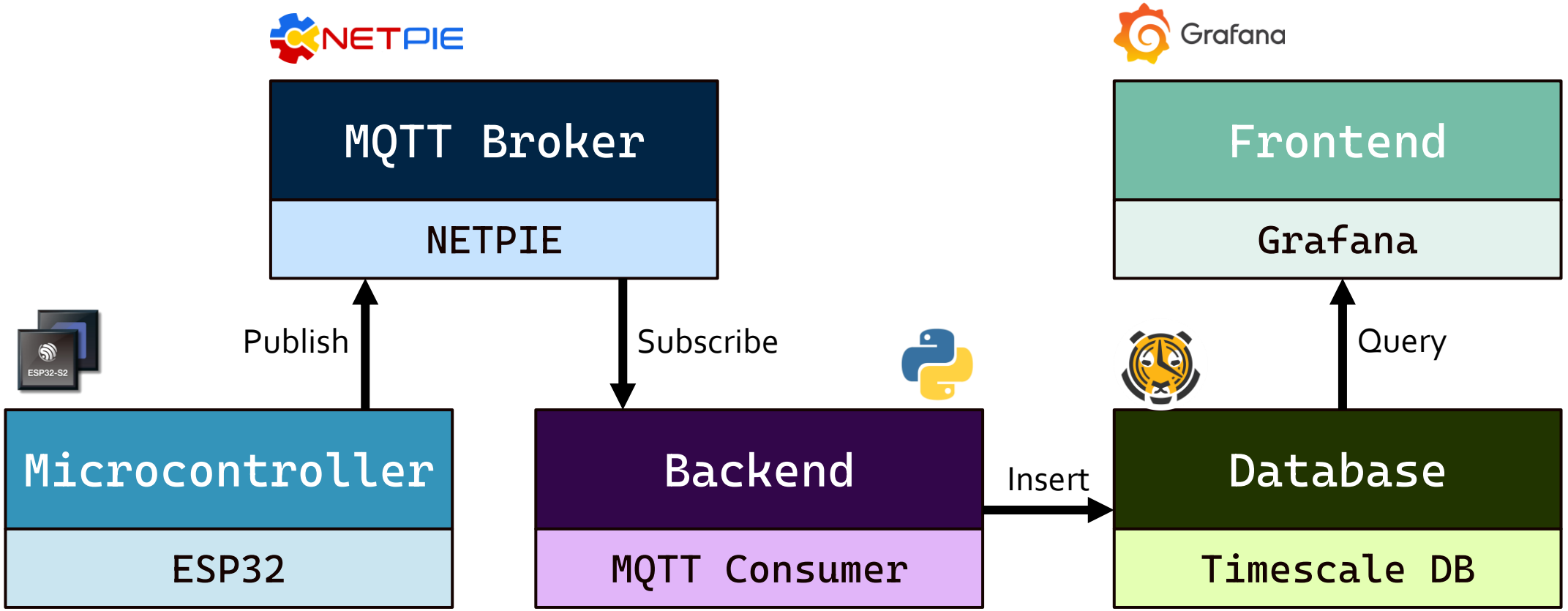
You can create custom API routes to run logic like calculations or database queries.

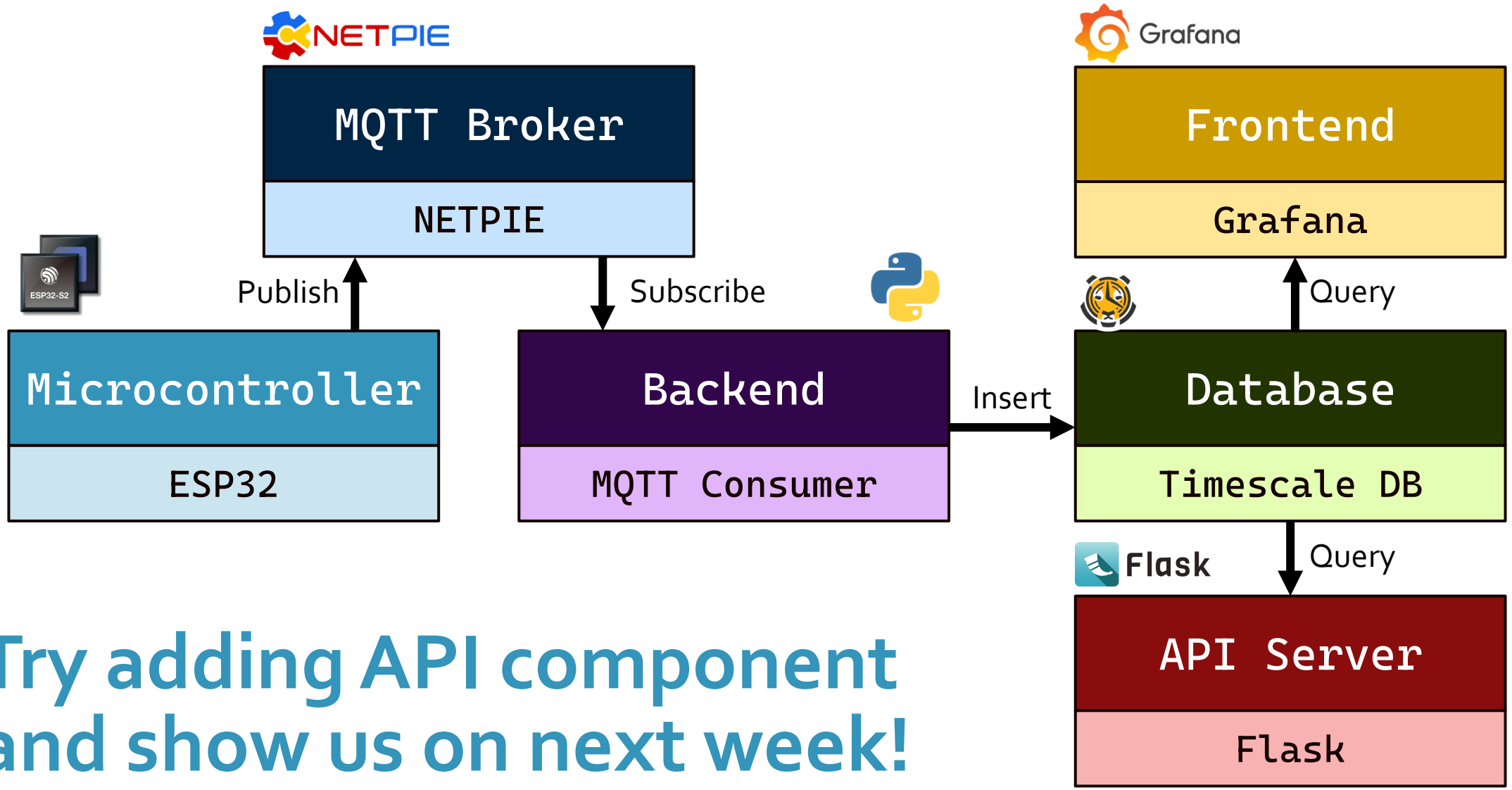
Try sending this payload to that route

```
{
  "date_of_birth": "yyyy-mm-dd"
}
```

```python
from datetime import datetime, date

@app.route('/user/birth', methods=['POST'])
def calculate_birth_info():
    data = request.get_json()
    dob_str = data.get('date_of_birth') # Expected format: YYYY-MM-DD

    try:
            dob = datetime.strptime(dob_str, '%Y-%m-%d').date()
            today = date.today()
            # Age in years
            age = today.year - dob.year - ((today.month, today.day) < (dob.month, dob.day))
            # Days and months since birth
            days_since_birth = (today - dob).days
            months_since_birth = (today.year - dob.year) * 12 + today.month - dob.month
            if today.day < dob.day:
                    months_since_birth -= 1
            # Day of the week and month name
            weekday = dob.strftime('%A')            # e.g., Sunday
            month_name = dob.strftime('%B')         # e.g., June
            birth_year = dob.year
            born_day = dob.day

            return jsonify({
                    'age': age,
                    'days_since_birth': days_since_birth,
                    'months_since_birth': months_since_birth,
                    'day_of_week_born': weekday,
                    'day': born_day,
                    'birth_month_name': month_name,
                    'birth_year': birth_year
            })
    except Exception as e:
            return jsonify({'error': 'Invalid date format. Use YYYY-MM-DD'}), 400
```

Last week system





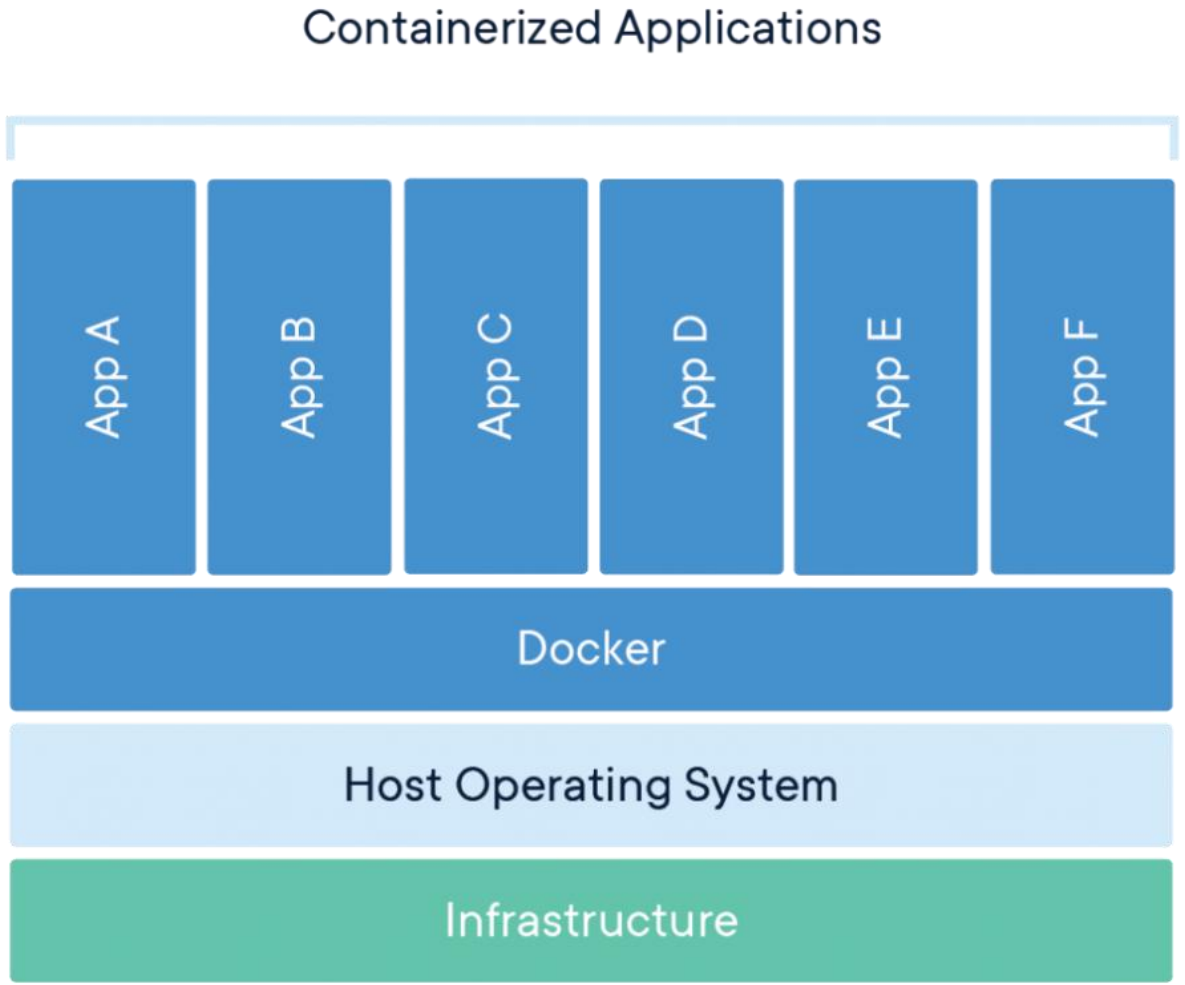
Try adding API component and show us on next week!

IoT Interns: Docker

4 June 2025

What is Docker?

Docker is a tool that lets you package an app with everything it needs — code, libraries, and settings — into a **container** so it runs the same anywhere

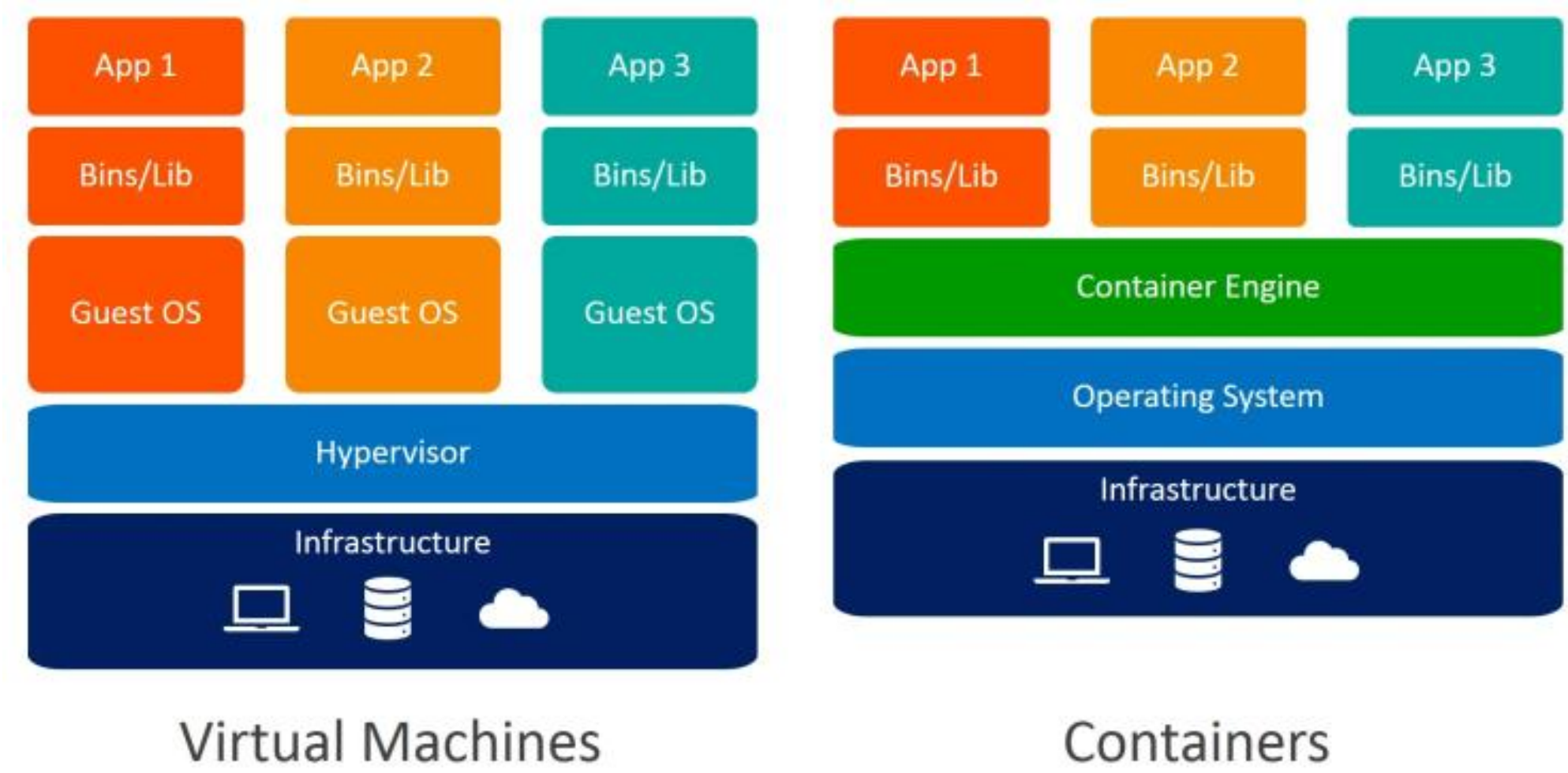


Lightweight
(no full OS required)

Portable
(Containerized as image)

http://docker.com/resources/what-container/

VMs vs Containers



https://www.linkedin.com/pulse/docker-vs-virtual-machines-ashish-yadav

Dockerfile

- To containerize an application, we use a *Dockerfile* that defines the instructions needed to build a *Docker image*.

```
# Use an official slimmed-down Python image as the base
FROM python:3.11-slim

# Set the working directory inside the container
WORKDIR /app

# Copy the requirements file into the container
COPY requirements.txt /app/

# Install Python dependencies without caching to reduce image size
RUN pip install --no-cache-dir -r requirements.txt

# Copy the rest of the project files into the container
COPY . /app/

# Expose the port Flask runs on (default is 5000)
EXPOSE 5000

# Default command to run the Flask app
CMD ["python", "app.py"]
```

Build Docker Image

- Go to directory of dockerfile > Build a docker image

docker build –t [image name] .

docker build –t [image name]:[tag name (default: latest)] .

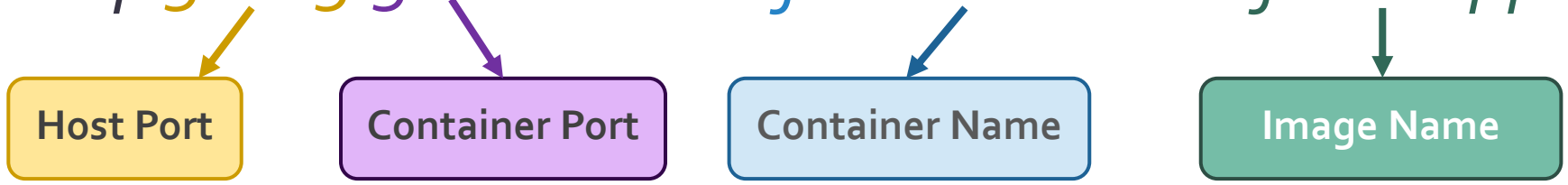
- List images on the machine

docker images

Run Docker Image

- Start a container

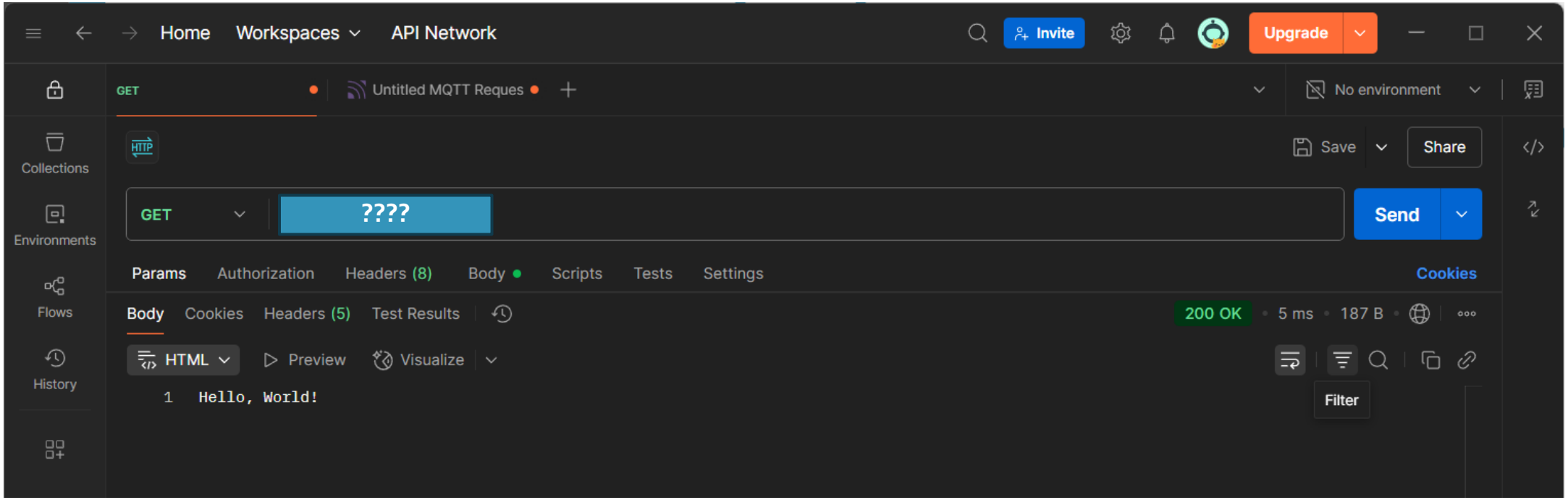
docker run -p 30003:5000 --name flask-container flask-app



docker run -p 30003:5000 -d flask-app:latest

Check the deployment with Postman

Question: To **which IP address and port** should you send the GET request????



Start / Stop Container

- *List running container*

docker ps *docker ps -a* (including stopped container)

- Stop the container

docker stop [container id]

- Start the container

docker start [container id]

- Restart the container

docker restart [container id]

Accessing the container

- Execute command inside container

docker exec -it [container_id or name] bash

docker exec -it [container_id or name] sh

- See logs inside container

docker logs [container_id or name]

Push your image to Docker Hub

- Login docker hub with docker desktop
- Change image name to be <Username>/<Image_name>:<tag>

docker tags [current image name] [new image name]

- Push to Docker Hub

docker push [new image name]

- Check your image on Docker Hub

Pull image from Docker Hub

- Remove images

docker rmi [image name]

- Remove stopped container

docker rm [container name]

- Pull Docker Image from Docker Hub

docker pull [image name on Docker Hub]:[tag]

Docker CLI cheat sheet

https://www.docker.com/resources/cli-cheat-sheet/

https://docs.docker.com

IMAGES

Docker images are a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings.

Build an Image from a Dockerfile
```
docker build -t <image_name>
```
Build an Image from a Dockerfile without the cache
```
docker build -t <image_name> . -no-cache
```
List local images
```
docker images
```
Delete an Image
```
docker rmi <image_name>
```
Remove all unused images
```
docker image prune
```

DOCKER HUB

Docker Hub is a service provided by Docker for finding and sharing container images with your team. Learn more and find images at https://hub.docker.com

Login into Docker
```
docker login -u <username>
```
Publish an image to Docker Hub
```
docker push <username>/<image_name>
```
Search Hub for an image
```
docker search <image_name>
```
Pull an image from a Docker Hub
```
docker pull <image_name>
```

CONTAINERS

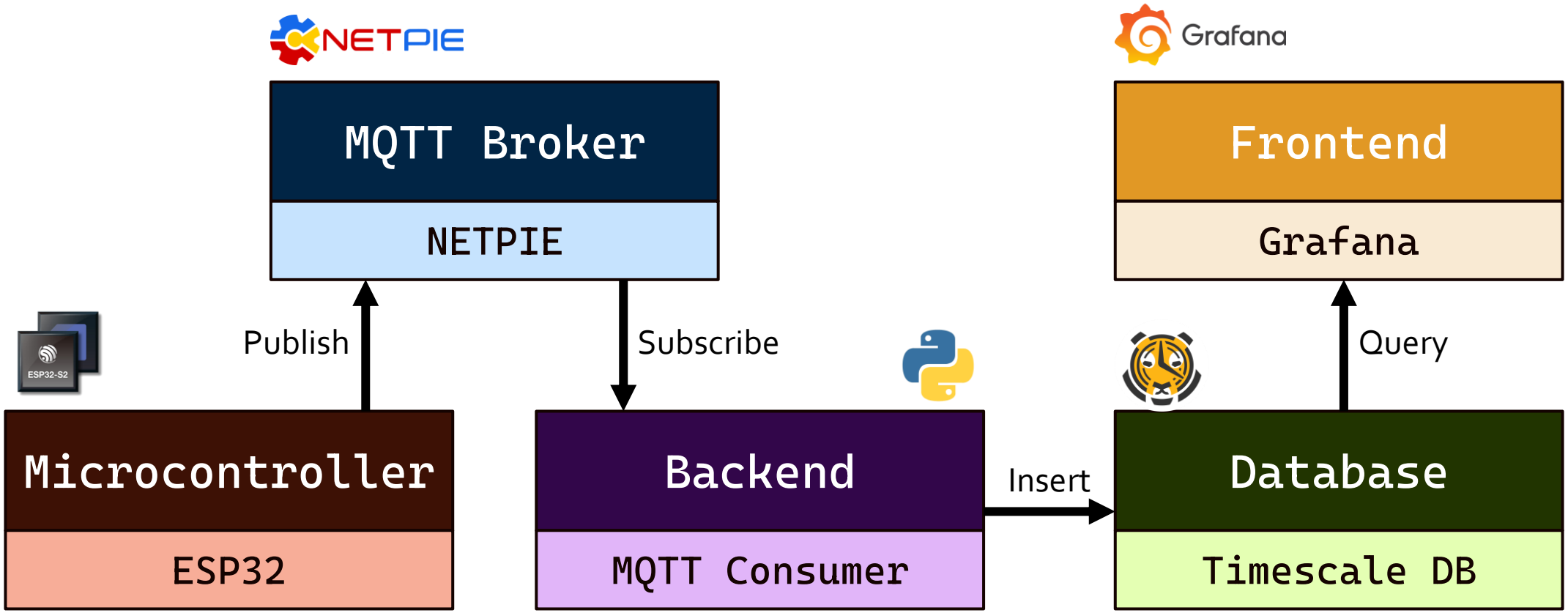
A container is a runtime instance of a docker image. A container will always run the same, regardless of the infrastructure. Containers isolate software from its environment and ensure that it works uniformly despite differences for instance between development and staging

Create and run a container from an image, with a custom name:
```
docker run --name <container_name> <image_name>
```
Run a container with and publish a container's port(s) to the host.
```
docker run -p <host_port>:<container_port>
<image_name>
```
Run a container in the background
```
docker run -d <image_name>
```
Start or stop an existing container:
```
docker start|stop <container_name> (or <container-
id>)
```
Remove a stopped container:
```
docker rm <container_name>
```
Open a shell inside a running container:
```
docker exec -it <container_name> sh
```
Fetch and follow the logs of a container:
```
docker logs -f <container_name>
```
To inspect a running container:
```
docker inspect <container_name> (or <container_id>)
```
To list currently running containers:
```
docker ps
```
List all docker containers (running and stopped):
```
docker ps --all
```
View resource usage stats
```
docker container stats
```

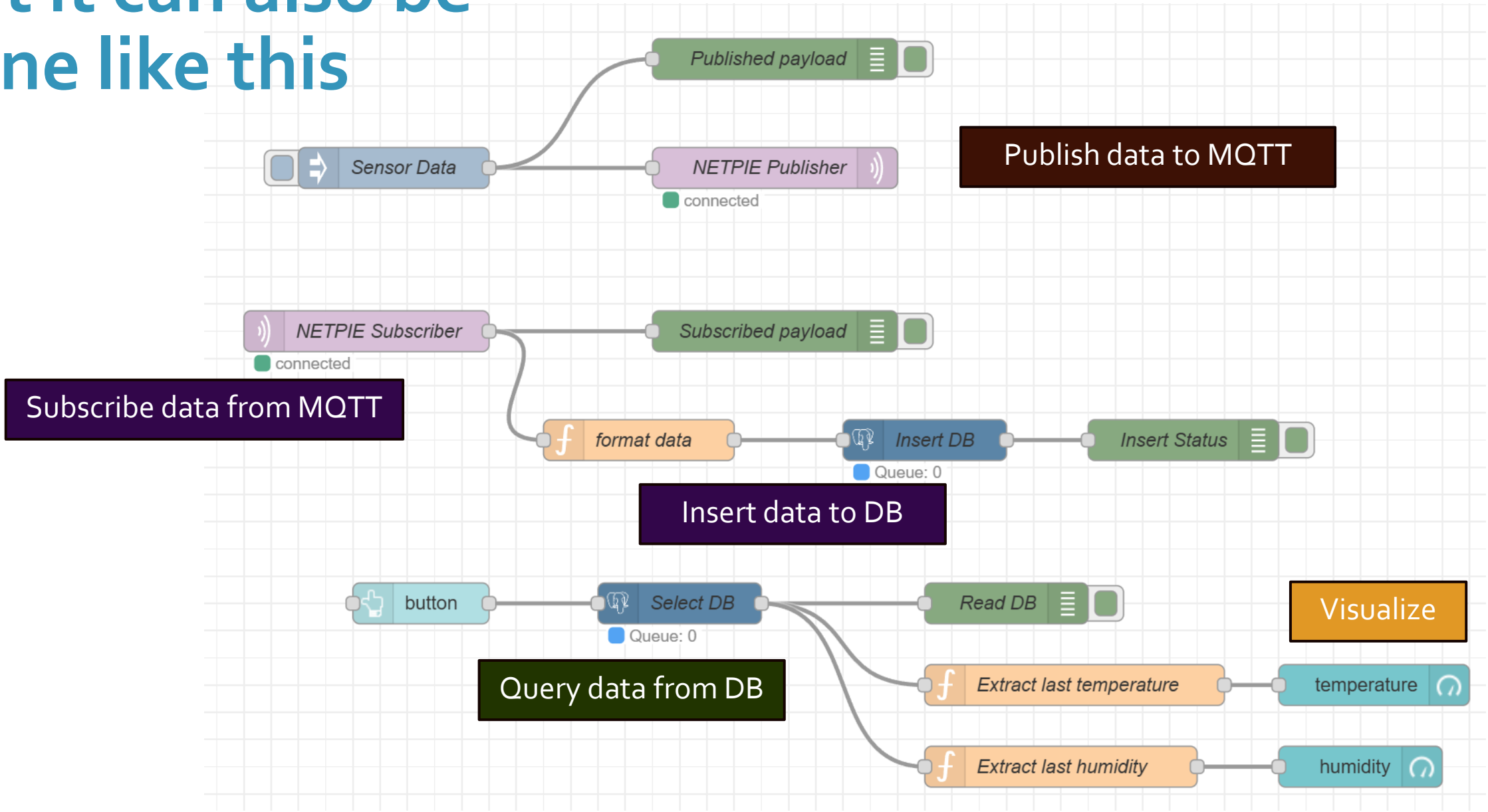
IoT Interns: Node-Red (Optional)

4 June 2025

We have already implemented this

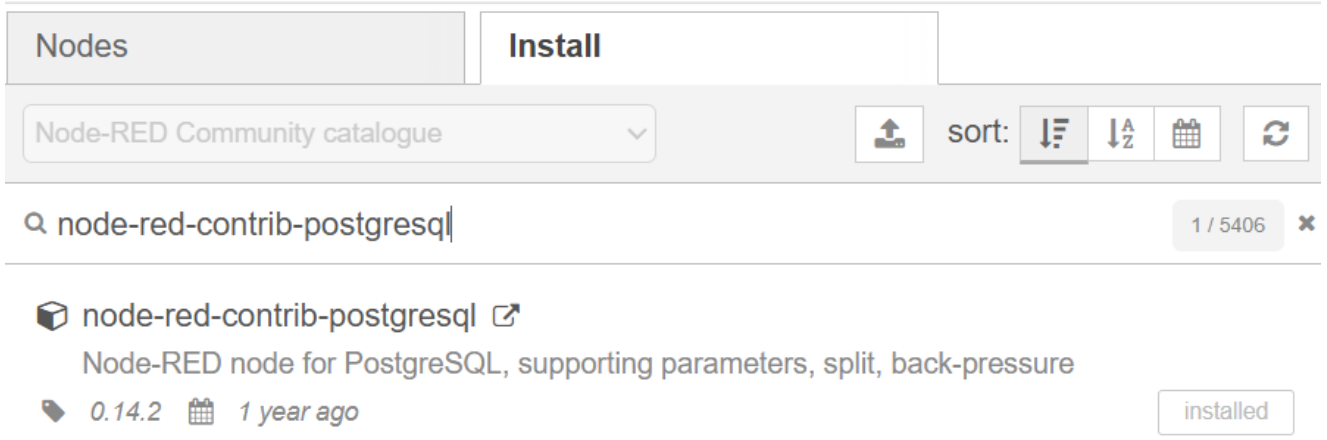


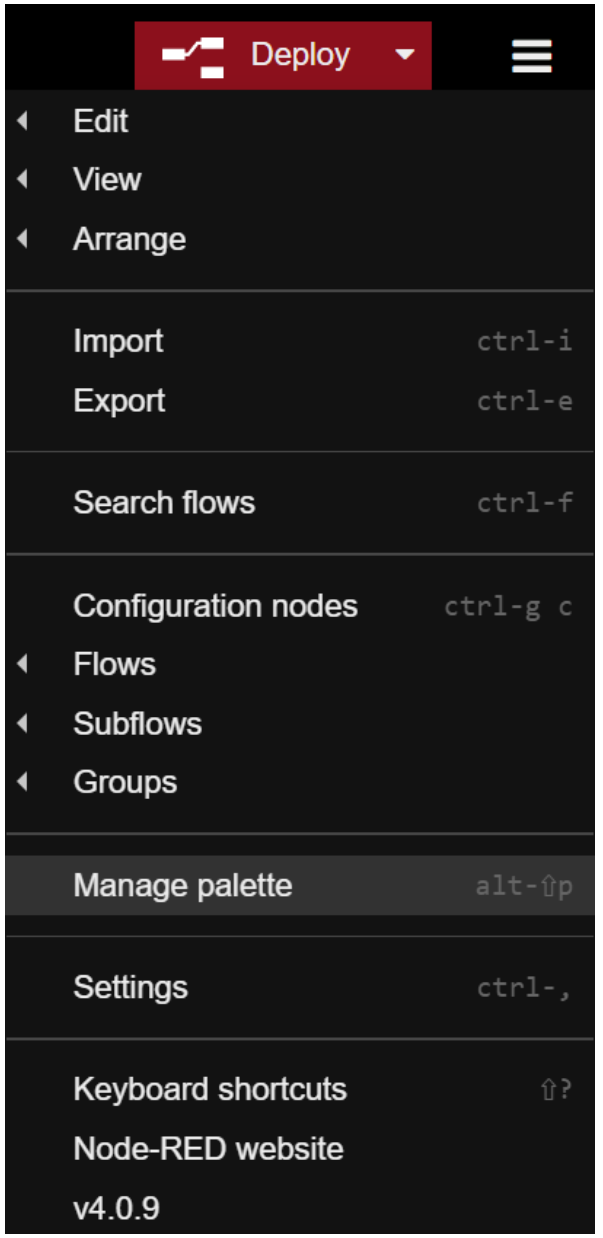
But it can also be done like this



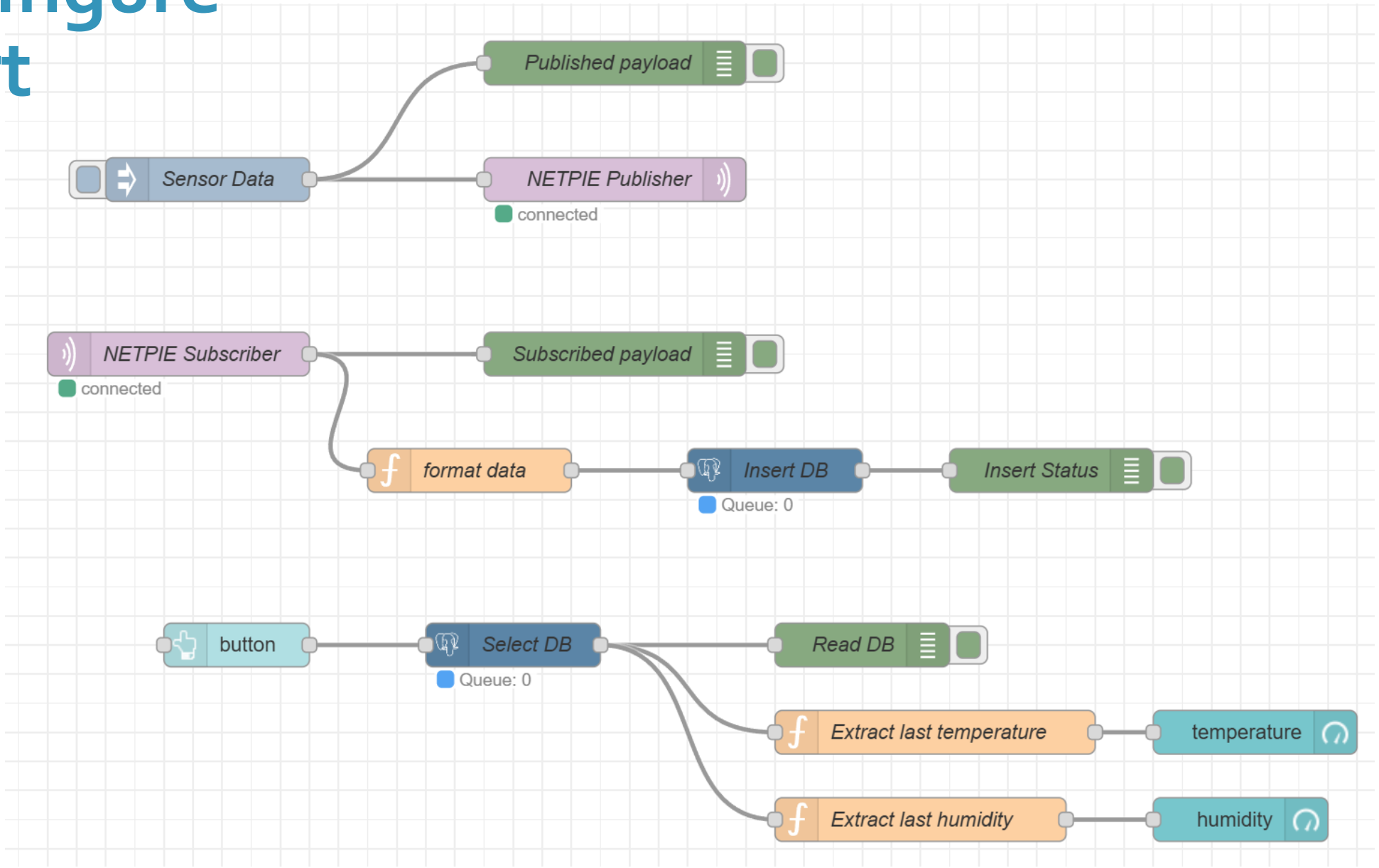
Requied Palette

- node-red-dashboard
- node-red-contrib-postgresql





Let's Configure Each Part

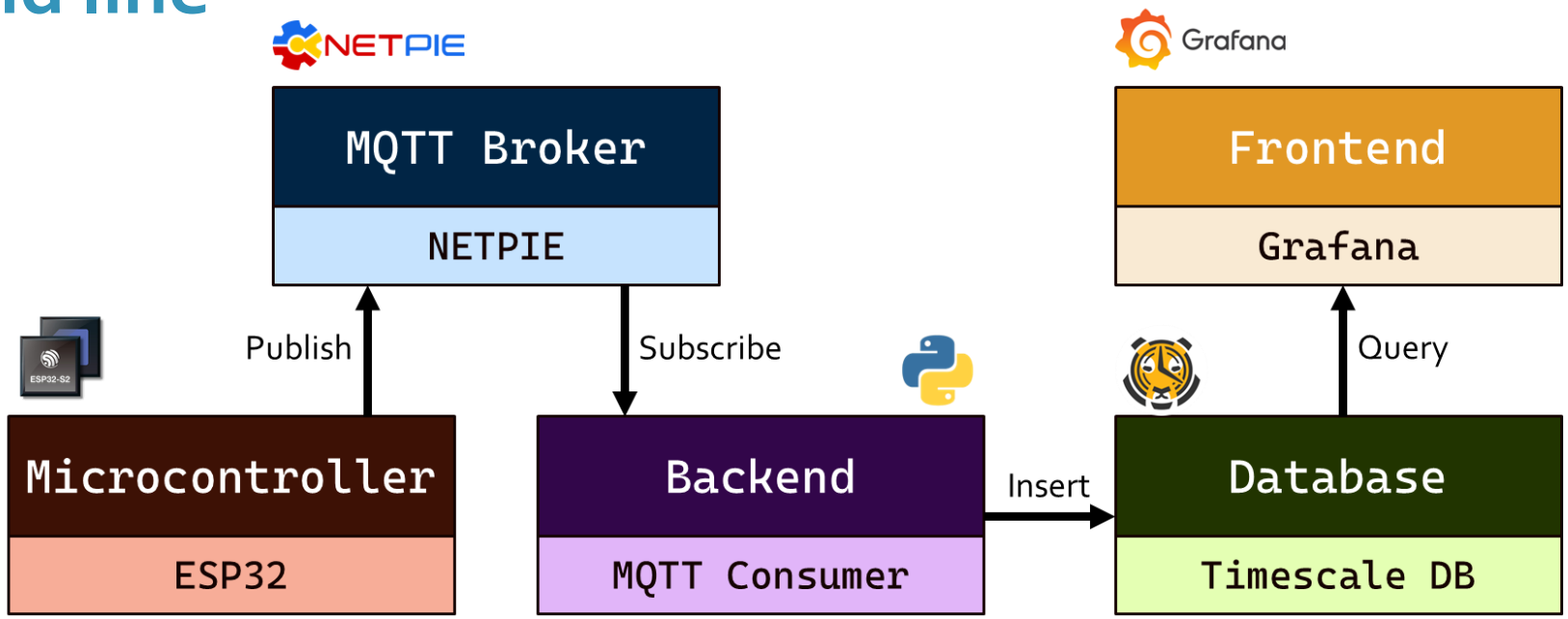


Summary

4 June 2025

Summary

- Linux Command line
- Git & GitHub
- Networking
- API Server
- Docker
- Node-Red

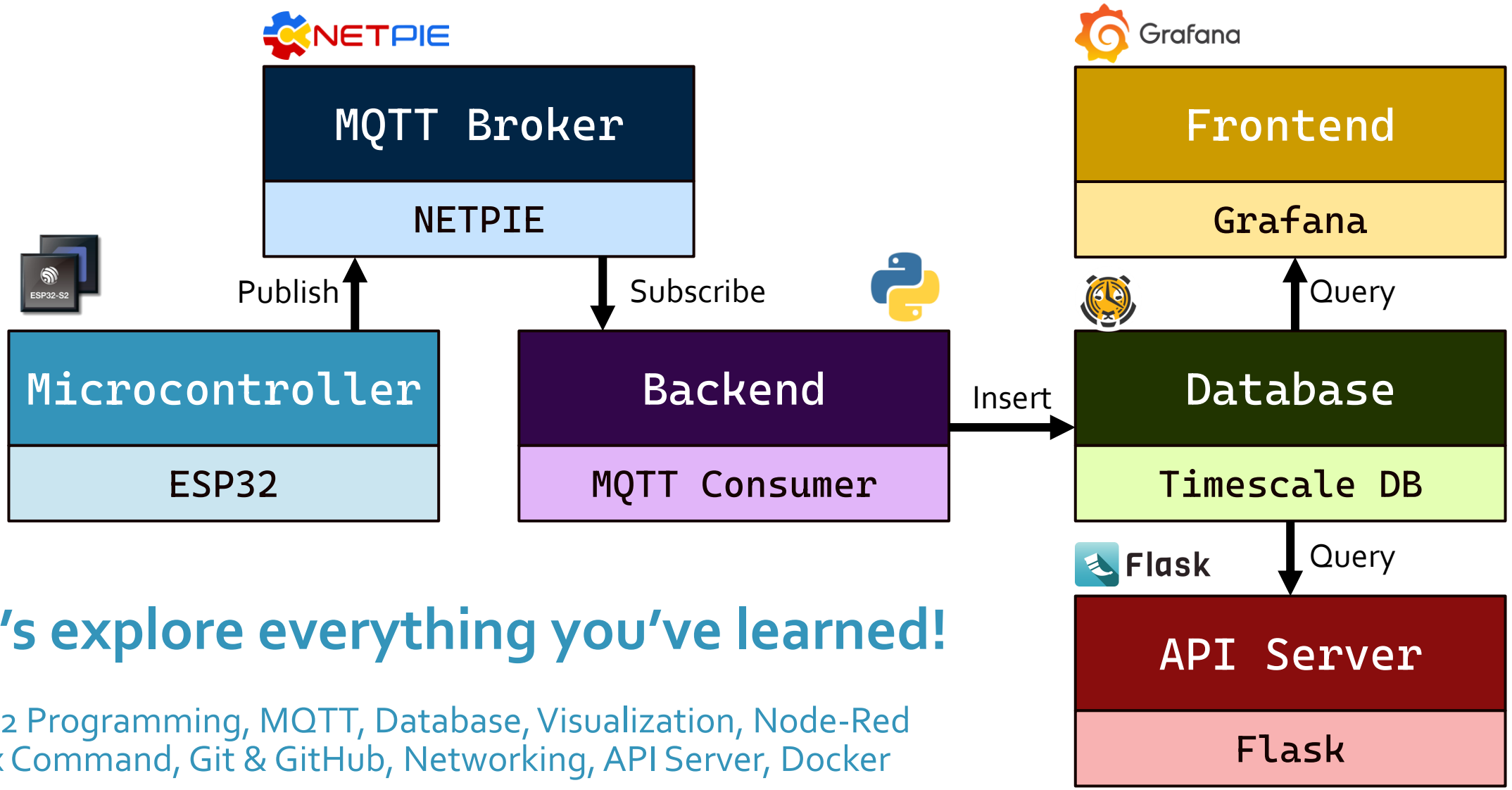


During this week!

1. Try Building Your Own API Server and Containerize It!
   - Design custom API endpoints to retrieve data from a database.
   - You can add endpoints to fetch historical data within a specified range (e.g., "last 7 days," "last 5 minutes," or "June 1 to June 3").
   - Try computing useful metrics such as mean, mode, or moving averages.
   - Implement any features you think would be useful for real-world applications.
2. Try running your database on one machine and deploy your API or related services on other machines (your friends' devices). Test if all services are accessible from every machine.
   Just make sure you're all connected to the same LAN.
3. Feel free to upload your group work to GitHub and share the repository in our Line group.

Feel free to try anything

This isn't an assignment. It's just a hands-on opportunity to learn and experiment!



Let's explore everything you've learned!

ESP32 Programming, MQTT, Database, Visualization, Node-Red
Linux Command, Git & GitHub, Networking, API Server, Docker

What'll we do next week?

1. Each group will present what they've worked on so far. It's totally okay if things aren't finished—just share what you've tried or explored.
2. We'll provide relevant materials for each group to help you get started on your project.
3. You'll also have a chance to share what you've read and discuss your plans for the group project.
4. We encourage open sharing and discussion—this way, you'll not only learn from your own project but also gain insight from your friends' work. Our goal is for everyone to grow and gain experience together!