



# Smart Cities Hackathon Tier 3 Challenge 10

### Tier 3 Challenge 10: Environmental Data Uploaded Over HTTP

This challenge will see you create a new sensor decoder in Core to support the environmental data and then develop a Python script which will accept an array of river bed IDs. It will also see you call the environmental agency API and extract the latest reading. The script will then check to see if the device exists and create it for the first time before uploading the readings for each river. You will learn about API's, Python, Docker and third party data ingest

The third party API docs can be found at:

<https://environment.data.gov.uk/flood-monitoring/doc/tidegauge>

The Core API docs can be found at:

<https://api.core.aql.com/doc/>

#### Step 1

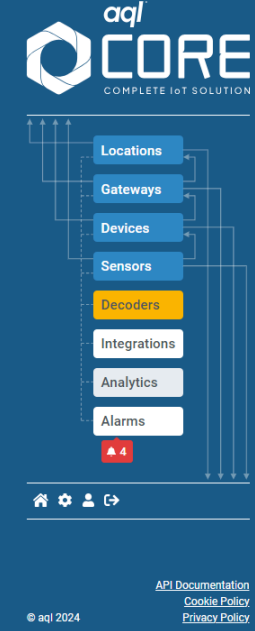
Install Docker as per **Tier 3 Challenge 9**, or run the scripts directly from your local machine.

#### Step 2

Log into the aql Core IoT platform (<https://core.aql.com/login>) with your provided event credentials.

#### Step 3

Create an **Other** decoder. Once logged into your user account, select “Decoders” from the menu on the left of the page, and then select the “Add Decoder” button from the decoders index page. You should then be loaded onto a new page with a form to fill in for creating a new decoder:



Locations  
Gateways  
Devices  
Sensors  
Decoders  
Integrations  
Analytics  
Alarms

API Documentation  
Cookie Policy  
Privacy Policy

© aqi 2024

## Add Decoder

Name: ①

Comments: ①

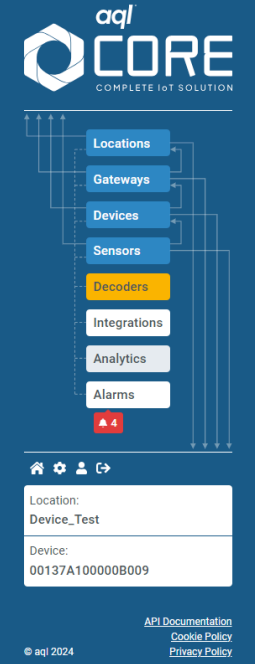
Device type: ①

Other

Cancel Submit →

## Step 4

Enter a name for this decoder “Team[YOURTEAMNUMBER]\_riverbed”. You will need to enter a comment. Make sure you select “Other” from the “Device type” dropdown menu before hitting submit.



Locations  
Gateways  
Devices  
Sensors  
Decoders  
Integrations  
Analytics  
Alarms

API Documentation  
Cookie Policy  
Privacy Policy

© aqi 2024

## Add Decoder

Name: ①

Team1\_\_riverbed

Comments: ①

\_riverbed

Device type: ①

Other

Cancel Submit →

Location:  
Device\_Test

Device:  
00137A100000B009

### Step 5

We next need to build a decoder that can parse the readings. The Python script we will build will prepare a simple JSON object that the decoder can return and be parsed to the sensor keys required. First, enter the JSON object below into the test Payload field and select Test Decoder. A JSON object will be returned.

Sample Payload:

```
{"level":2.94,"timestamp":1713424500000}
```

We don't need the timestamp as this will be prepared by the API. If we didn't provide a timestamp, the Core platform will use the data insertion date as the timestamp. We need to update the payload decoder to remove this element. Update the payload function as below

```
function parsePayload(payload, params) {  
  delete payload.timestamp;  
  return payload;  
}
```

### Step 6

Scroll down the decoder page to see that the Decoder has mapped the JSON output. Add the relevant sensors ("Liquid Level Depth" & "Time") that will be used with their sensor IDs and by changing the type accordingly, then select save.

### Step 7

We can now move to the Python elements of the task. These steps are based on the Docker approach and use the Docker image provided in the IoT Event Github, but the script can be run locally on a machine that has Python and the packages installed.

First, create a new directory for this project.

Inside this, create a requirements.txt with the following packages:

```
matplotlib  
requests
```

Then create a Dockerfile as follows:

```
# Use an official Python runtime as a parent image
FROM python:3.10-slim

# Set the working directory in the container
WORKDIR /app

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

# Install any needed dependencies specified in requirements.txt
RUN pip install --no-cache-dir -r requirements.txt

# Copy all Python files from the host's current folder into the container's
/app folder
COPY . /app/

CMD [ "python", "challenge10.py" ]
```

And a compose.yml:

```
services:
  python-app:
    build:
      context: .
      dockerfile: Dockerfile
    volumes:
      - ./app
```

### Step 8

Create a Challenge10.py file, and configure the imports.

```
import requests
from datetime import datetime as dt
```

### Step 9

We will next configure the constants that the script will use and replace the variables with the relevant variables covered in this and earlier tasks.

```
# Constants
#https://riverlevels.uk/
GOV_URL = 'http://environment.data.gov.uk/flood-monitoring'
CORE_URL = 'https://api.core.aql.com/v1/'
TOKEN = 'REPLACE WITH YOUR BEARER TOKEN'
LOCATION_ID = 'REPLACE WITH YOUR LOCATION ID'
DECODER_ID = 'REPLACE WITH YOUR DECODER ID'
TEAM = 'REPLACE WITH YOUR TEAM NUMBER'
```

```
STATION_IDS = [  
    "F1902",  
    "L1515",  
    "F1903",  
    "L1907",  
    "L2009",  
    "L1931",  
    "L2208",  
    "F2206",  
    "L2205",  
    "L2402",  
    "L2806",  
    "L2803",  
    "L2411",  
    "L2403"  
]
```

### Step 10

Let's add the loop to process each station ID and print these to the console.

```
# Get data from each station  
for station_id in STATION_IDS:  
    print("Getting data from", station_id)  
  
print("Challenge Complete")
```

To run the command to test the output you can run the following command or manually in Visual Studio code.

```
sudo docker compose up -build
```

The station IDs will be echoed out.

### Step 11

Next add a new definition to process the station, adding the function after the station IDs and replacing the data loop. This will call the GOV\_URL and prepare the data into a JSON object which is compatible with `api.core.aql.com`.

```
def process_station(station_id):  
    url = f"{GOV_URL}/id/stations/{station_id}/measures"  
    response = requests.get(url)  
    response.raise_for_status()  
    json_resp = response.json()  
    items = json_resp.get("items", [])
```

```
for item in items:
    notation = item.get("notation")
    latest_reading = item.get("latestReading", {})
    if not (notation and latest_reading.get("value") and
latest_reading.get("dateTime")):
        continue
    utc_dt = dt.strptime(latest_reading["dateTime"], "%Y-%m-%dT%H:%M:%SZ")
    data = {
        "reading": {
            "level": latest_reading["value"],
            "timestamp": int(utc_dt.timestamp() * 1000)
        }
    }
    print(data)
    print(notation, end=": ")
    print(latest_reading["value"], end="")
    print(item.get("unitName", ""))
    print()

# Get data from each station
for station_id in STATION_IDS:
    print("Getting data from", station_id)
    process_station(station_id)

print("Challenge Complete")
```

Running the code will echo the Station IDs and the JSON object that will be submitted.

```
sudo docker compose up -build
```

Well done! You have now completed this challenge.