INTERNET OF THINGS

ΧΡΗΣΗ RASPBERRY PI ΩΣ ΕΙΚΟΝΙΚΟ ΠΕΡΙΒΑΛΛΟΝΤΙΚΟ ΣΤΑΘΜΟ ΚΑΙ ΑΠΟΣΤΟΛΗ ΔΕΔΟΜΕΝΩΝ ΓΙΑ ΕΠΕΞΕΡΓΑΣΙΑ ΣΕ ΑΠΟΜΑΚΡΥΣΜΕΝΟ SERVER

ΠΕΡΙΓΡΑΦΗ ΣΥΣΤΗΜΑΤΟΣ

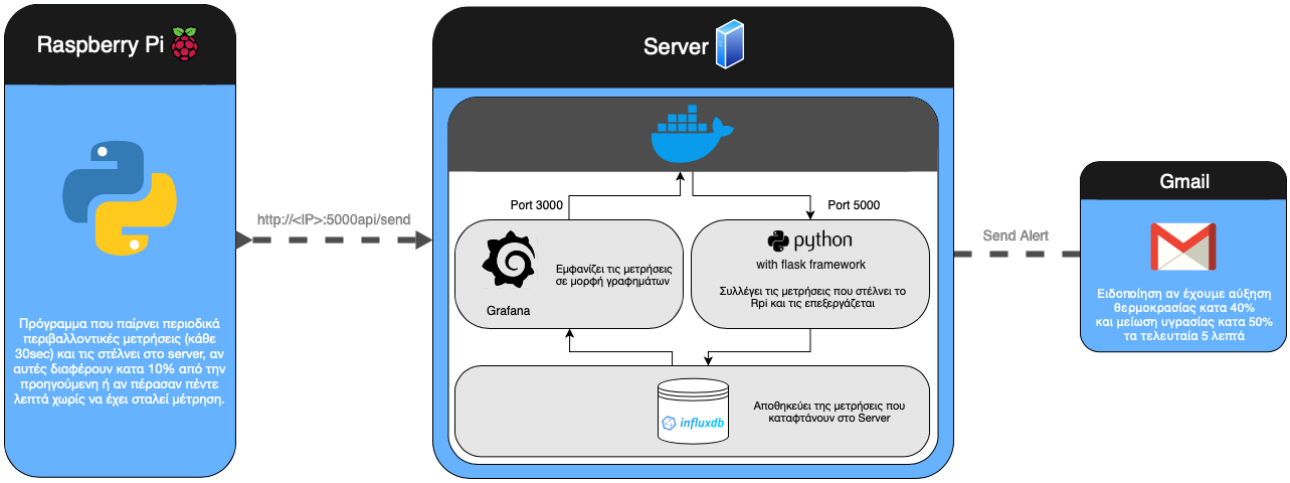
731 45517
ΝΙΚΟΣ ΚΟΥΤΣΟΛΕΛΟΣ

Εισαγωγη

Στόχος

Ο στόχος της άσκησης είναι να δημιουργήσουμε ένα εικονικό περιβαλλοντικό σταθμό που παράγει περιοδικά μετρήσεις θερμοκρασίας, υγρασίας αέρος, υγρασίας εδάφους και έντασης αέρος. Στη συνέχεια ελέγχει τις τιμές και αν αυτές διαφέρουν πάνω από 10% αποστέλλονται σε μηχανισμό συλλογής δεδομένων. Αν περάσει χρονικό διάστημα πέντε λεπτών από την τελευταία αποστολή δεδομένων, αποστέλλονται οι τρέχουσες τιμές. Ο Server επεξεργάζεται τα δεδομένα που καταφθάνουν και αν διαπιστωθεί ότι η θερμοκρασία περιβάλλοντος αυξήθηκε κατα 40% και η υγρασία αέρα μειώθηκε κατα 50% τα τελευταία πέντε λεπτά, αποστέλλει ειδοποίηση με email στο χρήστη. Τέλος αποθηκεύει τα δεδομένα σε βάση δεδομένων και παράγει γραφήματα με τις μετρήσεις.

Σχεδιάγραμμα

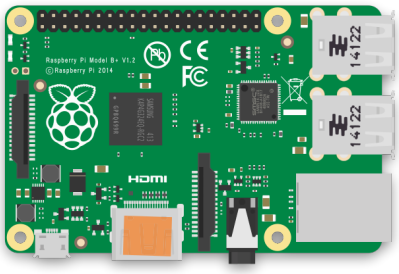


Το σύστημα αποτελείτε από δύο συσκευές και προγράμματα, τον IoT μετεωρολογικό σταθμό (στα αριστερά) σε πλατφόρμα Raspberry Pi και το Server, ο οποίος τρέχει μέσα στο Docker. Ο μετεωρολογικός σταθμός καθώς και ο Server τρέχουν προγράμματα γραμμένα σε γλώσσα Python.

Προεργασία

Για το Raspberry Pi

Το Raspberry pi είναι μια σειρά μικρών, χαμηλού κόστους υπολογιστών μιας πλακέτας (single-board computer) στο μέγεθος πιστωτικής κάρτας, που δημιουργήθηκε στο Ηνωμένο Βασίλειο το Φεβρουάριο του 2012 από το Raspberry pi Foundation, για την ευκολότερη εκμάθηση της επιστήμης των υπολογιστών στα σχολεία.



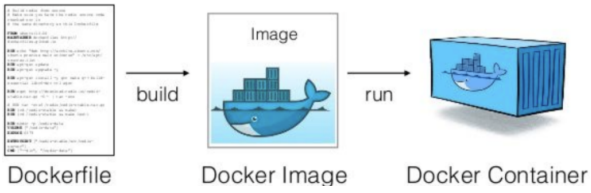
Θα πρέπει να κατεβάσουμε το πιο πρόσφατο λειτουργικό σύστημα για το pi από την ιστοσελίδα του κατασκευαστή, και να το εγκαταστήσουμε στην SD κάρτα. Στη συνέχεια μπορούμε αν θέλουμε να προσθέσουμε ένα κενό αρχείο με όνομα “ssh” στο *Boot* κατάλογο για να ενεργοποιήσουμε την εν λόγο λειτουργία και να μπορούμε να το χρησιμοποιήσουμε headless. Τέλος αρκεί να αναβαθμίσουμε τα προγράμματα και τα πακέτα μας τρέχοντας την εντολή:

```
sudo apt update && apt upgrade
```

Για το Server

Για τη δημιουργία του Server χρησιμοποίησα γλώσσα Python, το web framework Flask για δημιουργία του webapp και την διαχείρηση τις εισερχόμενης πληροφορίας και Docker προκειμένου να είναι πιο εύκολη η εγκατάσταση. Το Flask είναι ένα framework με το οποίο μπορούμε να δημιουργήσουμε ιστοσελίδες και διαδικτυακά προγράμματα καθώς και να διαχειριστούμε Rest api. Ενώ με το Docker μπορούμε να φτιάξουμε και να τρέξουμε Containers, πρόκειται για OS-level virtualisation.

Φτιάχνοντας ένα dockerfile στο οποίο ορίζουμε το λειτουργικό, τα προγράμματα που θέλουμε να εγκαταστήσουμε, καθώς και μεταβλητές ή εντολές που θέλουμε να εκτελέσουμε δημιουργείτε ένα Docker Image το οποίο μετά μπορούμε να τρέξουμε.



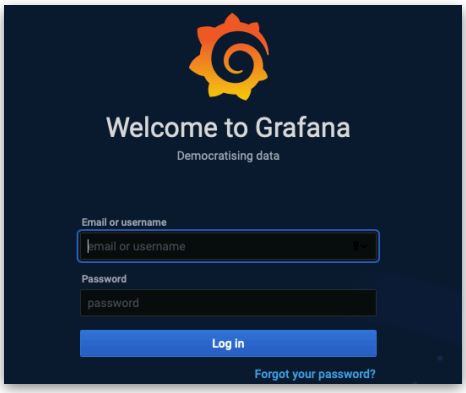
Εγκατάσταση και εκτέλεση προγραμμάτων

Για το Server

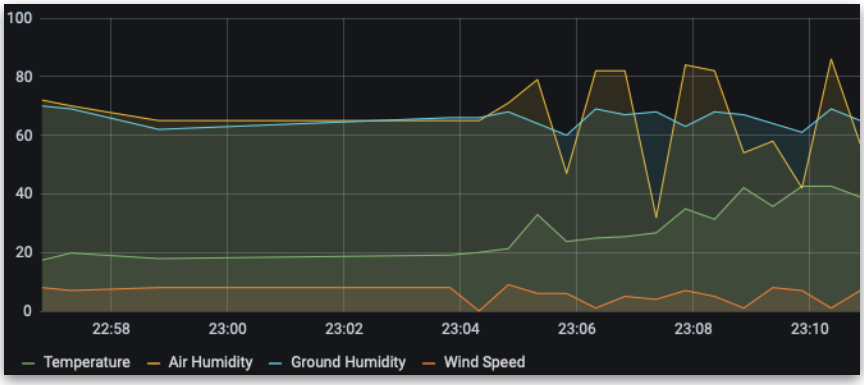
Για να εγκαταστήσουμε το πρόγραμμα θα πρέπει να έχουμε εγκατεστημένο το Docker. Στη συνέχεια αρκεί να κατεβάσουμε το αρχείο server τοπικά, να μεταφερθούμε στον υποκατάλογο και να τρέξουμε το αρχείο docker-compose.yml με την εντολή:

```
docker-compose -f "server/docker-compose.yml" up -d --build
```

Η εντολή αυτή θα δημιουργήσει και θα τρέξει το image. Αν μεταφερθούμε στη διεύθυνση του μηχανήματος στο οποίο κάναμε την εγκατάσταση (πχ http://localhost:5000 αν το τρέξαμε τοπικά) θα πρέπει να δούμε το μήνυμα “Server is up and ready to receive”, ενώ αν μεταφερθούμε στην σελίδα localhost:3000 και συνδεθούμε στο WebApp του Grafana με όνομα χρήστη και κωδικό admin.



Παρατηρούμε ότι έχει δημιουργηθεί αυτόματα ένα DashBoard με όνομα Pi. Επιλέγοντας το, εμφανίζονται τα δεδομένα μας (αν υπάρχουν).



Για το περιβαλλοντικό σταθμό

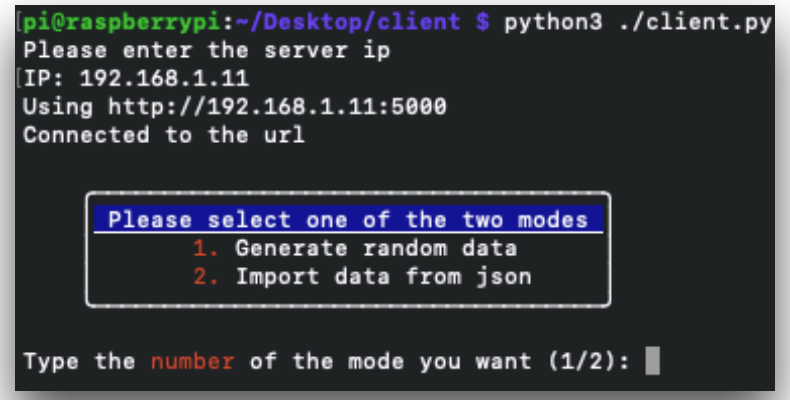
Για να εγκαταστήσουμε το πρόγραμμα αρκεί να έχουμε εγκατεστημένη την Python3 (έρχεται προεγκατεστημένη με το Pi), να αντιγράψουμε το φάκελο client τοπικά και να τρέξουμε το client.py. Αν δεν έχουμε εγκατεστημένη την βιβλιοθήκη requests θα πρέπει να τρέξουμε την εντολή

```
pip install -r requirements.txt
```

Για να εκτελέσουμε το πρόγραμμα τρέχουμε το client.py από το κατάλογο client χρησιμοποιώντας:

```
pi@raspberrypi:~/Desktop/client $ python3 client.py
```

Κατα την εκτέλεση θα μας ζητηθεί η διεύθυνση του Server (το port παίρνει τιμή αυτόματα 5000), ο οποίος θα πρέπει ήδη να τρέχει. Στη συνέχεια θα μας ζητηθεί να επιλέξουμε τρόπο λειτουργίας του προγράμματος.



Επιλέγοντας την πρώτη επιλογή το πρόγραμμα παράγει τυχαία δεδομένα, ενώ με την δεύτερη επιλογή γίνεται εισαγωγή των δεδομένων από αρχείο (.json). Αυτή η επιλογή κάνει ευκολότερη την εύρεση λαθών στο κώδικα, καθώς τρέχει γνωστά δεδομένα και ελέγχει όλες τις, για την άσκηση απαιτούμενες λειτουργίες (αποστολή δεδομένων, ειδοποιήσεων, κα). Κατα την εκτέλεση το πρόγραμμα τυπώνει στην οθόνη όλες τις μετρήσεις που πήρε, και αν αυτές αποστάλθηκαν στο Server.

```
Current sensor value is: {'time': '00:49:20', 'temperature': 17.4, 'ground-humidity': 70, 'air-humidity': 72, 'windspeed': 8}
Send: {'time': '00:49:20', 'temperature': 17.4, 'air-humidity': 72, 'ground-humidity': 70, 'windspeed': 8}

Current sensor value is: {'time': '00:49:50', 'temperature': 19.8, 'ground-humidity': 69, 'air-humidity': 70, 'windspeed': 7}
temperature
compaired {'time': '00:49:50', 'temperature': 19.8, 'ground-humidity': 69, 'air-humidity': 70, 'windspeed': 7} with {'time':
Send: {'time': '00:49:50', 'temperature': 19.8, 'air-humidity': 70, 'ground-humidity': 69, 'windspeed': 7}
```

1.

Υλοποιηση λογισμικου συσκευης IOT για συλλογη περιβαλλοντικων δεδομενων

Επεξήγηση κώδικα

Με τη χρήση του selection_menus.select_operating_mode() και selection_menus.server_ip_address() εμφανίζονται τα μενού που ζητάνε από το χρήστη το Port και τη λειτουργία που θα ξεκινήσει.

```
def server_ip_address():
    '''Server ip user input and menu draw'''

    print("Please enter the server ip")
    while True:
        url = "http://" + input("IP: ") + ":" + "5000"

        print("Using "+url)
        timeout = 5
        try:
            requests.get(url, timeout=timeout)
            print("Connected to the url")
            return url+"/api/send"
        except:
            print("No internet connection.")
```

```
def select_operating_mode():
    '''Mode selection and menu draw'''

    draw_menu = u"""
    ┌──────────────────────────────────────┐
    │\u001b[4m\u001b[44m Please select one of the two modes \u001b[0m│
    │       \u001b[31m1.\u001b[0m Generate random data       │
    │       \u001b[31m2.\u001b[0m Import data from json      │
    └──────────────────────────────────────┘ """
    draw_question = u"\nType the \u001b[31mnumber\u001b[0m of the mode you want (1/2): "

    print(draw_menu)
    while True:
        ans = input(draw_question)
        if ans == "1":
            return 0
        elif ans == "2":
            return 1
        print("ERROR")
```

Το πρόγραμμα δημιουργεί τυχαίες μέτρησης με τη χρήση του παρακάτω κώδικα. Εδώ, αν το επιθυμούμε μπορούμε να αλλάξουμε το εύρος των τιμών. Ο κώδικας για την θερμοκρασία είναι λίγο διαφορετικός για να παράγει θερμοκρασίες με ακρίβεια ενός δεκαδικού.

```
def weather_station():
    '''Create random weather data'''

    temp_min = 30       #Temperature range in Celsius
    temp_max = 45

    airhum_min = 40     #Air humidity range in percentage
    airhum_max = 90

    grdhum_min = 60     #Ground humidity range in percentage
    grdhum_max = 70

    wind_min = 0        #Wind speed in Km/h
    wind_max = 10

    measurment = {
        "temperature": random.randrange(temp_min * 10, temp_max * 10)/10,
        "air-humidity": random.randrange(airhum_min, airhum_max),
        "ground-humidity": random.randrange(grdhum_min, grdhum_max),
        "windspeed": random.randrange(wind_min, wind_max),
        "time": datetime.now().strftime("%H:%M:%S"),    #Log current time
    }
    return measurment
```

Η weather_station() επιστρέφει ένα αντικείμενο dict(). Το οποίο συγκρίνει, με την χρήση των βοηθητικών συναρτήσεων percentage() (σχ. 1.0) και five_minutes_passed() (σχ. 1.1) με την τελευταία τιμή που εστάλη.

```python
def percentage(new, old):
    '''Check if numbers are within 10% of eachother (old value ±10%)'''
    return bool(not (old - old * 10 / 100.0) <= new <= (old + old * 10 / 100.0))
```

σχ. 1.0

```python
def five_minute_passed(time_last_send):
    '''Checks if 5min have passed'''

    time_last_send = time_last_send['time']
    time_now = datetime.now().strftime("%H:%M:%S")
    tdelta = datetime.strptime(time_now, '%H:%M:%S') - datetime.strptime(time_last_send, '%H:%M:%S')

    if tdelta.seconds >= 300:
        print("SEND CAUSE TIME PASSED")
        return True
```

σχ 1.1

Αν οι μετρήσεις διαφέρουν κατα δέκα τα εκατό (τελευταία τιμή ±10%) ή αν έχουν περάσει πέντε λεπτά από την τελευταία φορά που εστάλη τιμή αποστέλλεται και αυτή στο Server σε μορφή JSON.

```python
if more_than_10_percent(last_send, curr_measurement) or five_minute_passed(last_send):
            last_send = curr_measurement.copy()
            print("Send: " + str(last_send) + "\n")
            try:
                put(url=url, json=json.dumps(last_send))
            except ConnectionError:
                print("ERROR: Lost connection with server")
                return 0
```

Δεν χρησιμοποιείται καμία μέθοδος εξακρίβωσεις στοιχείων, αν σταλούν δεδομένα στο /api/send, και είναι στη μορφή που περιμένει ο Server θα τα δεχτεί. Αν τρέχουμε το σύστημα στο τοπικό μας δίκτυο (όπως κάνουμε για την άσκηση) δεν υπάρχει πρόβλημα. Ωστόσο αν θέλαμε να λαμβάνουμε μετρήσεις από μετεωρολογικούς σταθμούς μέσω του διαδύκτιου, θα ήταν σκόπιμο να κάνουμε χρήση μιας τέτοιας μεθόδου, όπως για παράδειγμα API Tokens.

2.
Αποστολη δεδομενων σε απομακρυσμενο server

Επεξήγηση κώδικα

Το πρόγραμμα μας περιμένει να λάβει δεδομένα στην παρακάτω διεύθυνση σε μορφή JSON:

```
http://localhost:5000/api/send/
```

Τα δεδομένα που αποστέλλονται σώζονται τοπικά σε μια λίστα, η οποία διατηρεί τις πρόσφατες μετρήσεις έτσι ώστε να μπορεί να γίνει η σύγκριση του επόμενου ερωτήματος.

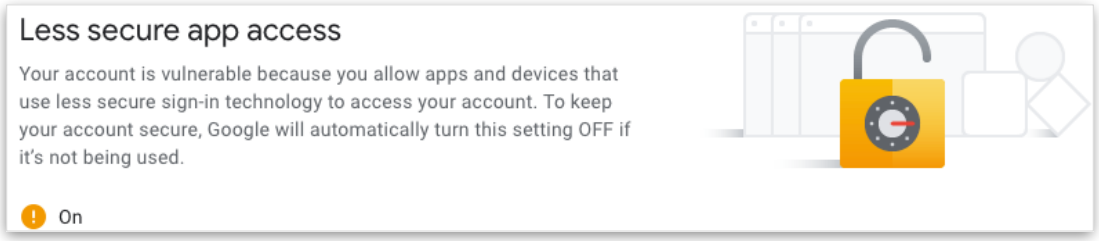
```python
class ReceiveMeasurments(Resource):

    def put(self):
        '''Receives data send to api'''

        measurments.append(json.loads(request.get_json()))
        data_process.real_time_processing(measurments)
        return 200

api.add_resource(ReceiveMeasurments, '/api/send')
```

3.
Υλοποιηση υπηρεσιας επιτηρησης πραγματικου χρονου

Τα στοιχεία που καταφτάνουν στο Server πρέπει να τα επεξεργαστούμε για να διαπιστώσουμε αν η θερμοκρασία περιβάλλοντος έχει αυξηθεί κατα 40% και η υγρασία αέρα μειωθεί κατα 50% τα τελευταία πέντε λεπτά στην περίπτωση που αυτό ισχύει θα πρέπει να αποστείλουμε ειδοποίηση στον χρήστη.

Προεργασία για το email

Για την αποστολή ειδοποιήσεων θα κάνουμε χρήση λογαριασμού Gmail (για χάρη τις άσκησης δημιούργησα καινούργιο) στον οποίο θα πρέπει να πλοηγηθούμε στην σελίδα με τις ρυθμίσεις για την ασφάλεια και να ενεργοποιήσουμε την λειτουργία Less secure app access.



Στη συνέχεια αρκεί να αντιγράψουμε το όνομα χρήστη και το κωδικό μας στο φάκελο .env (server/.env).

Επεξήγηση κώδικα

Όταν μια μέτρηση καταφτάνει στο server αποθηκεύεται σε μια λίστα (MEASUREMENTS) η οποία κρατάει πάντα τις τιμές των τελευταίων πέντε λεπτών, επίσης αποστέλλεται στη βάση δεδομένων.

```
def put(self):
        new_data = json.loads(request.get_json())
        MEASUREMENTS.append(new_data.copy())
        database_handle.send_data(new_data.copy())
        data_process.real_time_processing(MEASUREMENTS)
        return 200
```

Η τιμή που έφτασε (latest_measurment) συγκρίνεται με τις υπόλοιπες τιμές τις λίστας (βλέπε παρατήρηση). Αν διαπιστωθεί ότι μια τιμή τις λίστας είναι παλαιότερη από πέντε λεπτά, τότε αυτή αφαιρείται:

```
else:
  data.remove(measurement)
```

Ενώ αν είναι εντός των 5 λεπτών, γίνεται σύγκριση με την βοήθεια τις βοηθητικής συνάρτησης percentage():

```
def percentage(old, new, operator, percent):
    '''Check if numbers are within range'''

    if operator == '+':
        return bool(round((old + old * percent / 100.0), 1) == new)
    elif operator == '-':
        return bool(round((old - old * percent / 100.0)) == new)
```

```
if tdelta.seconds < 300:
    if percentage(measurement['temperature'], latest_measurement['temperature'], '+', 40) and
        percentage(measurement['air-humidity'], latest_measurement['air-humidity'], '-', 50):

        send_mail(measurement['temperature'], latest_measurement['temperature'], measurement['air-humidity'],
                  latest_measurement['air-humidity'])
```

Αν διαπιστωθεί ότι η καινούργια μέτρηση έχει αυξημένη θερμοκρασία κατα 40% και μειωμένη υγρασία κατα 50% τότε με τη χρήση της συνάντησης send_mail(), στην οποία προωθούμε της μετρήσεις έτσι ώστε να αποσταλούν και αυτές στο χρήστη, στέλνουμε την ειδοποίηση.

Παρατηρήσεις

Από την έκδοση Python 3.0 η χρήση της round() έχει αλλάξει με αποτέλεσμα για *.5 επιστρέφει τον ποιο κοντινό ζυγό αριθμό (Gaussian rounding ή Banker rounding). Για παράδειγμα το round(2.5) να επιστρέφει αποτέλεσμα 2 και όχι 3.

Στο κώδικα χρησιμοποιώ αυτή τη λειτουργία όταν υπολογίζεται αν η υγρασία αέρα έχει μειωθεί κατα 50% ή αν η θερμοκρασία έχει αυξηθεί κατα 40%. Έκρινα πως δεν υπάρχει λόγος να αλλάξω τη λειτουργία, ωστόσο αν κρίνουμε ότι επηρεάζει τα αποτελέσματα μας μπορούμε είτε να φτιάξουμε δικιά μας συνάρτηση ή να κάνουμε χρήση της math.ceil(x).

4.
Υλοποιηση υπηρεσιας επιτηρησης πραγματικου χρονου

Τα δεδομένα που λαμβάνει ο Server αποθηκεύονται σε βάση δεδομένων InfluxDB και στη συνέχεια απεικονίζονται σε γραφικό περιβάλλον με τη μορφή γραφημάτων με τη χρήση του Grafana.

Επεξήγηση κώδικα

Για να εγκαταστήσουμε τη βάση δεδομένων InfluxDB προσθέτουμε το παρακάτω κώδικα στο αρχείο Docker-compose.yml

```
influxdb:
  environment:
    INFLUXDB_DB: ${DatabaseName}
    INFLUXDB_ADMIN_USER: ${AdminUser}
    INFLUXDB_ADMIN_PASSWORD: ${AdminPass}
    INFLUXDB_HTTP_AUTH_ENABLED: "true"
    INFLUXDB_USER: ${User}
    INFLUXDB_USER_PASSWORD: ${UserPass}
  container_name: "influxdb"
  image: influxdb:latest
  volumes:
    - influxdata:/var/lib/influxdb
```

Αυτό περνάει στο container τις απαιτούμενες μεταβλητές (Usernames, passwords και Database name) από το αρχείο .env, κατεβάζει τη ποιο πρόσφατη έκδοση της βάσης (container image) και ορίζει που θα αποθηκεύονται τα δεδομένα της.

Για το Grafana προσθέτουμε τα παρακάτω στο ίδιο αρχείο:

```
grafana:
container_name: "grafana"
image: grafana/grafana:latest
depends_on:
  - influxdb
environment:
  INFLUXDB_DATABASE: ${DatabaseName}
  INFLUXDB_USERNAME: ${User}
  INFLUXDB_PASSWORD: ${UserPass}
ports:
  - 3000:3000
volumes:
  - ./grafana-provisioning/:/etc/grafana/provisioning
  - grafanadata:/var/lib/grafana
```

Όπως και για τη Influx, περνάμε τους κωδικούς έτσι ώστε να μπορεί να συνδεθεί στη βάση, κατεβάζουμε την πιο πρόσφατη έκδοση του Grafana, δρομολογούμε την πόρτα 3000 του container στην 3000 του Host System, ορίζουμε που θα αποθηκεύονται τα δεδομένα και περνάμε τον υποκατάλογο grafana-provisioning.

```
.
├── dashboards
│   ├── dashboard.yml
│   └── grafana_panel.json
└── datasources
    └── datasrouce.yml
```

Αυτός ο φάκελος τροφοδοτεί το Grafana με τα στοιχεία που χρειάζεται για να κάνει τη σύνδεση με τη βάση και να δημιουργήσει αυτόματα τα γραφήματα, χωρίς να χρειάζεται να το κάνει ο χρήστης.

Τα τέσσερα query, με τα αντίστοιχα Allias ονόματα που τρέχει το Grafana για την απεικόνιση των γραφημάτων είναι:

```
SELECT "temperature" FROM "Rpi_data"
SELECT "air-humidity" FROM "Rpi_data"
SELECT "ground-humidity" FROM "Rpi_data"
SELECT "windspeed" FROM "Rpi_data"
```