Τμήμα Εφαρμοσμένης Πληροφορικής

Πρόγραμμα Μεταπτυχιακών Σπουδών

Μάθημα: Παράλληλος και Κατανεμημένος Υπολογισμός

Ονοματεπώνυμο: Μάνος Κακαράκης

AM: mai21022

Θέμα εργασίας:

"Microservices using Docker in Python"

Επιβλέπων καθηγητής:

Μαργαρίτης Κωνσταντίνος

ΠΕΡΙΕΧΟΜΕΝΑ

1. ΕΙΣΑΓΩΓΗ

Στόχος της εργασίας είναι η εκτέλεση και παρουσίαση της αρχιτεκτονικής των microservices, δηλαδή του επιμερισμού καθηκόντων και ευθυνών των διαφόρων λειτουργιών που θα έχει η εφαρμογή μας, με τη χρήση της τεχνολογίας Docker.

Για την διαδικασία αυτή, δηλαδή την όσο γίνεται αποκλειστική «δουλειά» του κάθε service (την χρησιμότητα της αρχιτεκτονικής θα παρουσιάσουμε παρακάτω) σημαντική και ριζική συμμετοχή καθώς και βοήθεια μας παρέχει η τεχνολογία Docker η οποία είναι βασισμένη στην έννοια των containers.

Η εφαρμογή στην οποία έχει εφαρμοσθεί η αρχιτεκτονική των microservices και των containers, είναι ένα μοντέλο επικοινωνίας πελάτη-διακομιστή με web services όπου ο πελάτης επικοινωνεί μόνο με τον διακομιστή και ζητάει τον καιρό μιας συγκεκριμένης πόλης ή τα κορυφαία πρωτοσέλιδα μιας συγκεκριμένης χώρας. Έπειτα ο διακομιστής αναλαμβάνει το ρόλο του ενορχηστρωτή και αποφασίζει σε ποιο service θα απευθυνθεί για να λάβει το αποτέλεσμα που θα επιστρέψει στον πελάτη. Η επικοινωνία επιτυγχάνεται με απλά http αιτήματα και η πληροφορία μεταφέρεται μέσω της μορφής json. Αναλυτικότερα θα παρουσιαστεί μετέπειτα η εφαρμογή και με κάποια παραδείγματα.

Στη συνέχεια θα γίνει μία παρουσίαση σημαντικών τεχνολογιών και εφαρμογών που αφορούν άμεσα την υλοποίηση της εφαρμογής αλλά και του χώρου του development γενικότερα καθώς και κάποια παραδείγματα κώδικα.

2. MICROSERVICES

Τα microservices είναι ένα στυλ αρχιτεκτονικής εφαρμογών όπου ανεξάρτητα, αυτόνομα προγράμματα με έναν μόνο σκοπό το καθένα μπορούν να επικοινωνούν μεταξύ τους μέσω ενός δικτύου. Συνήθως, αυτές οι microservices μπορούν να αναπτυχθούν ανεξάρτητα επειδή έχουν ισχυρό διαχωρισμό ευθυνών μέσω μιας σαφώς καθορισμένης προδιαγραφής με συμβατότητα προς τα πίσω για να αποφευχθεί η ξαφνική διακοπή εξάρτησης.

Τον τελευταίο καιρό αποτελούν το πιο καυτό θέμα στην τεχνολογία και η αρχιτεκτονική των microservices ακολουθείται από τεχνολογικούς γίγαντες όπως το Netflix, το Twitter, το Amazon, το Walmart κ.λπ. καθώς και αρκετές νεοσύστατες επιχειρήσεις. Είναι η τέλεια εφαρμογή για τη σημερινή ευέλικτη διαδικασία ανάπτυξης λογισμικού όπου πραγματοποιείται συνεχής καινοτομία και τα προϊόντα παραδίδονται συνεχώς.

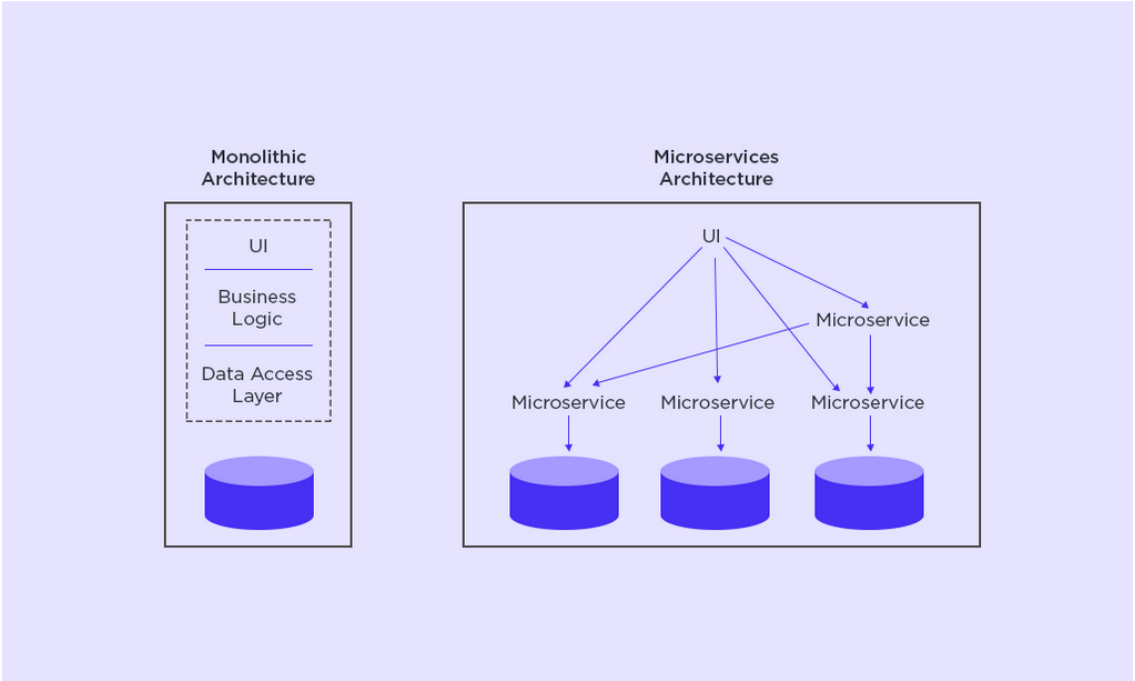
“In microservice architectures, applications are built and deployed as simple, highly decoupled, focussed services. They connect to each other over lightweight language agnostic communication mechanisms, which often times means simple HTTP APIs and message queues and are resilient in nature.”

https://sonusharma-mnnit.medium.com/building-a-microservice-in-python-ff009da83dac

Τα microservices ενδέχεται να περιέχουν αρκετές ανεξάρτητες υπηρεσίες που έχουν δημιουργηθεί για να εξυπηρετούν μόνο μία συγκεκριμένη επιχειρησιακή λειτουργία και να επικοινωνούν μέσω του ελαφρού πρωτοκόλλου όπως το HTTP. Σε μια μονολιθική αρχιτεκτονική, η οποία είναι η μεγαλύτερη αντίπαλος της αρχιτεκτονικής των microservices, όλο το business logic υπάρχει μέσα σε μια ενιαία υπηρεσία, πράγμα το οποίο καθιστά δύσκολη τη συντήρηση, δοκιμή και ανάπτυξη.

Τα οφέλη της αρχιτεκτονικής είναι:

- *Continuous Integration and Deployment (CI/CD): Τα microservices μπορούν να διαχειρίζονται από μία μικρή ομάδα ατόμων και καθώς είναι ανεξάρτητα και αυτόνομα είναι εφικτή η αλλαγή, διαγραφή ή προσθήκη κώδικα χωρίς να επηρεάζεται το σύστημα.*
- *Ανεξαρτησία γλώσσας προγραμματισμού και Framework*
- *Containerization*
- *Υψηλή επεκτασιμότητα, διαθεσιμότητα και ανθεκτικότητα*
- *Ταχύτητα και αποδοτικότητα*
- *Ελαχιστοποίηση ρίσκου*
- *Κάνε ένα πράγμα αλλά κάνε το καλά*



1 Microservices vs Monolithic Architecture https://www.clariontech.com/blog/5-best-technologies-to-build-microservices-architecture

Μειονεκτήματα των microservices:

Δεδομένου ότι οποιοδήποτε νόμισμα έχει δύο πλευρές, οι αρχιτεκτονικές που βασίζονται σε microservices φέρουν επίσης κάποιες δυσκολίες και μειονεκτήματα. Αυτός είναι ο λόγος για τον οποίο τα microservices δεν ταιριάζουν σε κάθε περίπτωση.

Εάν δεν έχουν κατασκευαστεί χρησιμοποιώντας βέλτιστες πρακτικές (όπως συμβαίνει με το Netflix), οι εταιρείες αντιμετωπίζουν πολλές προκλήσεις. Αν και, ακόμη και αν η αρχιτεκτονική έχει κατασκευαστεί σωστά, ενδέχεται να εμφανιστούν ορισμένα προβλήματα:

- *Απαιτεί μεγάλες προσπάθειες για να εφαρμοστεί από το μηδέν ή να αναπαράγεται το μονολιθικό σύστημα.*
- *Περίπλοκες δοκιμές σε κατανεμημένα περιβάλλοντα.*
- *Η πολύπλοκη υποδομή απαιτεί πολλές υπηρεσίες που προκαλούν σύγχυση.*
- *Πιο δύσκολη διαχείριση με αύξηση του αριθμού των υπηρεσιών.*
- *Απαιτείται σωστή ρύθμιση ισορροπίας φορτίου και μορφών μηνυμάτων λόγω καθυστέρησης δικτύου και χαμηλής ανοχής σφαλμάτων.*
- *Μερικές φορές μπορεί να είναι πιο ακριβό λόγω απομακρυσμένων κλήσεων, απομακρυσμένων API κ.λπ.*

Ωστόσο, αυτά τα μειονεκτήματα είναι πιο εύκολο να εξαλειφθούν από τα μειονεκτήματα των μονολιθικών συστημάτων.

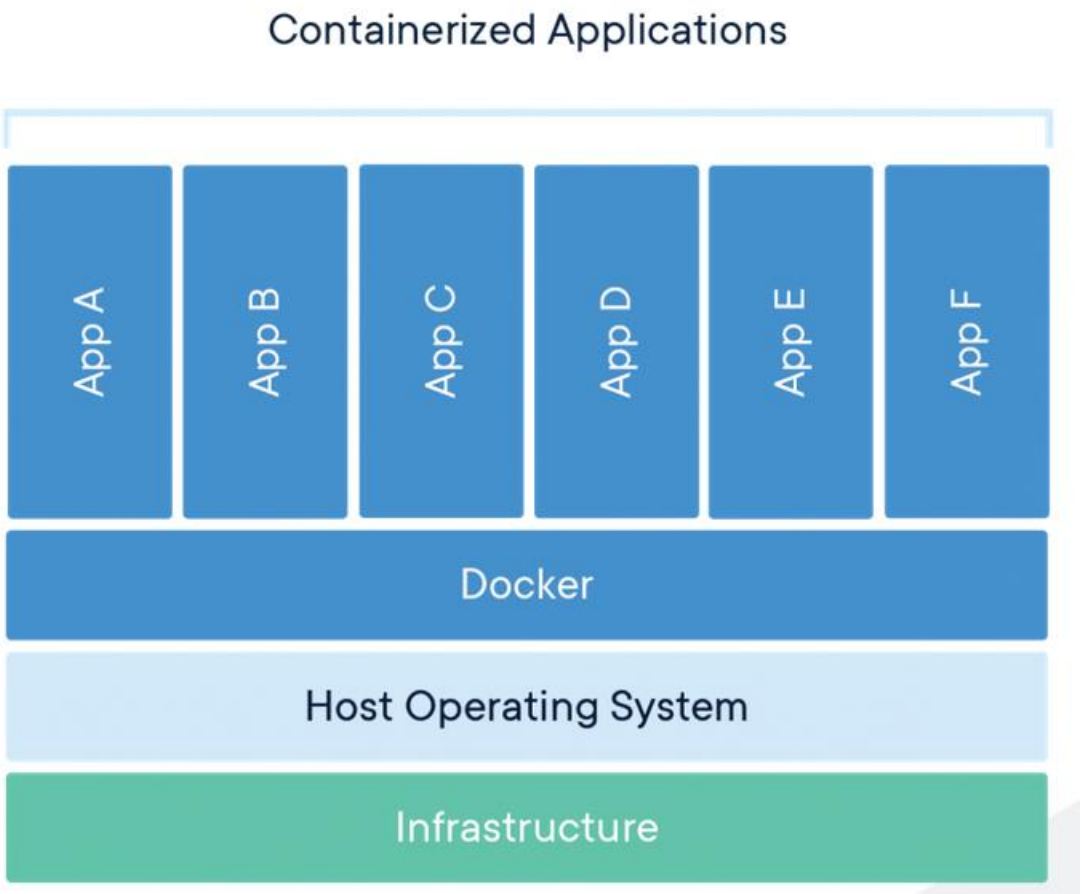
Python for Microservices:

Η Python είναι μια γλώσσα προγραμματισμού υψηλού επιπέδου που προσφέρει ενεργή υποστήριξη για ενσωμάτωση με διάφορες τεχνολογίες. Οι προγραμματιστές που εφαρμόζουν τα Microservices στην Python χρησιμοποιούν μια προσέγγιση RESTful API - έναν ολοκληρωμένο τρόπο χρήσης πρωτοκόλλων ιστού και λογισμικού για χειρισμό αντικειμένων από απόσταση. Με αυτήν την τεχνολογία, γίνεται ευκολότερη η παρακολούθηση της εφαρμογής, δεδομένου ότι χωρίζεται σε στοιχεία. Υπάρχει ένα ευρύ φάσμα πλαισίων microservices στην Python για να διαλέξει κανείς για την ανάπτυξη εφαρμογών ιστού.

3. CONTAINERS &DOCKER

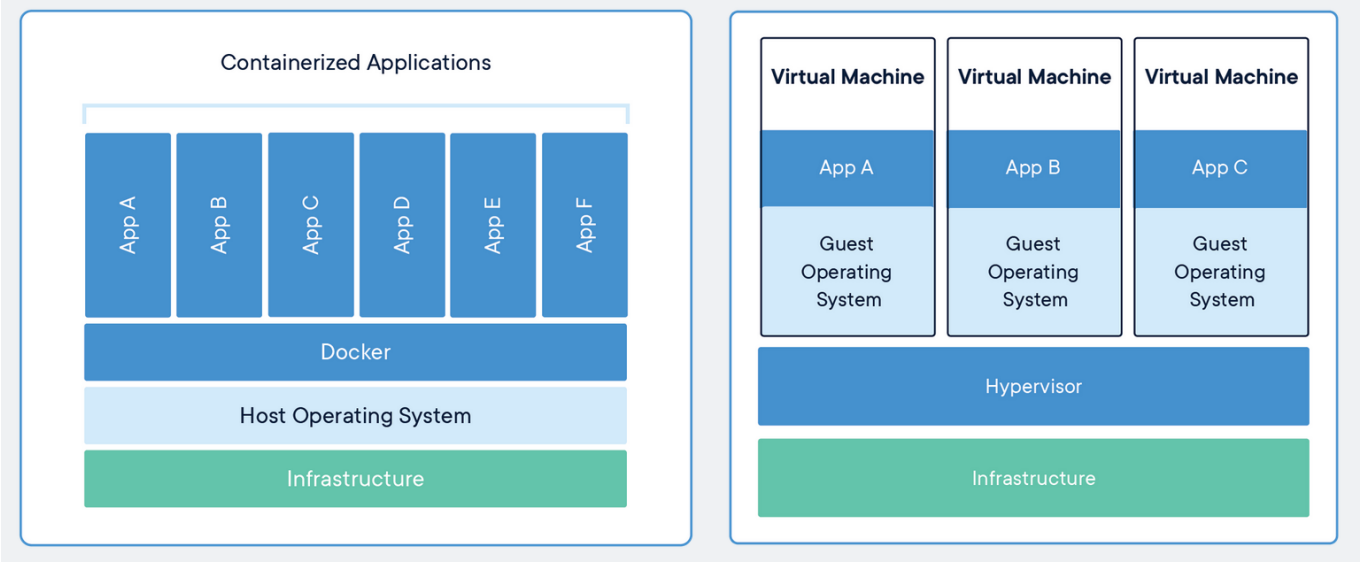
Τι είναι container? Container είναι μία αυτόνομη και ανεξάρτητη μονάδα λογισμικού με κώδικα και όλες τις εξαρτήσεις του, έτσι ώστε η εφαρμογή να εκτελείται γρήγορα και αξιόπιστα από το ένα περιβάλλον υπολογιστών στο άλλο.

Container image είναι ένα ελαφρύ, αυτόνομο, εκτελέσιμο πακέτο λογισμικού που περιλαμβάνει όλα όσα χρειάζονται για την εκτέλεση μιας εφαρμογής: κωδικός, χρόνος εκτέλεσης, εργαλεία συστήματος, βιβλιοθήκες συστήματος και ρυθμίσεις. Τα container images γίνονται container κατά το χρόνο εκτέλεσης και στην περίπτωση των Docker Containers οι εικόνες γίνονται containers όταν εκτελούνται στο Docker Engine. Διαθέσιμο για εφαρμογή σε Linux και Windows, το λογισμικό με container θα λειτουργεί πάντα το ίδιο, ανεξάρτητα από την υποδομή. Τα container απομονώνουν το λογισμικό από το περιβάλλον του και διασφαλίζουν ότι λειτουργεί ομοιόμορφα παρά τις διαφορές, για παράδειγμα, μεταξύ ανάπτυξης και scaling.



2 https://www.docker.com/resources/what-container

Containers vs Virtual Machines:



3 Containers vs VM "https://www.docker.com/resources/what-container"

Container:

Είναι μία αφαίρεση στο επίπεδο εφαρμογής που συμπεριλαμβάνει κώδικα και εξαρτήσεις μαζί. Πολλά container μπορούν να τρέχουν στον ίδιο υπολογιστή και να μοιράζονται τον πυρήνα του OS με άλλα container και το καθένα λειτουργεί ως απομονωμένη διαδικασία στο χώρο του χρήστη. Τα container καταλαμβάνουν λιγότερο χώρο από τα VM (τα container images είναι συνήθως δεκάδες MB σε μέγεθος), μπορούν να χειριστούν περισσότερες εφαρμογές και απαιτούν λιγότερα VM και λειτουργικά συστήματα.

VM:

Οι εικονικές μηχανές (VM) είναι “abstraction of hardware” που μετατρέπει έναν διακομιστή σε πολλούς διακομιστές. Το hypervisor επιτρέπει την εκτέλεση πολλαπλών VM σε ένα μόνο μηχάνημα. Κάθε VM περιλαμβάνει ένα πλήρες αντίγραφο ενός λειτουργικού συστήματος, της εφαρμογής, των απαραίτητων δυαδικών αρχείων και βιβλιοθηκών - καταλαμβάνοντας δεκάδες GB. Η εκκίνηση των VM μπορεί επίσης να είναι αργή.

4. USING DOCKER

Dockerfile:

Το Docker μπορεί να δημιουργήσει εικόνες αυτόματα διαβάζοντας τις οδηγίες από το Dockerfile. Το Dockerfile είναι ένα έγγραφο κειμένου που περιέχει όλες τις εντολές που ένας χρήστης θα μπορούσε να καλέσει στη γραμμή εντολών για να δημιουργήσει μια εικόνα. Η χρήση του “docker build” μπορεί να δημιουργήσει μια αυτοματοποιημένη έκδοση που εκτελεί πολλές διαδοχικές οδηγίες γραμμής εντολών.

A Docker image consists of read-only layers each of which represents a Dockerfile instruction. The layers are stacked and each one is a delta of the changes from the previous layer. Consider this `Dockerfile` :

```
FROM ubuntu:18.04
COPY . /app
RUN make /app
CMD python /app/app.py
```

4 Dockerfile https://docs.docker.com/develop/develop-images/dockerfile_best-practices/

“docker run”:

Εκτελεί το/τα docker image.

Docker Compose:

Το Compose είναι ένα εργαλείο για τον καθορισμό και την εκτέλεση εφαρμογών Docker πολλαπλών container. Με το Compose, χρησιμοποιείτε ένα αρχείο YML για να διαμορφώσει τις υπηρεσίες της εφαρμογής. Στη συνέχεια, με μία μόνο εντολή, δημιουργεί και ξεκινάει όλες τις υπηρεσίες που έχουμε διαμορφώσει.

```yaml
version: "3.9"  # optional since v1.27.0
services:
  web:
    build: .
    ports:
      - "5000:5000"
    volumes:
      - .:/code
      - logvolume01:/var/log
    links:
      - redis
  redis:
    image: redis
volumes:
  logvolume01: {}
```

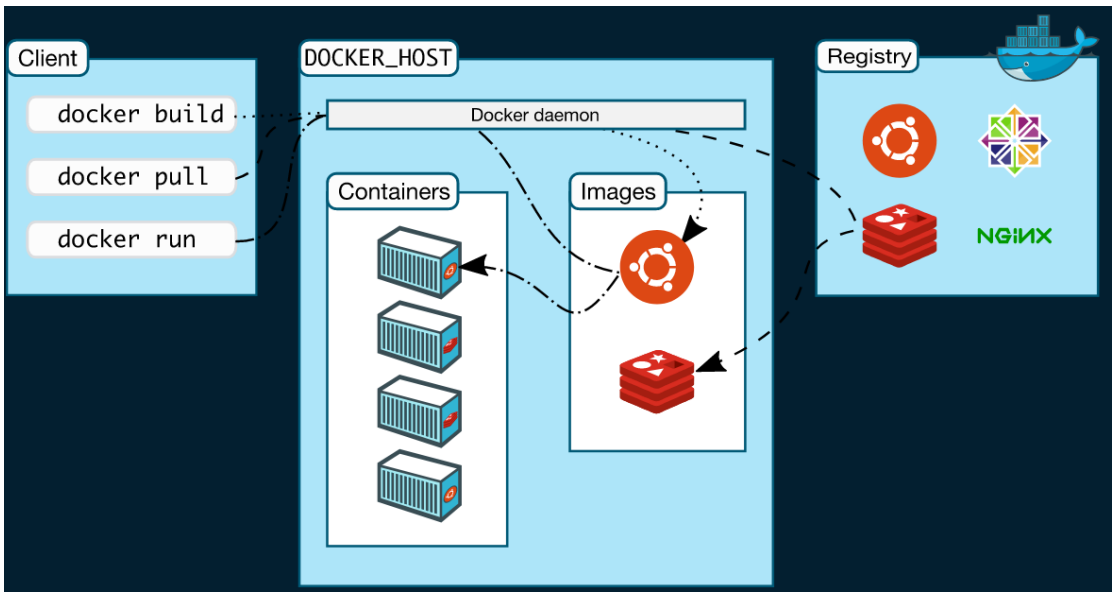
5 A docker-compose.yml file https://docs.docker.com/compose/

Χαρακτηριστικά που το καθιστούν δημοφιλές:

- *Πολλαπλά απομονωμένα περιβάλλοντα σε έναν κεντρικό υπολογιστή*
- *Διατηρεί τα δεδομένα τόμου κατά τη δημιουργία container*
- *Δημιουργεί μόνο container που έχουν αλλάξει*

Docker Architecture:

Το Docker χρησιμοποιεί αρχιτεκτονική πελάτη-διακομιστή. Ο πελάτης Docker μιλά με τον Docker deamon, ο οποίος αναλαμβάνει την «ανύψωση» της λειτουργίας και τη διανομή των container του Docker. Ο Docker client και Docker deamon μπορούν να εκτελούνται στο ίδιο σύστημα ή μπορεί να συνδεθεί ένας Docker client σε έναν απομακρυσμένο Docker deamon. Ο client και ο deamon επικοινωνούν χρησιμοποιώντας ένα REST API, μέσω υποδοχών UNIX ή διασύνδεσης δικτύου.



6 Docker Architecture https://docs.docker.com/get-started/overview/#what-can-i-use-docker-for

Further Information..

Kubernetes stops Docker..?

https://kubernetes.io/blog/2020/12/02/dont-panic-kubernetes-and-docker/

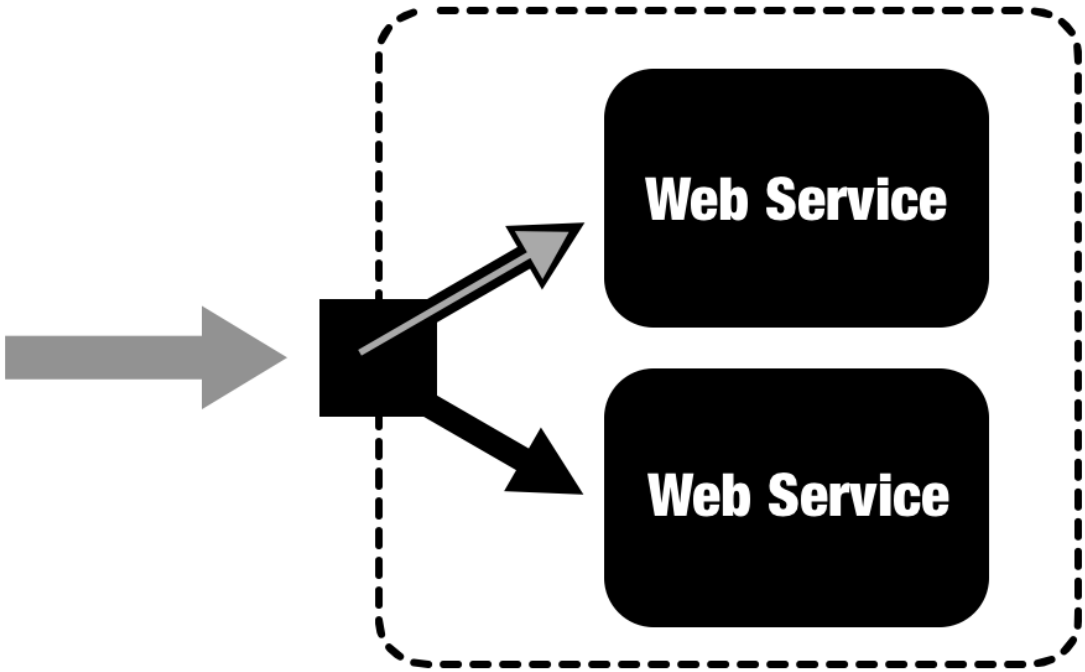
5. THE APPLICATION

Όπως προαναφέραμε στην εφαρμογή που θα παρουσιάσουμε παρακάτω έχουμε δημιουργήσει web services τα οποία επικοινωνούν μεταξύ τους με απλά http requests με τη χρήση του RESTful Flask Web Framework, τα οποία έχουμε εισάγει σε docker containers και θα αναπτύξουμε στο network που θα φτιάξουμε με το docker-compose.

Πριν συνεχίσουμε στην εφαρμογή αυτή καθαυτή είναι σημαντικό να τονίσουμε ένα σημαντικό πλεονέκτημα των web services που βοηθά αρκετά το διαμοιρασμό του φόρτου καθώς και το concurrency:

The load balancer:

Είναι ένα εργαλείο εξισορρόπησης φορτίου που επιτρέπει τη διανομή αιτημάτων HTTP (ή άλλων τύπων δικτύου) αιτήματα) μεταξύ πολλών πόρων backend. Η κύρια λειτουργία ενός load balancer είναι να επιτρέπει τον φόρτο να πηγαίνει προς μία διεύθυνση έτσι ώστε να μοιράζεται η “δουλειά” σε διαφόρους πανομοιότυπους servers ή services ώστε να επιτυγχάνεται μεγαλύτερη ταχύτητα και αποσυμφόρηση.



1 **"Jaime Buelta - Hands-On Docker for Microservices with Python_ Design, deploy, and operate a complex system with multiple microservices"**

Αλλά μπορεί επίσης να χρησιμοποιηθεί για την αντικατάσταση υπηρεσιών. Ο load balancer διασφαλίζει ότι κάθε αίτημα πηγαίνει καθαρά σε ένα service.

Η εφαρμογή λοιπόν είναι ένα σύστημα εμπνευσμένο από τον google assistant ή τη Siri όπου ζητάει ο πελάτης τον καιρό μίας πόλης ή τα κορυφαία πρωτοσέλιδα της ημέρας σε μία χώρα και του επιστρέφει τα αποτελέσματα. Έχουμε 3 υπηρεσίες, μία βασική ο master-server ο οποίος δέχεται απευθείας αιτήματα από τον client-χρήστη είτε για τον καιρό είτε για τα νέα και αναλόγως το αίτημα, αναλαμβάνει την επικοινωνία και την μεταβίβαση του αιτήματος στην ανάλογη υπηρεσία που αφορά το συγκεκριμένο αίτημα. Έπειτα η εκάστοτε υπηρεσία δέχεται το αίτημα, επικοινωνεί με τη βάση δεδομένων και επιστρέφει το αποτέλεσμα στον master και ο master στον client.

Ο τρόπος που είναι χτισμένο το σύστημα, όπου κάθε service έχει μία αποκλειστική ευθύνη και η εφαρμογή της τεχνολογίας docker και docker-compose, καθιστούν την εφαρμογή πάρα πολύ εύκολη σε περαιτέρω αναβάθμιση, διανομή, προσθήκη έξτρα υπηρεσιών και συντήρηση.

Η άντληση δεδομένων καιρού και νέων πραγματοποιήθηκε με free API που υπάρχουν στο διαδίκτυο και θα αναφερθούν στη συνέχεια.

Αναλυτικές οδηγίες για την εκτέλεση του κώδικα και της εφαρμογής υπάρχουν στο αρχείο README.md

Αρχικά για την σύνθεση του κώδικα δημιουργήθηκε ένα virtual environment καθώς αποτελεί σοφή επιλογή για την διαχείριση των εκάστοτε dependencies που μπορεί να χρησιμοποιεί κάθε κώδικας γραμμένος σε python.

```python
from flask import Flask, request
from flask_restful import Resource, Api, reqparse
import json
import os
import requests

app = Flask(__name__)
api = Api(app)

@app.route('/weather')
def weather():
    city = request.args.get('city')
    if city.isdigit():
        res = "City name must be string e.g. 'Amsterdam, Berlin'"
        return res
    response = requests.get("http://weather:3002/weather?city="+ city)
    return response.json()

@app.route('/news')
def news():
    country_name = request.args.get('country')
    if country_name.isdigit() or len(country_name) > 2 :
        resp = "Country name must be string. Choose from below: \n\nThe 2-letter ISO 3166-1 code of the country
        return resp
    response = requests.get("http://news:3003/news?country="+ country_name)
    return response.json()

if __name__ == '__main__':
    app.run(host="0.0.0.0",port=3001,debug=True,threaded=True)
```

VS code: Κώδικας master-server

Έχουν δημιουργηθεί δύο endpoints, ένα για τον καιρό και ένα για τα νέα. Ο server τρέχει στην πόρτα 3001, τα αιτήματα μεταφέρονται με simple http requests σε μορφή json και υπάρχουν και ο βασικός έλεγχος του ονόματος της πόλης και της χώρας αντίστοιχα.

Οι υπηρεσίες news και weather:

```python
from flask import Flask, request
from flask_restful import Resource, Api, reqparse
from json import dumps
from flask import jsonify
import json
import requests

app = Flask(__name__)
api = Api(app)

@app.route('/news')
def news():
    country_name = request.args.get('country')
    api_key = "70dcd1c6a0d24ebdb57ff071ff9b8ddc"
    base_url = "http://newsapi.org/v2/top-headlines?"
    complete_url = base_url + "country=" + country_name + "&apiKey=" + api_key
    response = requests.get(complete_url)
    return response.json()

if __name__ == '__main__':
    app.run(host="0.0.0.0",port='3003',threaded=True,debug=True)
```

2 VS Code: Κώδικας service news

```python
from flask import Flask, request
from flask_restful import Resource, Api, reqparse
from json import dumps
from flask import jsonify
import json
import requests

app = Flask(__name__)
api = Api(app)

@app.route('/weather')
def weather():
    city_name = request.args.get('city')
    api_key = "deb96cff96df7c74e93e62661b91c3c2"
    base_url = "http://api.openweathermap.org/data/2.5/weather?"
    complete_url = base_url + "appid=" + api_key + "&q=" + city_name
    response = requests.get(complete_url)
    return response.json()

if __name__ == '__main__':
    app.run(host="0.0.0.0",port='3002',threaded=True,debug=True)
```

3 VS Code: Κώδικας service weather

Αντίστοιχα βλέπουμε όπως και στον master-server, πως λαμβάνει ως παράμετρο city ό,τι έχει δώσει ο client και το περνάει στο νέο αίτημα.

Αξίζει να σημειωθεί πως ήδη το σύστημα μπορεί να εξυπηρετήσει μεγάλο αριθμό αιτημάτων από διαφορετικούς clients ταυτόχρονα καθώς είναι εφικτό το multithreading τόσο στον master-server όσο και στα επιμέρους services και δεν υπάρχει κάτι μοιραζόμενο ώστε να προκαλέσει σύγχυση στα αιτήματα και να χρειάζεται κάποιο κλείδωμα ή εξωτερική βάση δεδομένων. Εδώ βοηθάει ιδιαίτερα και ο load balancer που αναφέρθηκε παραπάνω.

Τα αιτήματα μπορούν να πραγματοποιηθούν είτε από το terminal είτε από postman είτε με κάποιον έξτρα κώδικα ο οποίος θα λαμβάνει arguments τα ονόματα.

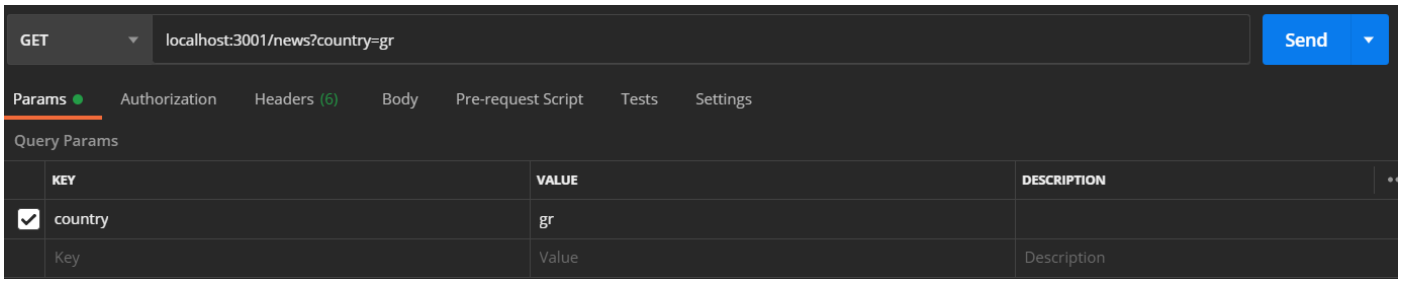
```python
from flask import Flask, request
import requests

def main(argv):

    if len(argv) > 3 or len(argv) < 2:
        print(f'Usage: {argv[0]} <City name> or/and <Country News>')
        exit(1)

    cityStr = argv[1]
    countryStr = argv[2]

    if len(argv) == 2:
        weather(cityStr)
    elif len(argv) == 3:
        news(countryStr)
        weather(cityStr)

def weather(city):
    cityStr = city
    if cityStr.isdigit():
        print("\nCity name must be string e.g. 'Amsterdam, Berlin' ")
        exit(1)
    response = requests.get("http://localhost:3001/weather?city="+ cityStr)
    weather = response.json()
    print(f"\nWeather of {cityStr}: {weather}")
    return weather

def news(country):
    countryStr = country
    if countryStr.isdigit() or len(countryStr) > 2 :
        print("Country name must be string. Choose from below: \n\nThe 2-letter ISO 3166-1 code of the c
        return
    response = requests.get("http://localhost:3001/news?country="+ countryStr)
    news = response.json()
    print(f"{countryStr} News: {news}")
    return news

if __name__ == '__main__':
    main(argv)
```

4 VS Code: Κώδικας Client

Ο client επικοινωνεί μόνο με το βασικό μας server τυπώνει τις απαντήσεις όπως επίσης και μηνύματα λάθους στο χρήστη όσον αφορά τα ονόματα των πόλεων ή χωρών.Παράδειγμα αιτήματος στο terminal:

- *curl -XGET "localhost:3001/weather?city=amsterdam"*

Παράδειγμα αιτήματος στο Postman:



Στη συνέχεια επιχειρούμε να τοποθετήσουμε κάθε service σε ένα docker image για να εκμεταλλευτούμε όλα τα πλεονεκτήματα που αναφέρθηκαν παραπάνω στο αντίστοιχο κεφάλαιο.

Αυτό επιτυγχάνεται με τα Dockerfiles, τα οποία περιέχουν τις απαραίτητες οδηγίες για την κατασκευή του docker image και στη συνέχεια την εκτέλεση του.

Συντέθηκαν τρία Dockerfiles:

```
FROM python:3.8-slim-buster

# Install dependencies:
COPY requirements.txt .
RUN pip install -r requirements.txt

EXPOSE 3001

COPY master_assistant.py .

# Run the application:
CMD python master_assistant.py
```

5 VS Code: Dockerfile master-server

```
FROM python:3.8-slim-buster

# Install dependencies:
COPY requirements.txt .
RUN pip install -r requirements.txt

EXPOSE 3003

COPY news.py .

# Run the application:
CMD python news.py
```

6 VS Code: Dockerfile news service

```
FROM python:3.8-slim-buster

# Install dependencies:
COPY requirements.txt .
RUN pip install -r requirements.txt

EXPOSE 3002

COPY weather.py .

# Run the application:
CMD python weather.py
```

7 VS Code: Dockerfile weather service

Το σύστημα είναι έτοιμο να ενορχηστρωθεί και να «ανέβει». Με τη χρήση του docker-compose αυτό θα γίνει με μία μόνο εντολή «docker-compose up --build».

Παρακάτω το αρχείο docker-compose.yml όπου αξίζει να σημειωθεί πως γίνεται ένωση της πόρτας 3001 του network που φτιάξαμε με αυτήν του υπολογιστή μας “localhost”.

Ορίζουμε τα services, τις εντολές και τα Dockerfiles.

```
version: "3.9"
services:
  master:
    build:
      context: .
      dockerfile: Dockerfile
    ports:
      - "3001:3001"
  weather:
    build:
      context: .
      dockerfile: Dockerfile.weather
  news:
    build:
      context: .
      dockerfile: Dockerfile.news
```

8 VS Code: Docker Compose

Γίνεται build και τα services ανεβαίνουν και «τρέχουν» στο network που έχουμε δημιουργήσει.

Ενδιαφέρον έχει η αλλαγή που χρειάστηκε να γίνει στα requests από τον master-server προς τα services καθώς πλέον δεν απευθυνόμαστε σε localhost.

```python
@app.route('/weather')
def weather():
    city = request.args.get('city')
    if city.isdigit():
        res = "City name must be string e.g. 'Amsterdam, Berlin'"
        return res
    response = requests.get("http://weather:3002/weather?city="+ city)
    return response.json()
```

9 VS Code: Change of requests

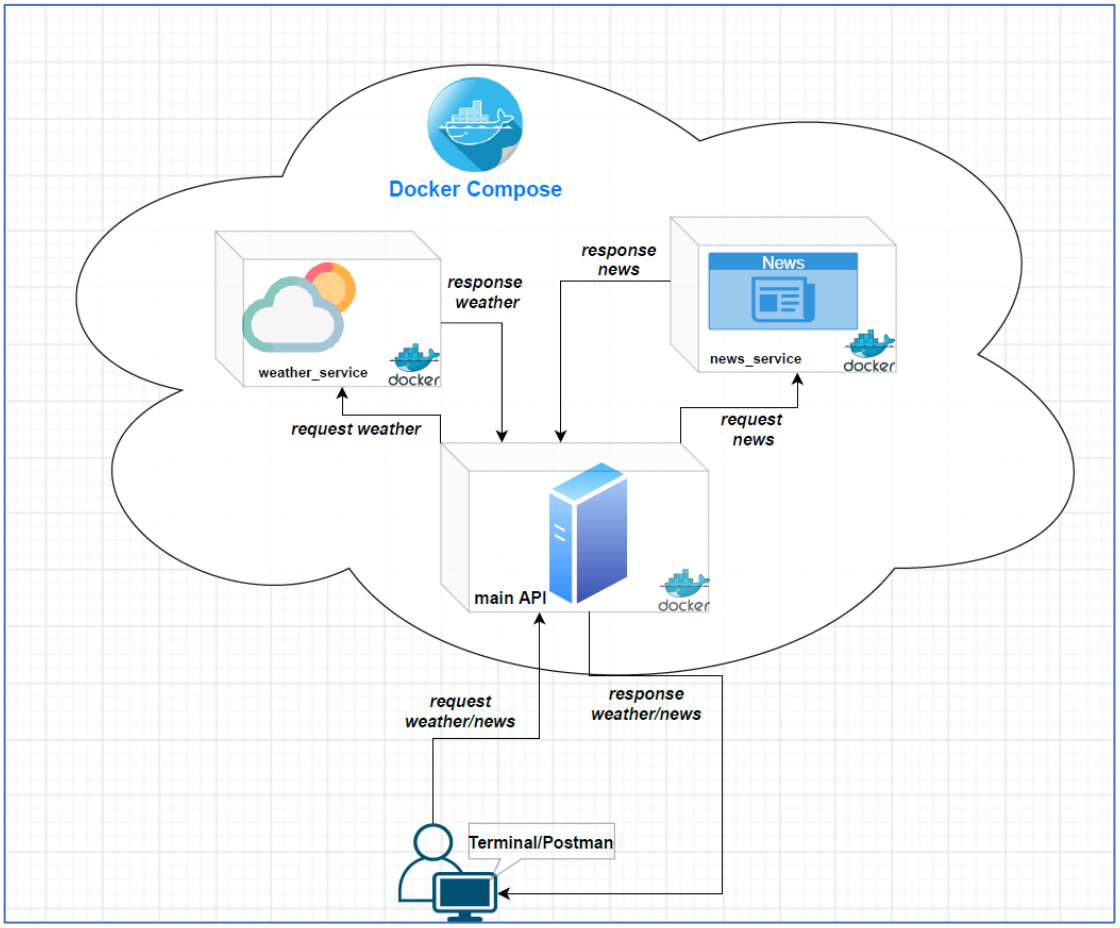
Εκτελώντας την εντολή docker-compose το αποτέλεσμα είναι αυτό:

```
 ---> Using cache
 ---> 3b3c52279c09
Step 5/6 : COPY news.py .
 ---> Using cache
 ---> 107388797bad
Step 6/6 : CMD python news.py
 ---> Using cache
 ---> cf3cd97cfca9
Successfully built cf3cd97cfca9
Successfully tagged python_service_news:latest
Starting python_service_news_1     ... done
Recreating python_service_master_1 ... done
Starting python_service_weather_1  ... done
Attaching to python_service_weather_1, python_service_news_1, python_service_master_1
weather_1  |  * Serving Flask app "weather" (lazy loading)
weather_1  |  * Environment: production
weather_1  |    WARNING: This is a development server. Do not use it in a production deployment.
weather_1  |    Use a production WSGI server instead.
weather_1  |  * Debug mode: on
weather_1  |  * Running on http://0.0.0.0:3002/ (Press CTRL+C to quit)
weather_1  |  * Restarting with stat
news_1     |  * Serving Flask app "news" (lazy loading)
news_1     |  * Environment: production
news_1     |    WARNING: This is a development server. Do not use it in a production deployment.
news_1     |    Use a production WSGI server instead.
news_1     |  * Debug mode: on
news_1     |  * Running on http://0.0.0.0:3003/ (Press CTRL+C to quit)
news_1     |  * Restarting with stat
master_1   |  * Serving Flask app "master_assistant" (lazy loading)
master_1   |  * Environment: production
master_1   |    WARNING: This is a development server. Do not use it in a production deployment.
master_1   |    Use a production WSGI server instead.
master_1   |  * Debug mode: on
master_1   |  * Running on http://0.0.0.0:3001/ (Press CTRL+C to quit)
master_1   |  * Restarting with stat
weather_1  |  * Debugger is active!
weather_1  |  * Debugger PIN: 861-916-554
news_1     |  * Debugger is active!
news_1     |  * Debugger PIN: 115-715-979
master_1   |  * Debugger is active!
master_1   |  * Debugger PIN: 286-712-536
master_1   | 172.21.0.1 - - [01/Feb/2021 21:41:45] "GET /weather?city=5 HTTP/1.1" 200 -
weather_1  | 172.21.0.4 - - [01/Feb/2021 21:41:52] "GET /weather?city=amsterdam HTTP/1.1" 200 -
master_1   | 172.21.0.1 - - [01/Feb/2021 21:41:52] "GET /weather?city=amsterdam HTTP/1.1" 200 -
weather_1  | 172.21.0.4 - - [01/Feb/2021 21:41:56] "GET /weather?city=rf HTTP/1.1" 200 -
master_1   | 172.21.0.1 - - [01/Feb/2021 21:41:56] "GET /weather?city=rf HTTP/1.1" 200 -
master_1   | 172.21.0.1 - - [01/Feb/2021 21:42:16] "GET /news?country=5 HTTP/1.1" 200 -
news_1     | 172.21.0.4 - - [01/Feb/2021 21:42:22] "GET /news?country=gb HTTP/1.1" 200 -
master_1   | 172.21.0.1 - - [01/Feb/2021 21:42:22] "GET /news?country=gb HTTP/1.1" 200 -
```

10 VS Code: Microservices running with docker compose

Η εφαρμογή είναι πλέον υλοποιημένη με την αρχιτεκτονική των microservices και τη χρήση της τεχνολογίας Docker.

Για την σαφέστερη και ξεκάθαρη κατανόηση του networking της εφαρμογής παρουσιάζεται γράφημα:

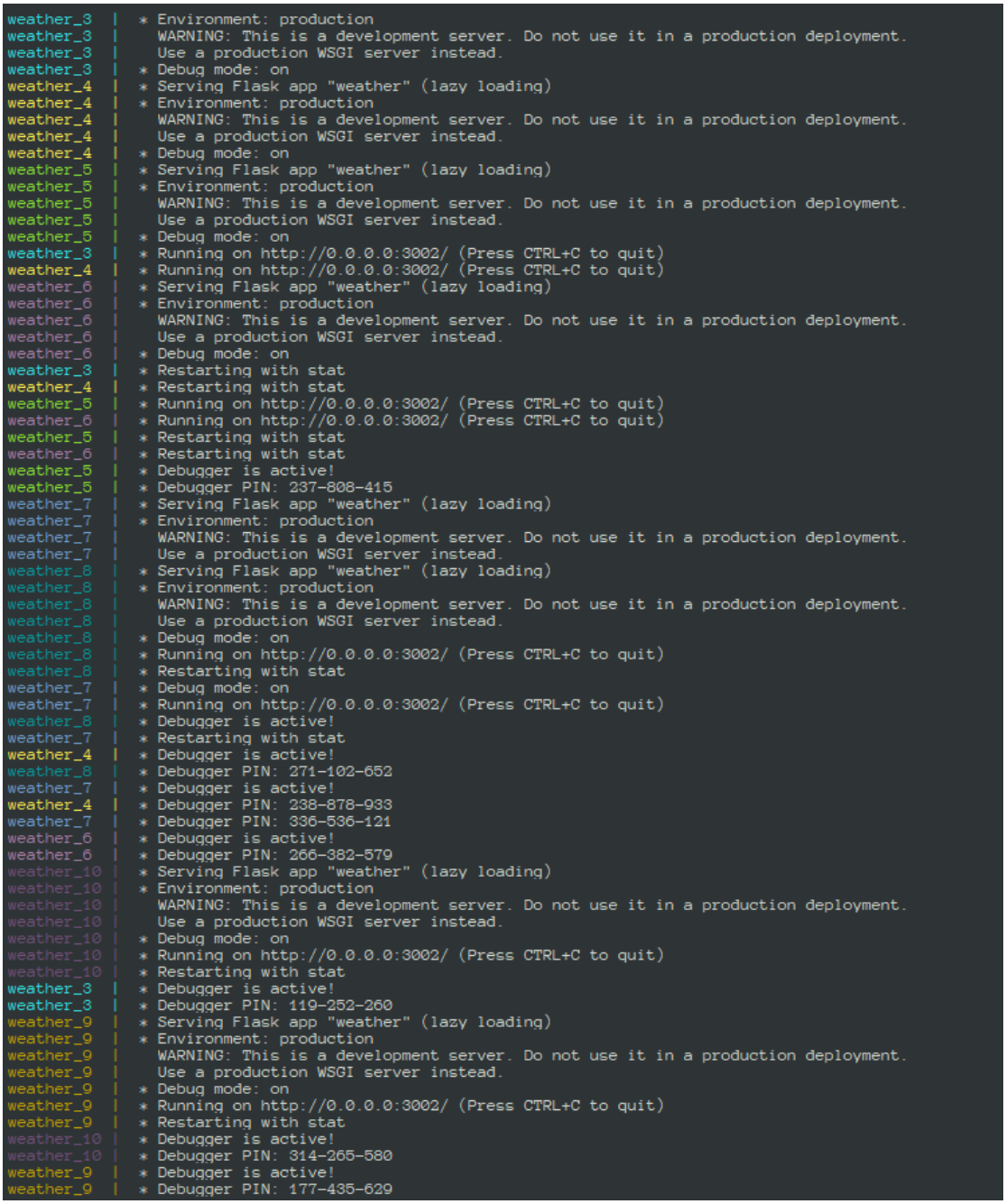


11 Architecture Overview

Σε αυτό το σημείο, είναι ιδιαίτερα σημαντικό να αναφερθεί ένα βασικό και τρομερά χρήσιμο και ενδιαφέρον feature του docker-compose το scale.

Με την εντολή “docker-compose up --build --scale (a_service) = x”

Το docker-compose δημιουργεί x instances-κλώνους του a_service τα οποία λειτουργούν αυτόνομα και ορθά ως ξεχωριστές διεργασίες. Αποτελεί μία άμεση και ανώδυνη λύση σε περίπτωση υψηλού φόρτου αιτημάτων προς ένα service.



12 VS Code: Παράδειγμα scale για weather=10

6. ΒΙΒΛΙΟΓΡΑΦΙΑ

- https://alpacked.io/blog/microservices-use-cases
- https://www.clariontech.com/blog/5-best-technologies-to-build-microservices-architecture
- https://code.visualstudio.com/docs/remote/containers
- https://www.docker.com/resources/what-container
- "Practical Docker with Python Build, Release and Distribute your Python App with Docker, Sathyajith Bhat Bangalore, Karnataka, India"
- "Hands-On Docker for Microservices with Python, Jaime Buelta"
- https://docs.docker.com/compose/
- https://kubernetes.io/blog/2020/12/02/dont-panic-kubernetes-and-docker/
- https://stackoverflow.com/
- https://docs.docker.com/get-started/overview/#what-can-i-use-docker-for
- https://docs.docker.com/engine/reference/builder/
- https://developers.redhat.com/blog/2016/09/15/writing-microservices-an-example-through-a-simple-to-do-application/
- https://medium.com/@sonusharma.mnnit/building-a-microservice-in-python-ff009da83dac
- https://www.fullstackpython.com/microservices.html
- https://www.otto.de/jobs/technology/techblog/artikel/why-microservices_2016-03-20.php