ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΣΧΕΔΙΟ ΠΑΡΟΥΣΙΑΣΗΣ
ΑΤΟΜΙΚΗΣ ΔΙΠΛΩΜΑΤΙΚΗΣ ΕΡΓΑΣΙΑΣ

Μάιος 2021

Ατομική Διπλωματική Εργασία

ΠΡΟΣΟΜΟΙΩΣΗ ΣΥΣΤΗΜΑΤΟΣ IOT ΓΙΑ ΤΗΝ ΠΑΡΑΚΟΛΟΥΘΗΣΗ ΚΑΙ ΕΛΕΓΧΟ ΚΕΝΤΡΩΝ ΔΕΔΟΜΕΝΩΝ ΤΡΟΦΟΔΟΤΟΥΜΕΝΩΝ ΑΠΟ ΑΝΑΝΕΩΣΙΜΕΣ ΠΗΓΕΣ ΕΝΕΡΓΕΙΑΣ

Θεοφάνης Ιωακείμ

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μάιος 2021

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΠΡΟΣΟΜΟΙΩΣΗ ΣΥΣΤΗΜΑΤΟΣ IOT ΓΙΑ ΤΗΝ ΠΑΡΑΚΟΛΟΥΘΗΣΗ ΚΑΙ ΕΛΕΓΧΟ ΚΕΝΤΡΩΝ ΔΕΔΟΜΕΝΩΝ ΤΡΟΦΟΔΟΤΟΥΜΕΝΩΝ ΑΠΟ ΑΝΑΝΕΩΣΙΜΕΣ ΠΗΓΕΣ ΕΝΕΡΓΕΙΑΣ

Θεοφάνης Ιωακείμ

Επιβλέπων Καθηγητής
Μάριος Δικαιάκος

Η Ατομική Διπλωματική Εργασία υποβλήθηκε προς μερική εκπλήρωση των απαιτήσεων απόκτησης του πτυχίου Πληροφορικής του Τμήματος Πληροφορικής του Πανεπιστημίου Κύπρου

Μάιος 2021

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

1.1 Σκοπός και συνεισφορά

Σκοπός αυτής της διπλωματικής είναι η δημιουργία ενός συστήματος προσομοίωσης, για επιχειρηματίες ή προγραμματιστές που θέλουν να προσομοιώσουν ένα σύστημα από φωτοβολταϊκά και υποδομές έτσι ώστε να εκμεταλλευτούν στο έπακρον την ηλεκτρική ενέργεια που παράγουν οι συστοιχίες των φωτοβολταϊκών τους πριν επενδύσουν και αναπτύξουν τέτοια πάρκα. Η χρήση ενός τέτοιου συστήματος μπορεί να βοηθήσει διάφορους επιχειρηματίες να προσομοιώσουν ένα σύστημα φωτοβολταϊκών κοντά σε διάφορες μονάδες που χρειάζονται ηλεκτρική ενέργεια και να παρακολουθήσουν το όφελος που θα τους προσφέρουν αυτές οι συστοιχίες στους λογαριασμούς ηλεκτρικής ενέργειας και τη μείωση του αποτυπώματος άνθρακα στον πλανήτη πριν προχωρήσουν στην αγορά των φωτοβολταϊκών. Ακόμα θα μπορούν να συγκρίνουν την παραγωγή και εξοικονόμηση ενέργειας με βάση την τοποθεσία και το configuration των συστοιχιών. Αυτό ισχύει και για προγραμματιστές που θέλουν να προσομοιώσουν ένα σύστημα με διάφορα microservices για σκοπούς πειραμάτων.

Για παράδειγμα ας υποθέσουμε ότι ένας επιχειρηματίας έχει διάφορα κέντρα δεδομένων σε διάφορα σημεία του κόσμου τα οποία καταναλώνουν ηλεκτρική ενέργεια από μη ανανεώσιμες πηγές. Αυτά τα κέντρα δεδομένων καταναλώνουν μεγάλες ποσότητες ηλεκτρικής ενέργειας και κατά συνέπεια ένα ποσό από το εισόδημα της εταιρείας. Ακόμα λόγω της μεγάλης κατανάλωσης έχουν αρνητικό αντίκτυπο στον πλανήτη αφού παράγουν μεγάλες ποσότητες διοξειδίου του άνθρακα. Ο επιχειρηματίας αποφασίζει να επενδύσει σε ανανεώσιμες πηγές

ενέργειας όπως φωτοβολταϊκά τα οποία θα παρέχουν ηλεκτρική ενέργεια στα κέντρα δεδομένων της εταιρείας. Σε αυτό το σημείο σκέφτεται ποιος θα ήταν ο καλύτερος τρόπος να επενδύσει τα χρήματά του έτσι ώστε να αξιοποιήσει το μέγιστο των δυνατοτήτων των φωτοβολταϊκών. Εδώ έρχεται το σύστημα που υλοποιήσαμε, απλά και γρήγορα μπορεί να προσομοιώσει την λειτουργία φωτοβολταϊκών πάρκων στις τοποθεσίες των κέντρων δεδομένων, να συγκρίνει την μείωση της κατανάλωσης σε κάθε υποδομή και να κάνει την επόμενη κίνηση για αγορά των φωτοβολταϊκών έχοντας τα δεδομένα που χρειάζεται. Ακόμα μπορεί να δώσει διάφορα configurations για τις συστοιχίες και να πετύχει την μέγιστη αποδοτικότητα τους. Το συγκεκριμένο σύστημα θα μπορούσε να έχει και άλλες εφαρμογές. Για παράδειγμα θα μπορούσε να χρησιμοποιηθεί και από ένα προγραμματιστή που θέλει να δει πόση ηλεκτρική ενέργεια μπορεί να παραχθεί από ένα σύστημα φωτοβολταϊκών με κάποια συγκεκριμένα χαρακτηριστικά. Το πρόβλημα είναι ότι δεν έχει πρόσβαση στα απαραίτητα δεδομένα και δεν μπορεί να δαπανήσει χρήματα για την ανέγερση τους. Έτσι χρησιμοποιεί το σύστημα που δημιουργήσαμε για να προσομοιώσει τις συστοιχίες και την ενέργεια που θα παραχθεί από τα φωτοβολταϊκά. Μόλις περιγράψει το σύστημα που θέλει να προσομοιώσει μπορεί να προσθέσει και άλλες υπηρεσιες για να γίνει πιο ολοκληρωμένη η προσομοίωση. Έτσι δεν χαμηλώνει μόνο το κόστος ενός τέτοιου έργου αλλά εξοικονομεί πόρους και χρόνο.

1.2 Παρακίνηση

Το ενδιαφέρον για την παραγωγή ηλεκτρικής ενέργειας με την χρήση ανανεώσιμων πηγών ενέργειας συνεχίζει να αυξάνεται τις τελευταίες δεκαετίες, καθώς όλο και περισσότεροι άνθρωποι βλέπουν τα περιβαλλοντικά οφέλη τους. Η ηλιακή ενέργεια είναι μια πρακτικά ανεξάντλητη ανανεώσιμη πηγή, η οποία αναμένεται να διαδραματίσει αυξανόμενο ρόλο στη μελλοντική υποδομή παραγωγής ηλεκτρικής ενέργειας. Τα φωτοβολταϊκά συστήματα έχουν αναπτυχθεί προκειμένου να παράγουν ηλεκτρισμό μετατρέποντας την ηλιακή ακτινοβολία σε ηλεκτρική ενέργεια.

Η εγκατάσταση φωτοβολταϊκών σε ένα πάρκο είναι μια δαπανηρή διαδικασία που πολλές φορές απαιτεί ενα μεγάλο κεφάλαιο από τον ενδιαφερόμενο. Γι’ αυτό το λόγο, παρόλο που η σωστή εγκατάσταση των φωτοβολταικών μπορεί να είναι πολύ ωφέλιμη για τον ενδιαφερόμενο, η “τυφλή” εγκατάστασή τους σε μια περιοχή ή σε κάποιο πάρκο ενδέχεται να διακατέχεται από τον κίνδυνο μη πλήρης αξιοποίησης του συστήματος για μέγιστη παραγωγή.

Οπόταν υπάρχει ανάγκη για την ανάπτυξη ενός εργαλείου, μέσω του οποίου ο ενδιαφερόμενος θα μπορεί να προσομοιώσει την επίδοση και την παραγωγή των φωτοβολταικών δοκιμάζοντας διάφορους τρόπους εγκατάστασης, έτσι ώστε να μεγιστοποιηθεί η παραγωγή ηλεκτρικής ενέργειας.

1.3 Δομή

Η δομή της διπλωματικής είναι η εξής:

Κεφάλαιο 2: Υψηλου επιπέδου περιγραφή λύσης και τεχνολογιών που χρησιμοποιήθηκαν.

Κεφάλαιο 3: Λεπτομερής περιγραφή της υλοποίησης του συστήματος και εξαγωγή αποτελεσμάτων.

Κεφάλαιο 2

Περιγραφή Λύσης

2.1 ΕΝΕΔΗ

Η λύση είναι βασισμένη στο έργο ΕΝΕΔΗ το οποίο μελετά και εφαρμόζει μεθόδους μεγιστοποίησης ενεργειακής απόδοσης και έξυπνης διαχείρισης ενέργειας με κατάλληλη διαμόρφωση των φορτίων που δημιουργεί η λειτουργία των υπολογιστικών και αποθηκευτικών συστημάτων των κέντρων δεδομένων, και γίνεται εγκατάσταση φωτοβολταϊκών για την παραγωγή μέρους της αναγκαίας ηλεκτρικής ενέργειας. Το έργο προτείνει την εισαγωγή τεχνολογιών με την εγκατάσταση Ανανεώσιμων Πηγών Ενέργειας υποστηρίζοντας τη μετάβαση προς μια οικονομία χαμηλών εκπομπών διοξειδίου του άνθρακα στον απαιτητικό τομέα των κέντρων δεδομένων για τις ανάγκες της εκπαίδευσης και της έρευνας.

Το σύστημα ΕΝΕΔΗ και τα συστήματα ενεργειακής διαχείρισης της κατανομής υπολογιστικού φορτίου έχουν ως αποτέλεσμα τη μείωση της συνολικής κατανάλωσης ενέργειας και τη μείωση των εκπεμπόμενων αερίων του θερμοκηπίου και της ατμοσφαιρικής ρύπανσης. Παράλληλα, η εγκατάσταση φωτοβολταϊκών συστημάτων για τη παραγωγή μέρους της απαιτούμενης ηλεκτρικής ενέργειας αυξάνει το ποσοστό συνεισφοράς με την παραγωγή ενέργειας από Ανανεώσιμες Πηγές Ενέργειας.

Ειδικότερα, στο πλαίσιο της διπλωματικής εργασίας στόχος είναι να προσομοιωθεί ένα μέρος του συστήματος ΕΝΕΔΗ και η συμβολή της εγκατάστασης ενός φωτοβολταικού συστήματος, η εγκατάσταση μετρητικών διατάξεων κατανάλωσης ηλεκτρικής ενέργειας και των παρεμβάσεων σε ένα κέντρο δεδομένων, τα οποία συνιστούν ένα πλέγμα δράσεων τα οποία συμβάλουν στην εξοικονόμηση ενέργειας, στη μείωση των λειτουργικών δαπανών και στη μείωση του περιβαλλοντικού αποτυπώματος. Η λύση που περιγράφεται πιο κάτω είναι ανεξάρτητη από το σύστημα ΕΝΕΔΗ και έχει διάφορες εφαρμογές ανάλογα με το αποτέλεσμα που θέλει να πετύχει ο κάθε χρήστης του συστήματος.

Η εξοικονόμηση ενέργειας είναι άμεση και έμμεση. Η άμεση εξοικονόμηση ενέργειας προέρχεται από τη λειτουργία του φωτοβολταικού συστήματος. Η μείωση των λειτουργικών εξόδων για ενέργεια συνεπάγεται την αποδέσμευση πόρων για αναπτυξιακές χρήσεις. Επίσης, η μείωση της εκπομπής Διοξειδίου του Άνθρακα συνεπάγεται τον περιορισμό της ατμοσφαιρικής ρύπανσης και τη μείωση του περιβαλλοντικού αποτυπώματος. Η έμμεση εξοικονόμηση ενέργειας προέρχεται από τη λειτουργία του πληροφοριακού συστήματος στο οποίο καταγράφονται τα δεδομένα από τις μετρητικές διατάξεις, ώστε να μπορούν να χρησιμοποιηθούν για παρεμβάσεις σε επίπεδο λειτουργίας αλλά και υλικοτεχνικής και κτηριακής υποδομής στοχεύοντας στην ενεργειακή εξοικονόμηση και την μείωση του ενεργειακού αποτυπώματος μέσω της εξοικονόμησης ενέργειας του κέντρου δεδομένων.

2.2 Περιγραφή Προσομοιωτή Φωτοβολταικών

Ένα κομμάτι του συστήματος είναι ο προσομοιωτής φωτοβολταϊκών ο οποίος προσομοιώνει το API του Fronius ενώ προσομοιώνει και τις τιμές που επιστρέφει αυτό όπως η παραγωγή ενέργειας για οποιαδήποτε στιγμή, η ημερήσια, ετήσια και συνολική παραγωγή μιας συστοιχίας, η θερμοκρασία, η ταχύτητα του ανέμου και η ακτινοβολία.

2.2.1 Fronius

Οι Fronius Inverters είναι αποδοτικοί και αξιόπιστοι και αποτελούν την απαραίτητη καρδιά κάθε φωτοβολταϊκού συστήματος. Οι κατηγορίες ισχύος που κυμαίνονται από 1,5 έως 100 kW εγγυώνται την καταλληλότητα για σχεδόν οποιοδήποτε μέγεθος συστήματος, από ένα οικογενειακό σπίτι έως ένα σύστημα μεγάλης κλίμακας. Οι μετατροπείς Fronius παρέχουν λύσεις επικοινωνίας δεδομένων για φωτοβολταϊκά συστήματα, διασφαλίζοντας αξιόπιστη παρακολούθηση συστήματος και άμεση ενσωμάτωση σε άλλα συστήματα. Τέλος παρέχουν στους χρήστες όχι μόνο το hardware αλλά και το software έτσι ώστε να μπορούν να παρακολουθούν διάφορες

μετρικές και μέσω του api που προσφέρουν να γίνει η ενσωμάτωση με άλλα συστήματα.

Ο προσομοιωτής των φωτοβολταϊκών που υλοποιήθηκε απαντά στην μορφή του Fronius Api αφού οι inverters στο πανεπιστήμιο χρησιμοποιούν το συγκεκριμένο api για να επιστρέψουν τις τιμές που έχουν την συγκεκριμένη στιγμή οι αισθητήρες των φωτοβολταϊκών. Για να επιστέψει δεδομένα πρέπει να γίνει ένα HTTP request σε ένα συγκεκριμένο FCGI. Το FastCGI ή FCGI είναι ένα binary πρωτόκολλο για την αλληλεπίδραση με έναν web server μέσω διαδραστικών προγραμμάτων. Είναι μια παραλλαγή της προηγούμενης τεχνολογίας Common Gateway Interface (CGI). Ο κύριος στόχος του FastCGI είναι να μειώσει την επιβάρυνση που σχετίζεται με την επικοινωνία του web server και των εφαρμογών CGI, επιτρέποντας σε έναν διακομιστή να χειρίζεται περισσότερα αιτήματα ανά μονάδα χρόνου.

Οι πιο σημαντικές κλήσεις που υλοποιήθηκαν είναι οι /solar_api/v1/GetPowerFlowRealtimeData.fcgi και /solar_api/v1/GetSensorRealtimeData.cgi. Περισσότερες λεπτομέρειες σε αυτές τις κλήσεις θα συζητηθούν πιο κάτω. Ο προσομοιωτής επιστέφει τα δεδομένα δομημένα με τον ίδιο τρόπο με τους πραγματικούς αισθητήρες σε μορφή json έτσι ώστε να γίνει πιο εύκολη η προσαρμογή των τεχνολογιών που ήδη χρησιμοποιούνται στο υφιστάμενο σύστημα.

Για να γίνει χρήση του προσομοιωτή ο χρήστης θα πρέπει να δώσει ένα configuration file σε μορφή json το οποίο θα περιγράφει τις συστοιχίες που θέλει να προσομοιώσει στην πιο κάτω μορφή με παραμέτρους όπως ο αριθμός, η ηλικία, το μέγεθος, η αποδοτικότητα, η κλίση και ο τύπος των πάνελ καθώς και οι απώλειές συστήματος, η τοποθεσία, ο τύπος εγκατάστασης, η χωρητικότητα και το performance ratio της συστοιχίας.

Sample Configuration:


```
{"parks":[
        {
"panels":"63","panels_age":"1","panels_size":"1.7","panels_efficency":"17","panel_type":"0","system_losses":"14","location":"aglantzia","i
nclination":"17","installation_type":"0","azimuth":"180","electricity_rate_type":"residential","electricity_rate":"0.233","pr":"79.26","capacit
y":"17.01","type":"114"},
        {
"panels":"34","panels_age":"1","panels_size":"1.7","panels_efficency":"17","panel_type":"0","system_losses":"14","location":"aglantzia","i
nclination":"17","installation_type":"0","azimuth":"180","electricity_rate_type":"residential","electricity_rate":"0.233","pr":"79.26","capacit
y":"9.18","type":"232"
        },
          {
"panels":"56","panels_age":"1","panels_size":"1.7","panels_efficency":"17","panel_type":"0","system_losses":"14","location":"aglantzia","i
nclination":"17","installation_type":"0","azimuth":"180","electricity_rate_type":"residential","electricity_rate":"0.233","pr":"79.26","capacit
y":"15.12","type":"115"
        }
          ]
}
```

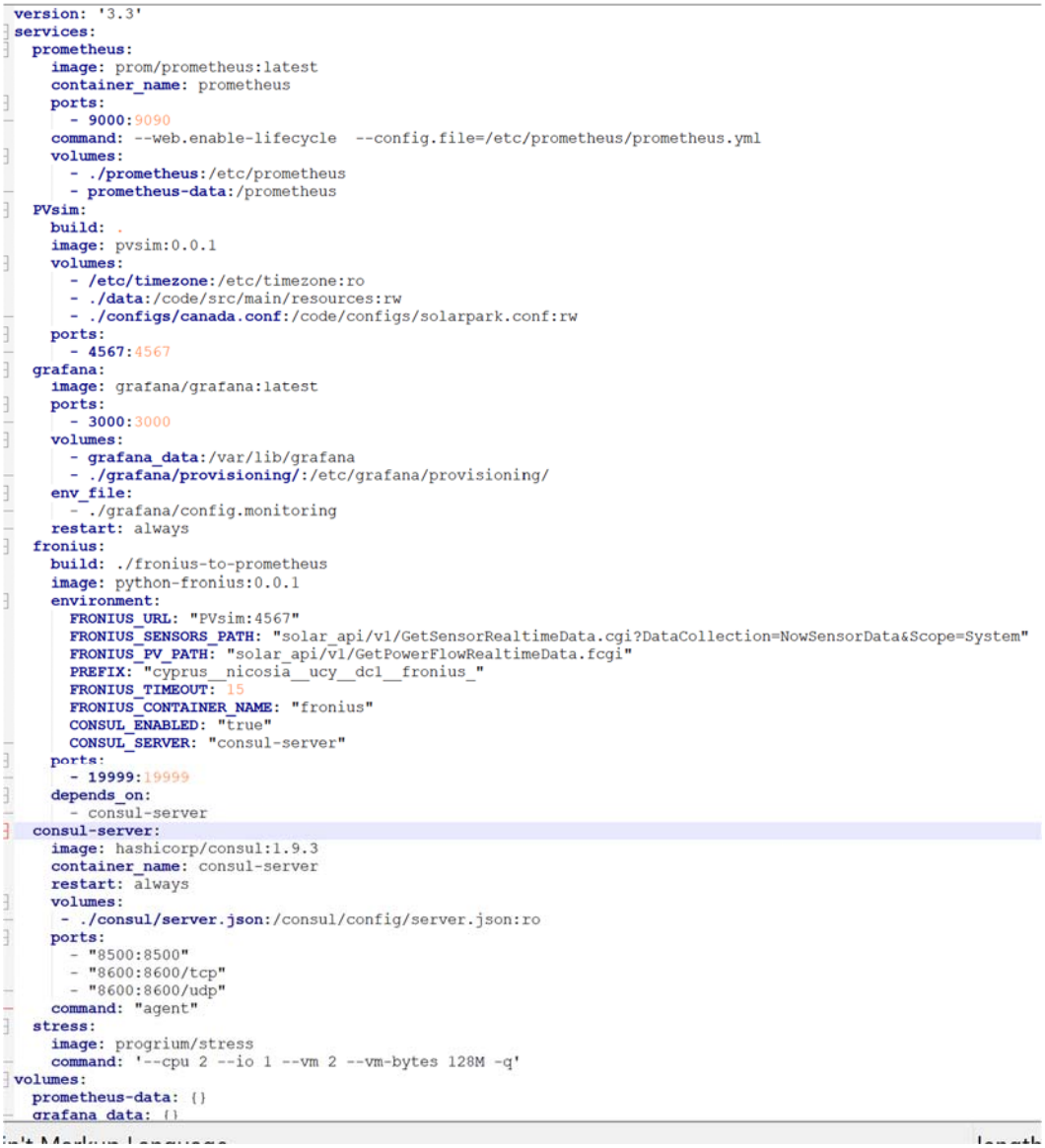
Description of Parameters:
Interval: interval between polling data in ms
panels: Number of panels in array
panels_age: Panels age in years,
panels_size: Panels size in array (square meters),
panels_efficency: Panels efficency around 15-22%, number is a percentage,
panel_type: Panel type. Allowed typed:0- standard, 1-premium, 2-thin film,
system_losses: The system losses account for performance losses you would expect in a real system,
location: Location of the array(string). The conversion to coordinates is done automatically. Location can be city or town name(q=London,united+kingdom),ip address(XXX.XXX.XXX.XXX),Latitude and longitude seperated by comma(XX.XXX,XX.XXX in decimal degrees),
inclination: Inclination of panels in degrees -- A zero tilt angle means that the face of the panel is aimed directly overhead. A positive tilt angle means that the panel faces more towards the equator. In the northern hemisphere that would mean tilting so it faces towards the South. Rarely, the tilt angle can be negative; this means the panel faces away from the equator,
installation_type: 0-fixed,1-1_axis_tracking,2-1_axis_backtracking,3-2_axis_tracking
azimuth: the azimuth angle is the angle clockwise from true north describing the direction that the array faces,
electricity_rate_type: commercial or residential,
electricity_rate: in euro/kWh -- 0.233 for residential and 0.213 for commercial,
pr: Performance Ratio - the ratio of the actual and theoretically possible energy outputs (0-100),
capacity: (kWp),
type: type of array (integer)

Όταν ο προσομοιωτής λάβει ένα αίτημα διαβάζει το configuration file που έδωσε ο χρήστης και δημιουργεί ένα αντικείμενο που προσομοιώνει την συστοιχία. Η δομή του configuration file επιτρέπει την προσομοίωση πολλών συστοιχιών απλά με την πρόσθεση των επιπλέον παραμέτρων κάτω από τα υφιστάμενα στο json του configuration. Ακόμα υπάρχει η δυνατότητα να τρέξουμε πολλούς προσομοιωτές με διαφορετικά configurations και endpoints με την χρήση του docker.

2.2.2 Docker - Compose

Το Docker αυτοματοποιεί την ανάπτυξη εφαρμογών λογισμικού σε containers παρέχοντας ένα επιπλέον επίπεδο αφαιρετικότητας και αυτοματοποίησης έτσι ώστε να ενσωματώσουμε τον κώδικα που θα προσομοιώνει τα components του συστήματος σε containers και να μπορέσουμε να έχουμε πολλά instances των προγραμμάτων όταν θα προσομοιώσουμε το σύστημα σε μεγαλύτερη κλίμακα.

Το Compose είναι ένα εργαλείο για τον καθορισμό και την εκτέλεση εφαρμογών Docker πολλαπλών κοντέινερ. Με το Compose, χρησιμοποιείται ένα αρχείο YAML για να διαμορφώσουμε τις υπηρεσίες της εφαρμογής μας. Ένα αρχείο YAML έχει την πιο κάτω μορφή.



```yaml
version: '3.3'
services:
  prometheus:
    image: prom/prometheus:latest
    container_name: prometheus
    ports:
      - 9000:9090
    command: --web.enable-lifecycle  --config.file=/etc/prometheus/prometheus.yml
    volumes:
      - ./prometheus:/etc/prometheus
      - prometheus-data:/prometheus
  PVsim:
    build: .
    image: pvsim:0.0.1
    volumes:
      - /etc/timezone:/etc/timezone:ro
      - ./data:/code/src/main/resources:rw
      - ./configs/canada.conf:/code/configs/solarpark.conf:rw
    ports:
      - 4567:4567
  grafana:
    image: grafana/grafana:latest
    ports:
      - 3000:3000
    volumes:
      - grafana_data:/var/lib/grafana
      - ./grafana/provisioning/:/etc/grafana/provisioning/
    env_file:
      - ./grafana/config.monitoring
    restart: always
  fronius:
    build: ./fronius-to-prometheus
    image: python-fronius:0.0.1
    environment:
      FRONIUS_URL: "PVsim:4567"
      FRONIUS_SENSORS_PATH: "solar_api/v1/GetSensorRealtimeData.cgi?DataCollection=NowSensorData&Scope=System"
      FRONIUS_PV_PATH: "solar_api/v1/GetPowerFlowRealtimeData.fcgi"
      PREFIX: "cyprus__nicosia__ucy__dc1__fronius_"
      FRONIUS_TIMEOUT: 15
      FRONIUS_CONTAINER_NAME: "fronius"
      CONSUL_ENABLED: "true"
      CONSUL_SERVER: "consul-server"
    ports:
      - 19999:19999
    depends_on:
      - consul-server
  consul-server:
    image: hashicorp/consul:1.9.3
    container_name: consul-server
    restart: always
    volumes:
     - ./consul/server.json:/consul/config/server.json:ro
    ports:
      - "8500:8500"
      - "8600:8600/tcp"
      - "8600:8600/udp"
    command: "agent"
  stress:
    image: progrium/stress
    command: '--cpu 2 --io 1 --vm 2 --vm-bytes 128M -q'
volumes:
  prometheus-data: {}
  grafana_data: {}
```

Figure 1, Docker-compose.yaml (Source code)

Στη συνέχεια, με μία εντολή, δημιουργούμε και ξεκινούμε όλες τις υπηρεσίες μας. Η χρήση του Compose είναι βασικά μια διαδικασία τριών βημάτων. Ορίζουμε το περιβάλλον της εφαρμογής με ένα Dockerfile, ώστε να μπορεί να αναπαραχθεί οπουδήποτε. Ορίζουμε τις υπηρεσίες που αποτελούν την εφαρμογή μας στο docker-compose.yml, ώστε να μπορούν να εκτελούνται μαζί σε ένα απομονωμένο περιβάλλον. Τέλος ξεκινά η εντολή σύνθεσης Docker και εκτελείτε ολόκληρη η εφαρμογή. Αυτός είναι και ο τρόπος που θα εκτελούμε όλες τις υπηρεσίες του συστήματος που θα αναφερθούν πιο κάτω έτσι ώστε να έχουμε ένα επίπεδο αφαιρετικότητας στο περιβάλλον της εφαρμογής.

Για κάθε συστοιχία γίνονται δυο requests για να πάρει την τοποθεσία και τα δεδομένα που χρειάζονται για την προσομοίωση. Τα δεδομένα από τις διάφορες απαντήσεις των api αποθηκεύονται σε αρχεία έτσι ώστε να έχουμε ελάχιστα request προς αυτά. Για παράδειγμα

αν γίνει ένα request για την εύρεση των συντεταγμένων μιας περιοχής και γίνει επανεκκίνηση του προσομοιωτή δεν θα γίνει κάποιο request στο αντίστοιχο api αλλά θα διαβαστεί η τιμή που ήδη αποθηκεύσαμε από το προηγούμενο run. Ακολουθώντας αυτή την λογική αν είχαμε κάποια ιστορικά δεδομένα από κάποια άλλη πηγή θα μπορούσαμε να τα προσθέσουμε στα υφιστάμενα δεδομένα από τα οποία διαβάζει ο προσομοιωτής έτσι ώστε να έχουμε πιο ακριβή αποτελέσματα.

Το πρώτο request γίνεται στο api.positionstack.com/v1/forward?access_key=api_key&query=location_name&limit=1&output=json το οποίο θα μετατρέψει την τοποθεσία από φυσική γλώσσα σε συντεταγμένες έτσι ώστε να μπορούμε να τις περάσουμε στο api καιρού που ακολουθεί. Σε αυτή την κλήση θα πρέπει να δώσουμε σαν parameters το api_key, το όνομα τις τοποθεσίας που διαβάσαμε από το configuration file, το όριο συντεταγμένων που θα μας επιστρέψει και το είδος της απάντησης που θέλουμε που είναι json. Η απάντηση του api έχει την εξής μορφή:

{"data":[{"latitude":35.154412,"longitude":33.392108,"type":"locality","name":"Aglandjia","number":null,"postal_code":null,"street":null,"confidence":1,"region":"Nicosia","region_code":"NI","county":null,"locality":"Aglandjia","administrative_area":null,"neighbourhood":null,"country":"Cyprus","country_code":"CYP","continent":"Asia","label":"Aglandjia, NI, Cyprus"}]}

Αυτό που περνούμε από την απάντηση είναι το latitude και το longitude για κάθε περιοχή που έγινε το request και την αποθηκεύουμε. Το επόμενο request γίνεται στο https://api.solcast.com.au/world_radiation/forecasts?format=json&latitude=lat&longitude=lon&hours=168&api_key=api_key το οποίο θα μας επιστρέψει τα δεδομένα καιρού που χρειαζόμαστε για να γίνει το simulation. Θα πρέπει να περάσουμε το είδος απάντησης που είναι και πάλι json, το latitude και το longitude που πήραμε από το προηγούμενο api call, για πόσες ώρες θέλουμε να μας δώσει δεδομένα – 168 είναι ο μέγιστος αριθμός που μπορούμε να βάλουμε και το api key το οποίο πήραμε από την ιστοσελίδα όπου δημιουργήσαμε τον λογαριασμό στο solcast (https://solcast.com/). Όσο περισσότερα δεδομένα καιρού έχουμε τόσο πιο ακριβείς θα είναι οι τιμές που θα επιστρέφει ο προσομοιωτής. Η απάντηση έχει ως εξής:

{"forecasts":[{"ghi":0,"ghi90":0,"ghi10":0,"ebh":0,"dni":0,"dni10":0,"dni90":0,"dhi":0,"air_temp":18,"zenith":105,"azimuth":61,"cloud_opacity":20,"period_end":"2021-04-27T18:00:00.0000000Z","period":"PT30M"},{"ghi":0,"ghi90":0,"ghi10":0,"ebh":0,"dni":0,"dni10":0,"dni90":0,"dhi":0,"air_temp":18,"zenith":111,"azimuth":55,"cloud_opacity":17,"period_end":"2021-04-27T18:30:00.0000000Z","period":"PT30M"}, …. }

Forecast

Attributes	Description	Details
ghi	int	Global Horizontal Irradiance (W/m2) - centre value (mean)
ghi90	int	Global Horizontal Irradiance (W/m2) - 90th percentile value (high scenario)
ghi10	int	Global Horizontal Irradiance (W/m2) - 10th percentile value (low scenario)
dni	int	Direct Normal Irradiance (W/m2) - centre value (mean)
dni10	int	Direct Normal Irradiance (W/m2) - 10th percentile value (low scenario)
dni90	int	Direct Normal Irradiance (W/m2) - 90th percentile value (high scenario)
dhi	int	Diffuse Horizontal Irradiance
air_temp	int	Air temperature (degrees Celsius)
zenith	int	Solar zenith angle (degrees). Zero means directly upwards/overhead. Varies from 0 to 180. A value of 90 means the sun is at the horizon.
azimuth	double	Solar azimuth angle (degrees). Zero means true north. Vaies from -180 to 180. A value of -90 means the sun is in the east.
cloud_opacity	int	The attenuation of incoming light due to cloud. Varies from 0 (no cloud) to 100 (full attenuation of incoming light).
period_end	datetime	End of the averaging period in ISO8601 datetime format in UTC timezone
period	string	Length of the averaging period in ISO8601 duration format

Figure 2, https://docs.solcast.com.au/?_ga=2.220675600.989272553.1651069256-2015426038.1651069256#forecasts-by-location

Τα δεδομένα που μας επιστρέφει είναι τα ghi, ebh, dni, dhi, air_temp, zenith, azimuth, cloud_opacity, period_end, period και περιγράφονται πιο πάνω από το documentation του solcast. Οι τιμές επιστρέφονται ανά ώρα γι' αυτό και έχουμε κάποια κενά στα αποτελέσματα της προσομοίωσης που πρέπει να μετατρέψουμε σε ρεαλιστικά δεδομένα. Ακόμα υπάρχει περιορισμός σε 10 κλίσεις την μέρα. Εμείς αποθηκεύουμε μόνο τα δεδομένα για τις επόμενες 24 ώρες άρα μπορούμε να κάνουμε κλίσεις για 10 τοποθεσίες την μέρα. Υπάρχει και η δυνατότητα εξαγωγής ιστορικών δεδομένων αφού όλα τα δεδομένα που παίρνουμε από το api και τα δεδομένα που παράγονται από τον προσομοιωτή αποθηκεύονται σε αρχεία. Από προεπιλογή τα δεδομένα εξάγονται σε αρχεία csv για την κάθε τοποθεσία με την οποία έτρεξε ο προσομοιωτής.

Στην συνέχεια για το κάθε πάνελ παράγονται οι τιμές με βάση το μοντέλο παραγωγής ενέργειας που επιλέξαμε και στο τέλος όλες οι τιμές συνδυάζονται για να βγουν οι τελικές τιμές.

2.2.3 Μοντέλο παραγωγής ενέργειας

Το μοντέλο παραγωγής ενέργειας που χρησιμοποιήσαμε είναι το μοντέλο που χρησιμοποιείτε από το PVWatts (https://pvwatts.nrel.gov/) το οποίο μας δόθηκε από το FOSS Research Centre for Sustainable Energy (https://www.foss.ucy.ac.cy/).

```
PVWatts model:
means <- 48 #granularity e.g. 30-minute intervals
# Global variables
# Reference Irradiance (default value is 1000 W/m2)
Eo = 1000
# Reference Temperature (default value is 25 C)
To = 25
# convective heat transfer component not considered (power temperature coeffcient)
g = -0.0042
# PV module rated power (KW)
Pmp0 = 400
# Derating factors
Soiling      = 1 #was 2
Shading      = 0 #was 3
Snow         = 0 #was 0
Mismatch  = 1 #was 2
Wiring       = 1 #was 2
Connections              = 0.5 #was 0.5
LID          = 0 #was 1
Nameplate =              1  #was 1
Age          = 1 #was 0
Availability = 3 #was 3
a = -3.47       # Thermal Properties Isc (%/K)
b = -0.0594      # Thermal Properties Voc (%/K)
WD <- #wind direction
WS <- #wind speed
Gpoa <- #incident irradiance (or global horizontal irradiance)
Tamb <- #ambient temperature
NOCT <- 48 # The best module operated at a NOCT of 33°C, the worst at 58°C and the typical module at 48°C
S <- 80#insolation in mW/cm2
Tm <- Tamb+((NOCT-20)/80)*S# module temperature (calculated from ambient temperature)
# Predicted power production with PVwatts Model
for(i in 1:48){
  #Check for the level of irradiance
  if (Gpoa [i] >=125)
    Pmp_predicted[i] <- ((Gpoa [i]/Eo) * Pmp0 * (1 + g * (Tm[i] - To))) * ((100 - Deratingfactor)/100)
  else
    Pmp_predicted[i] <- ((0.008 * (Gpoa [i]^2) /Eo) * Pmp0 * (1 + g * (Tm[i] - To))) * ((100 - Deratingfactor)/100)
}
```

Τέλος επιστέφουμε την απάντηση σε μορφή json με την βοήθεια του java spark πίσω στον χρήστη η οποία είναι όμοια με την απάντηση του Fronius Api.

2.2.4 Spark Java

Το Spark Framework είναι ένα απλό και εκφραστικό πλαίσιο DSL Java / Kotlin που δημιουργήθηκε για ταχεία ανάπτυξη. Η πρόθεση του Spark είναι να παρέχει μια εναλλακτική λύση για προγραμματιστές Kotlin / Java που θέλουν να αναπτύξουν τις εφαρμογές τους στο διαδίκτυο όσο το δυνατόν πιο εκφραστικές και με ελάχιστη επανάληψη κώδικα. Στο πλαίσιο της διπλωματικής εργασίας χρησιμοποιήσαμε το Spark για την υλοποίηση του Fronius api αφού δίνεται η δυνατότητα της υλοποίησης

του προσομοιωτή σε java και η επιστροφή των αποτελεσμάτων μέσω ενός http transaction όπως γίνεται στο api του Fronius. Ακόμα μέσω του spark η διαδικασία του containerize γίνεται πιο απλή.

```
BasicConfigurator.configure();
// port(8080);
get("/solar_api/v1/GetPowerFlowRealtimeData.fcgi", (request, response) -> {
    String timestamp = request.queryParams("Timestamp").replace("T", " ");
    String unixtimestamp = request.queryParams("UnixTimestamp");
    return GetPowerFlowRealtimeData(init(), timestamp, unixtimestamp);
});
get("/solar_api/v1/GetSensorRealtimeData.cgi", (request, response) -> {
    String timestamp = request.queryParams("Timestamp").replace("T", " ");
    String unixtimestamp = request.queryParams("UnixTimestamp");
    String scope = request.queryParams("Scope");
    String datacollection = request.queryParams("DataCollection");
    String deviceid = request.queryParams("DeviceId");
    return GetSensorRealtimeData(scope, datacollection, deviceid, init(), timestamp, unixtimestamp);
});
```

Figure 3, Endpoint Creation with Spark Java (Source code)

2.3 Constraints - Dependencies

Ο προσομοιωτής είναι εξαρτώμενος σε μεγάλο βαθμό από τα δεδομένα που παίρνει από το api καιρού αφού αν τα δεδομένα που χρησιμοποιεί δεν είναι ακριβή θα έχουμε μεγάλες αποκλίσεις στις τιμές που θα παράγονται με βάση το μοντέλο παραγωγής ενέργειας. Ακόμα το συγκεκριμένο api που χρησιμοποιείται από το λογισμικό δεν μας δίνει απεριόριστα δεδομένα. Στην πραγματικότητα μας δίνει 10 requests κάθε μέρα με αποτέλεσμα να μπορούμε να κάνουμε προσομοίωση για 10 τοποθεσίες κάθε μέρα και κατ' επέκταση να σηκώσουμε 10 προσομοιωτές στο σύστημα αυτή την στιγμή. Για να αντιμετωπίσουμε αυτό τον περιορισμό κάνουμε μόνο ένα request και αποθηκεύουμε δεδομένα διάρκειας 24ων ωρών για αυτή την τοποθεσία και δεν κάνουμε άλλο request αλλά διαβάζουμε από το προηγούμενο μέχρι να μην υπάρχουν αλλά δεδομένα. Ακόμα ένας περιορισμός είναι το ότι αν θέλουμε ιστορικά δεδομένα θα πρέπει να τα αιτηθούμε από την σελίδα που περνούμε τα δεδομένα αφού δεν υπάρχει κάποιο api call ενώ το συγκεκριμένο request για ιστορικά δεδομένα έχει μεγάλο κόστος. Για να αντιμετωπίσουμε αυτό το πρόβλημα μπορούμε να αιτηθούμε ιστορικά δεδομένα και απλά να προσθέσουμε το αρχείο στον χώρο του προγράμματος ή μπορούμε να αφήσουμε το πρόγραμμα να τρέχει για αρκετό καιρό και να χτίσει ένα ιστορικό αφού έχουμε προνοήσει με κάθε νέο api call όλα τα δεδομένα του να προθέτονται στα προηγούμενα δεδομένα και έτσι χτίζουμε ιστορικά δεδομένα από την στιγμή που ξεκίνησε να τρέχει το λογισμικό για μια συγκεκριμένη τοποθεσία. Όταν το πρόγραμμα κάνει ένα νέο αίτημα τότε μπορεί να περάσουν μέχρι και 5 δευτερόλεπτα μέχρι να δώσει απάντηση αλλά για όλα τα επόμενα αιτήματα είναι αρκετά πιο γρήγορο αφού τα δεδομένα είναι ήδη στην μνήμη. Τέλος υπάρχει ο περιορισμός ότι δεν μπορούν να τρέχουν 2 διεργασίες με configuration την ίδια τοποθεσία διότι δεν έχει υλοποιηθεί ταυτόχρονη πρόσβαση στα αρχεία. Τα δεδομένα καιρού αποθηκεύονται και διαβάζονται σε αρχεία με το όνομα της τοποθεσίας έτσι αν έχουμε 2 διεργασίες να διαβάζουν και να γράφουν

στο ίδιο αρχείο δεν μπορούμε να εγγυηθούμε το concurrency του προγράμματος. Για να το αντιμετωπίσουμε μπορούμε να δημιουργούμε την κάθε διεργασία με μια μοναδική τοποθεσία και προσθέτουμε στο configuration file του όσες συστοιχίες θέλουμε για την συγκεκριμένη περιοχή.

2.4 Software Product Features

- Το λογισμικό μπορεί να προσομοιώσει πολλές συστοιχίες ανά διεργασία-container με πολλές διαφορετικές παραμέτρους όπως αριθμό φωτοβολταικών, τύπο, μέγεθος, τοποθεσία, αποδοτικότητα, απώλειες συστήματος, κλίση, azimuth και χωρητικότητα συστήματος.
- Μικρός χρόνος εκτέλεσης.
- Προσομοίωση Fronius inverter και δεδομένων για μια χρονική περίοδο.
- Παρακολούθηση δεδομένων και παραγωγή διαγραμμάτων.
- Υπολογισμός ημερήσιας και ετήσιας παραγωγής ενέργειας.
- Αποθήκευση και επαναχρησιμοποίηση δεδομένων.
- Προσθήκη νέων υπηρεσιών που δεν είναι ενσωματωμένες στο αρχικό σύστημα
- Παρακολούθηση και απεικόνιση δεδομένων
- Εξαγωγή σημαντικών μετρικών και αποτελεσμάτων
- Δημιουργία και εφαρμογή ειδοποιήσεων

2.5 Software System Attributes

Αξιοπιστία: Σωστές εισόδους και αξιόπιστα δεδομένα από το api και τον χρήστη είναι απαραίτητα για να εξασφαλιστούν καλά αποτελέσματα.
Διαθεσιμότητα: Καλή σύνδεση στο Διαδίκτυο και μηχανή που μπορεί να βρεθεί από το δίκτυο
Ασφάλεια: Οποιοσδήποτε μπορεί να χρησιμοποιήσει το api, αρκεί να γνωρίζει τη διεύθυνση IP του μηχανήματος που λειτουργεί και είναι ανιχνεύσιμο από το δίκτυο
Συντηρησιμότητα: Το λογισμικό είναι χωρισμένο σε ενότητες έτσι ώστε να έχει καλή συντήρηση, πολλές κλάσης και αντικείμενα για κάθε χαρακτηριστικό και να συνδέεται κατά τρόπο ώστε να είναι εύκολο να κάνετε διορθώσεις και αλλαγές.
Φορητότητα: Το λογισμικό χρειάζεται μόνο λειτουργική σύνδεση στο Διαδίκτυο και ο χρήστης μπορεί να έχει πρόσβαση σε αυτόν από οποιονδήποτε υπολογιστή.

Κεφάλαιο 3

Υλοποίηση Συστήματος

3.1 Υλοποίηση

Περνάμε στην υλοποίηση του προγράμματος που προσομοιώνει τα φωτοβολταϊκά και συγκεκριμένα το Fronius. Το πρόγραμμα είναι υλοποιημένο σε java ενώ η διαχείριση του γίνεται με την βοήθεια του Maven. Όπως φαίνεται στο πιο κάτω διάγραμμα κλάσεων το πρόγραμμα ακολουθεί την εξής μεθοδολογία. Η κλάση Main είναι υπεύθυνη να αναλύσει τα διάφορα input από τον χρήστη, να καλέσει την κατάλληλη μέθοδο κατά την παραλαβή ενός http request και να δημιουργήσει τις κλάσεις solarPark για κάθε φωτοβολταικό πάρκο που υπάρχει στο configuration file που δόθηκε. Ακόμα είναι υπεύθυνη να δημιουργήσει την απάντηση όπως αυτή είναι ορισμένη στο api του Fronius. Η κλάση solarPark με την σειρά της θα αρχικοποιήσει τις διάφορες μεταβλητές που έδωσε ο χρήστης και θα δημιουργήσει πολλά instances την κλάσης solarPanel οι οποίες συμβολίζουν τα φωτοβολταϊκά της κάθε συστοιχίας.

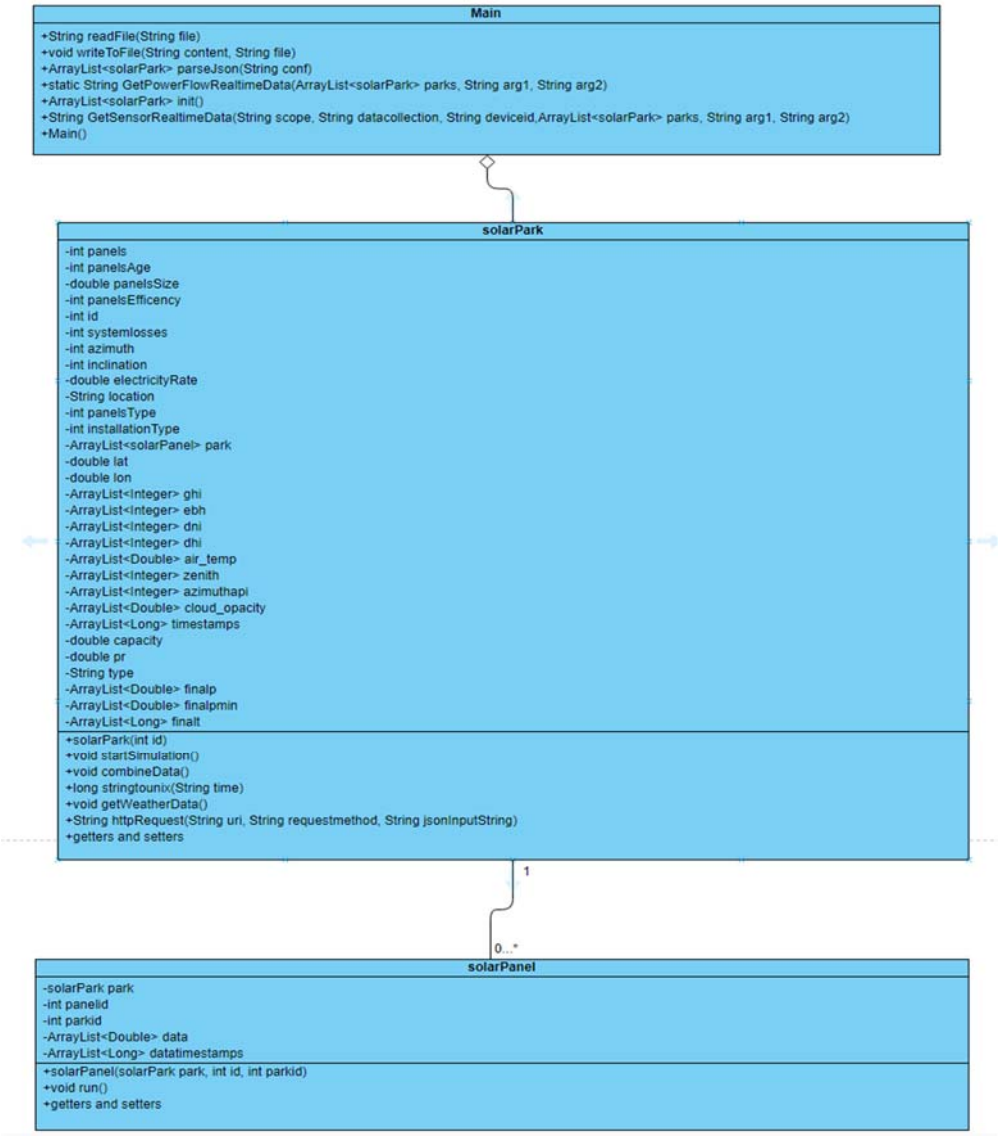


Figure 4, Class Diagram (Created using https://online.visual-paradigm.com/)

Αφού όλες οι βιβλιοθήκες έχουν φορτωθεί μέσω του Maven από το αρχείο pom.xml χρησιμοποιούμε το spark java για να δημιουργήσουμε τα 2 endpoints που χρησιμοποιεί το api του Fronius για να επιστρέψει δεδομένα, το /solar_api/v1/GetPowerFlowRealtimeData.fcgi και το /solar_api/v1/GetSensorRealtimeData.cgi. Όταν φτάσει μια αίτηση περνάμε τις διάφορες παραμέτρους στις αντίστοιχες μεθόδους. Στην περίπτωση της αίτησης GetPowerFlowRealtimeData καλείται η μέθοδος init για την αρχικοποίηση των solarPark κλάσεων αφού γίνει το parsing του configuration file. Στην συνέχεια θα ξεκινήσει το simulation για κάθε πάρκο καλώντας την startSimulation από την κλάση solarPark. Αυτή με την σειρά της θα καλέσει την getWeatherData για να πάρει δεδομένα από το api καιρού. Θα προσπαθήσει να διαβάσει τα δεδομένα που αποθηκεύτηκαν από προηγούμενες εκτελέσεις πριν γίνει κάποιο api call αφού όπως αναφέρθηκε πριν ήμαστε περιορισμένοι από αυτό. Στην συνέχεια θα μετατρέψει την τοποθεσία που δόθηκε σε συντεταγμένες με την χρήση του api.positionstack.com και θα αποθηκεύσει το αποτέλεσμα. Μετά αν δεν υπάρχουν τα απαραίτητα δεδομένα θα γίνει ένα api call στο api.solcast.com.au/world_radiation/forecasts το

οποίο θα μας επιστέψει δεδομένα καιρού για τις επόμενες 7 μέρες. Εμείς αποθηκεύουμε τα δεδομένα για τις επόμενες 24 ώρες αφού θα ανανεωθούν τα api calls στο solcast και θα έχουμε πιο ακριβείς δεδομένα για τις επόμενες μέρες. Αφού έχουμε τα δεδομένα καιρού και τα δεδομένα από τον χρήστη δημιουργούμε ένα πίνακα με μέγεθος ίσο με τον αριθμό φωτοβολταϊκών που δόθηκε και δημιουργούμε τα αντίστοιχα objects για το κάθε πάνελ από την κλάση solarPanel. Σε αυτή την κλάση έχουμε το μοντέλο παραγωγής ενέργειας των φωτοβολταϊκών που χρησιμοποιείται στο PVWatts.

Μόλις παραχθούν τα δεδομένα για κάθε panel συνδυάζουμε όλα τα δεδομένα σε ένα πίνακα έτσι ώστε να μπορούμε να τα επεξεργαστούμε. Επειδή παίρνουμε τα δεδομένα μας από το api ανά ώρα θα έχουμε τις ίδιες τιμές για μεγάλα διαστήματα στα δεδομένα μας. Για να είναι πιο ρεαλιστικό το αποτέλεσμα θα μικρύνουμε αυτό το διάστημα σε 1 λεπτό και θα αυξάνουμε ή θα μειώνουμε την τιμή του αποτελέσματος με βάση την τιμή της επομένης έτσι ώστε όταν κάνουμε request κάθε λεπτό να έχουμε διαφορετικό αποτέλεσμα όπως γίνεται σε ένα πραγματικό σύστημα φωτοβολταικών. Όπως θα δούμε στα αποτελέσματα φτάνουμε αρκετά κοντά στις πραγματικές τιμές που είχαμε από τα φωτοβολταϊκά του Πανεπιστημίου Κύπρου. Η προσομοίωση γίνεται για όλα τα δεδομένα που έχουμε διαθέσιμα έτσι ο χρήστης μπορεί να αιτηθεί την τιμή οποιασδήποτε χρονικής στιγμής αρκεί να υπάρχουν τα δεδομένα καιρού στην μνήμη. Τέλος η Main θα δημιουργήσει την απάντηση του Fronius για την χρονική στιγμή που ζητήθηκε και θα την επιστρέψει. Στη περίπτωση που γίνει η αίτηση GetSensorRealtimeData ακολουθείται ακριβώς η ίδια διαδικασία αφού σε κάθε εκτέλεση έχουμε αποθηκευμένα όλα τα δεδομένα που χρειάζονται απλά η Main θα καλέσει την μέθοδο GetSensorRealtimeData() η οποία θα δημιουργήσει την ανάλογη απάντηση και θα πάρει τα αντίστοιχα δεδομένα με αυτά που θα επέστρεφε το Fronius api.

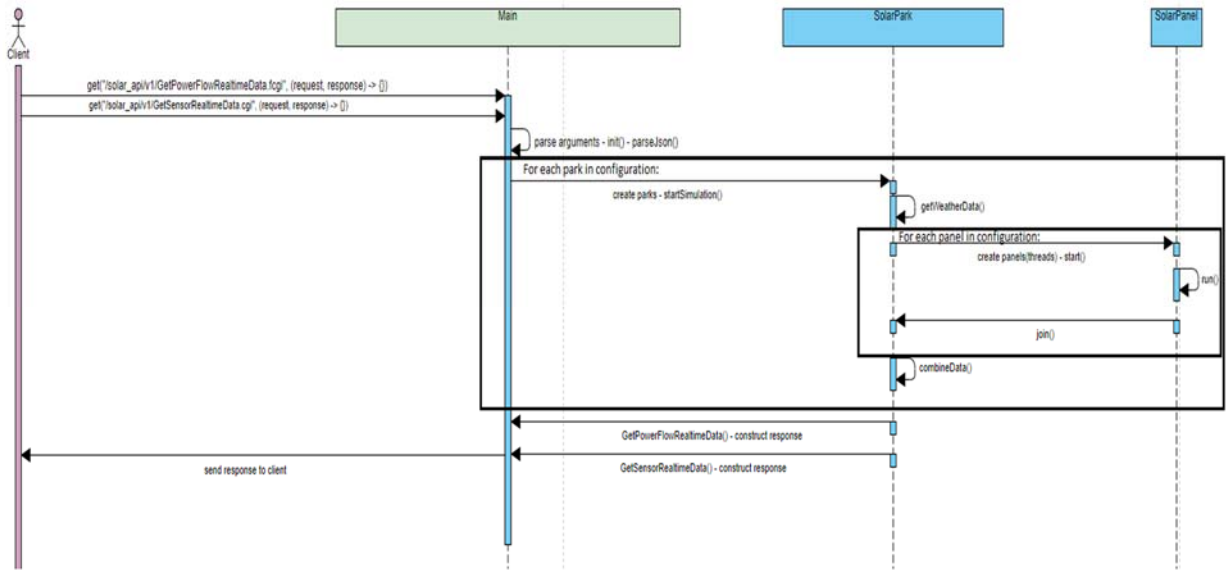


Figure 5, Sequence Diagram (Created using https://online.visual-paradigm.com/)

3.2 Αποτελέσματα Προσομοιωτή

Για να επαληθεύσουμε την αξιοπιστία του προσομοιωτή τρέξαμε τον προσομοιωτή σε ένα server και χρησιμοποιήσαμε ως configuration τα χαρακτηριστικά των φωτοβολταϊκών του Πανεπιστήμιου πάνω από το κτήριο ΘΕΕ-01/FST-01 τα οποία διακρίνονται σε 3 συστοιχίες(P114,P115,P232) με τα ακόλουθα χαρακτηριστικά.


```
{
	"parks": [
		{
			"panels": "63",
			"panels_size": "1.7",
			"panels_efficency": "17",
			"location": "aglantzia",
			"inclination": "17",
			"pr": "79.26",
			"capacity": "17.01",
			"type": "114"
		},
		{
			"panels": "34",
			"panels_size": "1.7",
			"panels_efficency": "17",
			"location": "aglantzia",
			"inclination": "17",
			"pr": "79.26",
			"capacity": "9.18",
			"type": "232"
		},
		{
			"panels": "56",
			"panels_size": "1.7",
			"panels_efficency": "17",
			"location": "aglantzia",
			"inclination": "17",
			"pr": "79.26",
			"capacity": "15.12",
			"type": "115"
		}
	]
}
```


Οι συγκεκριμένες συστοιχίες είναι συνδεδεμένες με ένα Fronius inverter και διαθέτουν διάφορους sensors έτσι μπορούμε να πάρουμε δεδομένα μέσω του υφιστάμενου συστήματος και του Fronius Api. Ο προσομοιωτής επιστρέφει την απάντηση στο ίδιο format με αυτό έτσι δημιουργήσαμε ένα script στο οποίο κάναμε ένα request στα δυο api κάθε 5 λεπτά σε περίοδο 14 ημερών και αποθηκεύαμε τα δεδομένα. Ακολουθούν γραφικές με τα αποτελέσματα που πήραμε από τις πραγματικές και τις simulated συστοιχίες φωτοβολταικών.

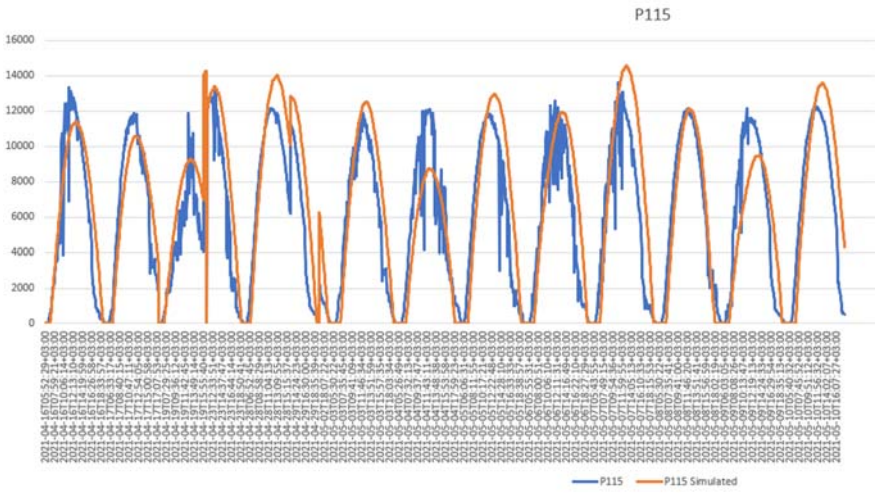
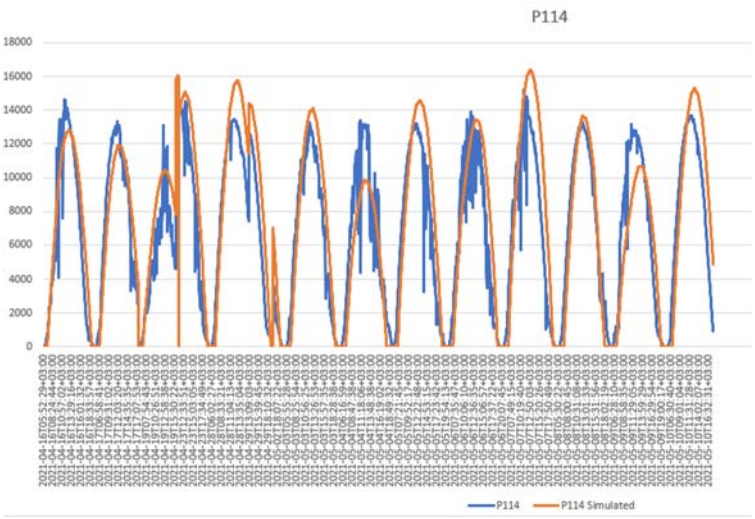


Figure 6, Chart comparing P115 Array located at UCY with the simulated P115 Array



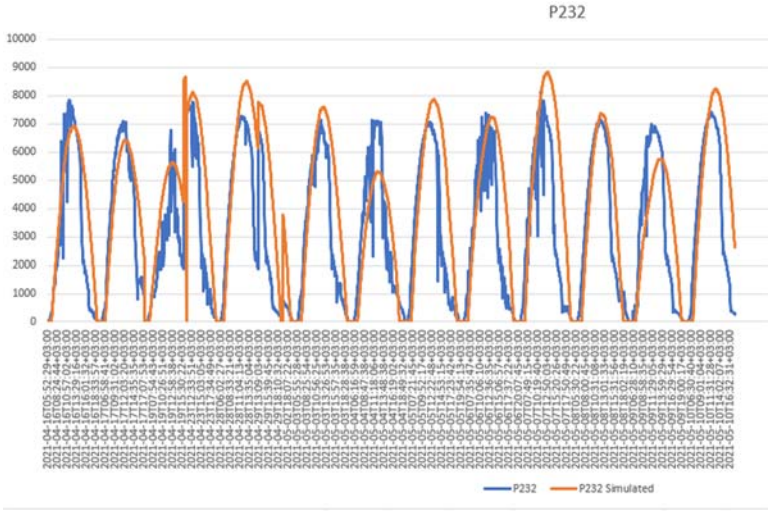


Figure 7, Charts comparing P114, P232 Arrays located at UCY with the simulated P114, P232 Arrays

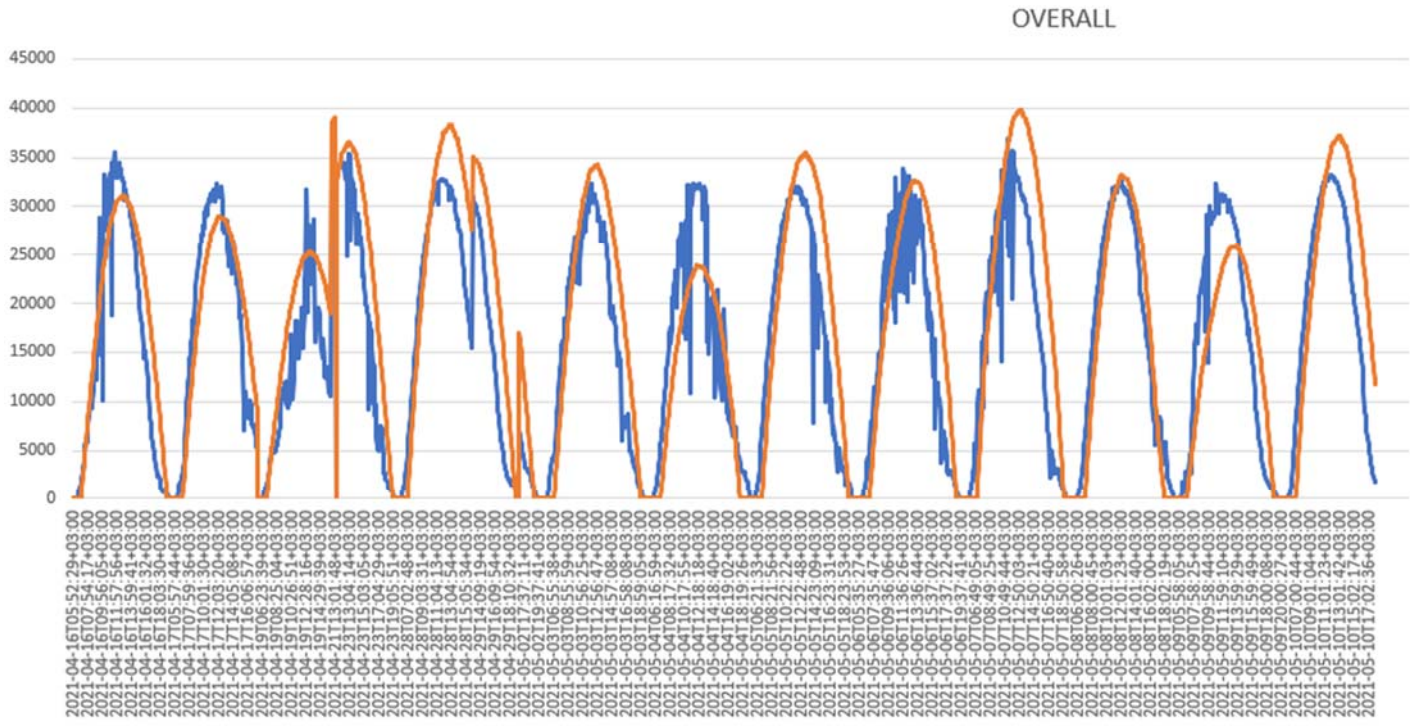


Figure 8, Chart comparing overall energy production of Arrays located at UCY with the simulated Arrays

Στον άξονα x έχουμε την χρονική σήμανση για κάθε αίτημά που έγινε ενώ στον άξονα y έχουμε την ενέργεια σε Watts. Όπως μπορούμε να δούμε τα αποτελέσματα της προσομοίωσης σε πολλές περιπτώσεις είναι αρκετά κοντά στα πραγματικά αποτελέσματα από τα φωτοβολταϊκά του Πανεπιστήμιου με μια μικρή απόκλιση ενώ υπάρχουν και περιπτώσεις οπού υπάρχει σημαντική απόκλιση στις τιμές. Αυτές οι αποκλίσεις μπορούν δικαιολογηθούν από την αδυναμία του api καιρού να μας δώσει ακριβή και συνεχόμενα δεδομένα αφού κάθε αίτημά στο api καιρού μας επιστρέφει μια τιμή για κάθε ώρα της μέρας έτσι δεν μπορούμε να γνωρίζουμε πως θα εξελιχθεί η παραγωγή ενέργειας μέσα σε εκείνο το χρονικό διάστημα. Υπάρχουν και περιπτώσεις στις οποίες βλέπουμε τις τιμές από το πραγματικό Fronius να ανεβοκατεβαίνουν σε μικρά χρονικά διαστήματα αλλά να ακολουθούν το trend των τιμών του simulated Fronius έτσι παρατηρούμε ακόμα μια αδυναμία του api καιρού στην οποία δεν έχουμε κάποια τιμή για την ηλιοφάνεια και την συννεφιά για κάθε στιγμή που γίνεται μια κλήση σε αυτό έτσι δεν μπορούμε να προβλέψουμε τέτοιου είδους φαινόμενα κατά την διάρκεια της μιας ώρας που δεν υπάρχουν δεδομένα. Σε μεταγενέστερο στάδιο θα μπορούσαμε να αλλάξουμε την πηγή των δεδομένων καιρού με ένα καλύτερο service έτσι ώστε να έχουμε ακόμα πιο ακριβής αποτελέσματα.

3.3 Integration with ENEDI

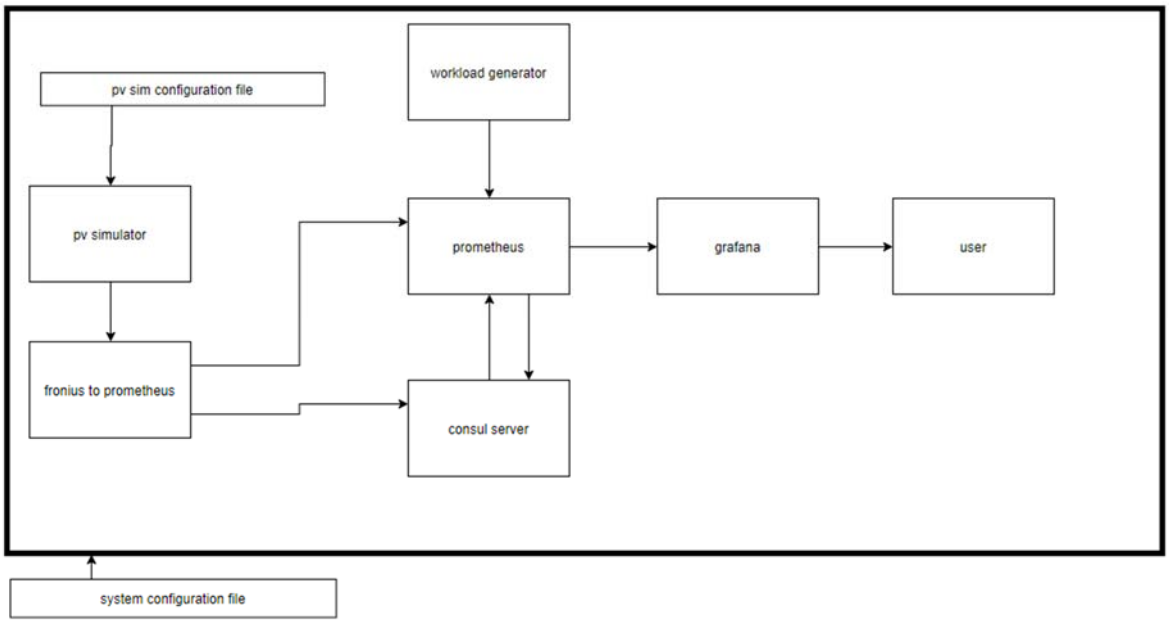


Figure 9, Diagram of all the components of the system

Αφού περιγράψαμε το πρόγραμμα που προσομοιώνει τα φωτοβολταϊκά μπορούμε να περάσουμε στην περιγραφή του συστήματος και το integration με το σύστημα ENEDI. Ο χρήστης θα πρέπει να δώσει στο σύστημα 2 configuration files. Το πρώτο είναι αυτό που θα περάσει στον προσομοιωτή φωτοβολταϊκών ενώ το δεύτερο είναι το αρχείο που θα περιγράφει το σύστημα και τα Services που θα χρησιμοποιηθούν για την προσομοίωση όπως ο προσομοιωτής φωτοβολταικών, το Prometheus, το Grafana, το consul και τέλος ένα πρόγραμμα που μετατρέπει την έξοδο του προσομοιωτή φωτοβολταικών στην γλώσσα του Prometheus το οποίο θα περιγράψουμε αμέσως μετά.

3.3.1 Prometheus

Το Prometheus είναι ένα εργαλείο που το χρησιμοποιούμε για να παρακολουθούμε και να απεικονίσουμε τα δεδομένα που παράγει ο προσομοιωτής, είναι λογισμικό ανοιχτού κώδικα και μπορούμε ευκολά να το προσαρμόσουμε στον προσομοιωτή αφού ήδη χρησιμοποιούνται για το υφιστάμενο σύστημα του Πανεπιστήμιου.

Επιπλέον συλλεγεί χιλιάδες μετρήσεις από containers και τα παρουσιάζει στον χρήστη. Μέσω αυτού του εργαλείου μπορεί να γίνει πιο εύκολα η αποσφαλμάτωση διαφόρων προβλημάτων επιβράδυνσης και ανωμαλιών στην υποδομή μας με σημαντικές απεικονίσεις και διάφορους συναγερμούς που προσφέρει. Παράλληλα αποθηκεύει όλα τα δείγματα τοπικά και τρέχει κανόνες για αυτά τα δεδομένα είτε για να συγκεντρώσει

και να καταγράψει νέες χρονοσειρές από υπάρχοντα δεδομένα ή να δημιουργήσει ειδοποιήσεις.

Το Prometheus λειτουργεί καλά για την καταγραφή καθαρών αριθμητικών χρονοσειρών. Σε έναν κόσμο μικροϋπηρεσιών, η υποστήριξή του για τη συλλογή πολυδιάστατων δεδομένων και την αναζήτηση είναι μια ιδιαίτερη δύναμη. Το Prometheus έχει σχεδιαστεί για αξιοπιστία, ώστε να είναι το σύστημα στο οποίο επισκέπτεστε κατά τη διάρκεια μιας διακοπής για να σας επιτρέψει να διαγνώσετε γρήγορα προβλήματα. Κάθε διακομιστής Prometheus είναι αυτόνομος, δεν εξαρτάται από την αποθήκευση δικτύου ή άλλες απομακρυσμένες υπηρεσίες. Επίσης απεικονίζει τα διάφορα δεδομένα που συλλεγεί μέσω του Grafana για καλύτερη κατανόηση από τους χρήστες.

Όλα τα services τρέχουν σε containers με την βοήθεια του docker compose. Το πρώτο container περιέχει τον προσομοιωτή φωτοβολταϊκών που περιγράφεται πιο πάνω. Σε ένα άλλο container τρέχει ένα script που υλοποιήσαμε με σκοπό την μετατροπή της απάντησης του προσομοιωτή φωτοβολταϊκών σε γλώσσα που καταλαβαίνει το Prometheus. Ο τρόπος με τον οποίο λειτουργεί είναι ο εξής, ο προσομοιωτής φωτοβολταϊκών τρέχει σαν διαφορετικό service από αυτό το script έτσι μπορεί οποιοδήποτε άλλο service να κάνει request και να πάρει δεδομένα. Για να καταλάβει το Prometheus την απάντηση του server το βάζουμε να παρακολουθεί το service που δημιουργήσαμε και όχι κατευθείαν τον προσομοιωτή. Όταν προσπαθήσει το Prometheus να κάνει scrape από το script τότε αυτό κάνει ένα request στον προσομοιωτή και μεταφράζει την απάντηση του, η οποία είναι σε μορφή json, σε μορφή που καταλαβαίνει το Prometheus. Το Prometheus τρέχει σε ένα άλλο container και έχει το δικό του configuration. Στο πείραμα μας θα ανεβάσουμε πολλούς servers σε διάφορες τοποθεσίες με διαφορετικά URLs έτσι παρουσιάζεται ένα πρόβλημα. Το Prometheus δεν γνωρίζει τις διευθύνσεις των υπολοίπων services και δεν μπορούμε να τις περάσουμε στατικά στο configuration του αφού αλλάζουν συνεχώς κάθε φορά που θα τρέχουμε τον προσομοιωτή. Για να λύσουμε αυτό το πρόβλημα χρησιμοποιήσαμε το consul το οποίο τρέχει σε ένα ξεχωριστό container και πρέπει να εγκατασταθεί σε όλα τα container που θέλουμε να αναγνωρίζονται αυτόματα από το Prometheus. Μετά την εγκατάσταση του και την σωστή διαμόρφωση του το Prometheus μπορεί αυτόματα να αναγνωρίζει νέα services που πρέπει να κάνει scrape έτσι οποιοδήποτε νέο service που τρέχει είναι άμεσα προσβάσιμο από το Prometheus. Στη συνεχεία έχουμε το container που τρέχει το Grafana το οποίο είναι μέρος του Prometheus αφού χρησιμοποιεί τα δεδομένα του και μας βοηθά στην απεικόνιση των αποτελεσμάτων που επιστρέφουν οι προσομοιωτές. Μέσω του Grafana φτιάχνουμε διάφορα διαγράμματα με

ενδιαφέροντα στοιχεία και μετρικές από τους διάφορους προσομοιωτές. Τέλος ενσωματώνουμε όλα τα services που αναφέρθηκαν στον προσομοιωτή Fogify και βλέπουμε τα αποτελέσματα από όλους τους προσομοιωτές με την βοήθεια του Prometheus και του Grafana και εξάγουμε τα συμπεράσματα μας. Το σημαντικό είναι ότι μπορούμε να προσθέσουμε και αλλά services στο σύστημα μας και να δούμε τα δεδομένα τους αφού οποιεσδήποτε καινούργιες υπηρεσίες αποφασίσουμε να προσθέσουμε θα αναγνωριστούν αυτόματα και θα ενσωματωθούν στο σύστημα μέσω του consul.

3.3.2 Fog computing & Fogify

Το Fog Computing αναδύεται ως το κυρίαρχο παράδειγμα που γεφυρώνει το χάσμα υπολογισμού και συνδεσιμότητας μεταξύ συσκευών ανίχνευσης και υπηρεσίες ευαίσθητες σε καθυστέρηση. Ωστόσο, ο πειραματισμός και η αξιολόγηση των υπηρεσιών IoT είναι μια δύσκολη εργασία που περιέχει την μη αυτόματη διαμόρφωση και ανάπτυξη ενός μίγματος γεωδιανεμημένων φυσικών και εικονικών υποδομών με διαφορετικές απαιτήσεις πόρων και δικτύου. Αυτό οδηγεί σε μη βέλτιστο, δαπανηρές και επιρρεπείς σε σφάλματα εφαρμογές λόγω πολλών απροσδόκητων σφαλμάτων που δεν είχαν αρχικά οραματιστεί κατά τη φάση του σχεδιασμού και υποτιμούσαν συνθήκες δοκιμής που δεν μοιάζουν με το τελικό περιβάλλον.

Το Fogify παρέχει ένα σύνολο εργαλείων για μοντελοποίηση σύνθετων τοπολογιών αποτελούμενες από ετερογενείς πόρους, δυνατότητες δικτύου, κριτήρια Quality of Service, ανάπτυξη διαμορφωμένων configuration και υπηρεσιών που χρησιμοποιούν containerized υποδομή και είναι υλοποιημένα στο cloud η σε local περιβάλλον και τέλος τον πειραματισμό, τις μετρήσεις και την αξιολόγηση της υλοποίησης με την παρουσίαση σφαλμάτων και προσαρμογή διάφορων configurations κατά το χρόνο εκτέλεσης για να δοκιμάσουμε διαφορετικά σενάρια που αποκαλύπτουν τους περιορισμούς μιας υπηρεσίας πριν από την εισαγωγή στο κοινό. Κατά την αξιολόγηση, υπηρεσίες IoT με πραγματικό φόρτο εργασίας εισάγονται για να δείξουν το εύρος δυνατότητα εφαρμογής και οφέλη της ταχείας δημιουργίας πρωτοτύπων μέσω του Fogify.

Στο πλαίσιο της διπλωματικής εργασίας έχουμε κάνει containerize τον προσομοιωτή που υλοποιήθηκε και στη συνέχεια χρησιμοποιήσαμε το Fogify έτσι ώστε να τρέξουμε τον προσομοιωτή με διάφορα configurations και τοπολογίες έτσι ώστε να δούμε πως

συμπεριφέρεται με την προσθήκη σφαλμάτων και να πάρουμε αποτελέσματα και συμπεράσματα.

3.3.3 Consul

Το consul είναι ένα service mesh solution το οποίο παρέχει λειτουργίες ανακάλυψης, διαμόρφωσης και κατάτμησης. Κάθε ένα από αυτά τα χαρακτηριστικά μπορεί να χρησιμοποιηθεί ξεχωριστά ανάλογα με τις ανάγκες, ή μπορεί να χρησιμοποιηθεί μαζί για την κατασκευή ενός πλήρους πλέγματος εξυπηρέτησης. Τα βασικά χαρακτηριστικά του Consul είναι η ανακάλυψη υπηρεσιών, ο έλεγχος υγείας των υπηρεσιών, η ασφαλής επικοινωνία υπηρεσιών και έχει υποστήριξη για πολλά datacenters. Το Consul έχει σχεδιαστεί για να είναι φιλικό τόσο για την κοινότητα των DevOps όσο και για τους προγραμματιστές εφαρμογών, καθιστώντας το ιδανικό για σύγχρονες, ελαστικές υποδομές.

Κάθε κόμβος που παρέχει υπηρεσίες στο Consul τρέχει έναν Consul agent. Η εκτέλεση ενός πράκτορα δεν απαιτείται για την ανακάλυψη άλλων υπηρεσιών ή τη λήψη/ρύθμιση δεδομένων κλειδιού/τιμής. Ο πράκτορας είναι υπεύθυνος για τον έλεγχο της υγείας των υπηρεσιών στον κόμβο καθώς και στον ίδιο τον κόμβο. Οι Consul agents μιλούν με έναν ή περισσότερους Consul servers. Στους Consul servers αποθηκεύονται και αναπαράγονται δεδομένα. Οι ίδιοι οι διακομιστές εκλέγουν έναν ηγέτη. Ενώ ο Consul μπορεί να λειτουργεί με έναν διακομιστή, συνιστάται από 3 έως 5 για να αποφευχθούν σενάρια αποτυχίας που οδηγούν σε απώλεια δεδομένων. Συνιστάται μια ομάδα διακομιστών Consul για κάθε κέντρο δεδομένων. Οι διακομιστές διατηρούν έναν κατάλογο, ο οποίος σχηματίζεται από τη συγκέντρωση πληροφοριών που υποβάλλονται από τους Consul agents. Ο κατάλογος διατηρεί την προβολή υψηλού επιπέδου του συμπλέγματος, συμπεριλαμβανομένων των διαθέσιμων υπηρεσιών, ποιοι κόμβοι εκτελούν αυτές τις υπηρεσίες, πληροφορίες υγείας και πολλά άλλα.
Στην δική μας περίπτωση χρησιμοποιήσαμε την υπηρεσία ανακάλυψης υπηρεσιών έτσι ώστε να μπορεί το Prometheus να βρίσκει τις διευθύνσεις των pv simulators που τρέχουν.

Όπως προαναφέρθηκε το σύστημα είναι υλοποιημένο σε διάφορα containers με την βοήθεια του docker. Για να δημιουργήσουμε τις διάφορες διεργασίες θα πρέπει να δημιουργήσουμε το Docker-Compose αρχείο του συστήματος. Μέσα σε αυτό το αρχείο δηλώνουμε τα διάφορα συστατικά του συστήματος. Η δομή του αρχείου έχει ως εξής:

```
services:
  prometheus:
    image: prom/prometheus:latest
    container_name: prometheus
    ports:
      - 9000:9090
    command: --web.enable-lifecycle  --config.file=/etc/prometheus/prometheus.yml
    volumes:
      - ./prometheus:/etc/prometheus
      - prometheus-data:/prometheus
  PVsim:
    build: .
    image: pvsim:0.0.1
    volumes:
      - /etc/timezone:/etc/timezone:ro
      - ./data:/code/src/main/resources:rw
      - ./configs/canada.conf:/code/configs/solarpark.conf:rw
    ports:
      - 4567:4567
  grafana:
    image: grafana/grafana:latest
    ports:
      - 3000:3000
    volumes:
      - grafana_data:/var/lib/grafana
      - ./grafana/provisioning/:/etc/grafana/provisioning/
    env_file:
      - ./grafana/config.monitoring
    restart: always
  fronius:
    build: ./fronius-to-prometheus
    image: python-fronius:0.0.1
    environment:
      FRONIUS_URL: "PVsim:4567"
      FRONIUS_SENSORS_PATH: "solar_api/v1/GetSensorRealtimeData.cgi?DataCollection=NowSensorData&Scope=System"
      FRONIUS_PV_PATH: "solar_api/v1/GetPowerFlowRealtimeData.fcgi"
      PREFIX: "cyprus__nicosia__ucy__dc1__fronius_"
      FRONIUS_TIMEOUT: 15
      FRONIUS_CONTAINER_NAME: "fronius"
      CONSUL_ENABLED: "true"
      CONSUL_SERVER: "consul-server"
    ports:
      - 19999:19999
    depends_on:
      - consul-server
  consul-server:
    image: hashicorp/consul:1.9.3
    container_name: consul-server
    restart: always
    volumes:
     - ./consul/server.json:/consul/config/server.json:ro
    ports:
      - "8500:8500"
      - "8600:8600/tcp"
      - "8600:8600/udp"
    command: "agent"
  stress:
    image: progrium/stress
    command: '--cpu 2 --io 1 --vm 2 --vm-bytes 128M -q'
```

Στο πεδίο volumes περνούμε στην κάθε διεργασία τα αρχεία που χρειάζεται για την εκτέλεση του, στο πεδίο ports δηλώνουμε σε ποια ports θα ακούει η κάθε διεργασία ενώ στο πεδίο environment και command δίνουμε τα arguments που χρειάζεται η κάθε διεργασία. Στο πεδίο build δίνουμε το path που βρίσκετε ο κώδικας του service που θέλουμε να γίνει build μετά από την έναρξη του προγράμματος ενώ το πεδίο image και container name περιέχει το όνομα του service που θα χρησιμοποιηθεί. Στο πεδίο volumes περνούμε τα διαφορά αρχεία του κάθε service από τον χώρο του συστήματος στον αποθηκευτικό χώρο του container έτσι ώστε να έχει πρόσβαση το κάθε service στα αρχεία που χρειάζεται.

Στο τμήμα του Prometheus βλέπουμε ότι ακούει στο port 9000 ενώ δίνουμε το argument --web.enable-lifecycle έτσι ώστε αν αλλάξει το configuration file θα φορτώνετε αυτόματα χωρίς να χρειαστεί να γίνει επανεκκίνηση του συστήματος. Ακόμα περνάμε το --config.file όπου είναι το path στο configuration του Prometheus μέσα στο container. Για να βρει το Prometheus αυτό το αρχείο θα πρέπει να το περάσουμε στο container από το πεδίο volume έτσι περνάμε το αρχείο με την χρήση του ./prometheus:/etc/prometheus όπου παίρνει από τα περιεχόμενα του φακέλου με όνομα Prometheus από τον χώρο του project και τα τοποθετεί μέσα στο container στο path /etc/prometheus. Στην συνέχεια περνάμε τον φάκελο Prometheus-data σε ένα άλλο φάκελο μέσα στο container με το όνομα Prometheus.

Παρομοίως έχουμε δηλώσει και το Grafana, το οποίο ακούει στο port 3000 και έχει σαν volumes τα δεδομένα που χρειάζεται στο grafana_data:/var/lib/grafana και ./grafana/provisioning/:/etc/grafana/provisioning/ ενώ περνάμε και κάποια environmental variables τα οποία βρίσκονται στο αρχείο που βρίσκετε στο ./grafana/config.monitoring.

Στη συνέχεια έχουμε το τμήμα του Fronius-to-Prometheus το οποίο ακούει στο port 19999 και είναι εξαρτώμενο από το container με όνομα consul-server ενώ έχουμε και κάποια environmental variables το οποίο θα δούμε στην συνέχεια. Στο τμήμα του consul-server δίνουμε την μεταβλητή restart: always έτσι ώστε αν παρουσιαστεί κάποιο σφάλμα κατά την εκτέλεση να γίνετε αυτόματα επανεκκίνηση του service ενώ δίνουμε και το configuration file στο volume ./consul/server.json:/consul/config/server.json:ro στο οποίο βάζουμε στο τέλος την λέξη ro η οποία δίνει μόνο δικαίωμα διαβάσματος του αρχείου στο container. Ακόμα δίνουμε την εντολή agent έτσι ώστε να σηκώσουμε ένα consul agent στα port 8500 και 8600.

Τέλος έχουμε το τμήμα του stress το οποίο είναι ο workload generator στο παράδειγμα μας και το τμήμα του Pvsim όπου είναι ο προσομοιωτής των φωτοβολταϊκών μας. Ο workload generator θα μπορούσε να είναι οποιοδήποτε πρόγραμμα για παράδειγμα θα μπορούσε να παράγει τις τιμές κατανάλωσης από ένα κέντρο δεδομένων. Για το stress πρέπει απλά να περάσουμε τα ακόλουθα parameters. Η παράμετρος --cpu είναι ο αριθμός των επεξεργαστών που θέλουμε να χρησιμοποιήσουμε, το --io είναι ο αριθμός των workers που θα κάνουν κάποιου είδους input και output operation, το --vm είναι ο αριθμός των workers που θα κάνουν εργασίες στην μνήμη και το --vm-bytes το οποίο είναι το μέγεθος των chunks μνήμης που θα γράφονται σε κάθε κύκλο από την χρήση του –vm. Τέλος βάζουμε την παράμετρο -q για να έχουμε ελάχιστο output στην κονσόλα μας.

Στη περίπτωση του Pvsim (Προσομοιωτής φωτοβολταϊκών), το οποίο ακούει στο port 4567, πρέπει να δώσουμε τα ακόλουθα volumes. Στο /etc/timezone:/etc/timezone:ro περνάμε την ζώνη ώρας στην οποία θέλουμε να ακολουθεί το πρόγραμμα, στο ./data:/code/src/main/resources:rw τα αρχεία που πρέπει να έχει το πρόγραμμα για να μπορεί να λειτουργήσει τα οποία περιέχονται ήδη στο project και δεν χρειάζονται αλλαγή και στο ./configs/canada.conf:/code/configs/solarpark.conf:rw το configuration file για την συστοιχία που θέλουμε να προσομοιώσουμε. Στο τέλος των αρχείων που χρειάζεται το πρόγραμμα αλλά και του configuration file προσθέτουμε την λέξη rw έτσι ώστε ότι αλλαγές κάνει το πρόγραμμα μέσα στο container να διατηρούνται και εκτός του container για τις επόμενες εκτελέσεις του αφού αποθηκεύει όλα τα δεδομένα που παίρνει έτσι ώστε να μην γίνονται αχρείαστα api calls και να υπάρχουν ιστορικά δεδομένα στην μνήμη.

Αφού είδαμε πως δηλώσαμε το κάθε service του συστήματος μπορούμε να δούμε πιο αναλυτικά το κάθε ένα από αυτά ξεκινώντας με το Fronius-to-Prometheus. Είναι γραμμένο σε python και ο σκοπός αυτού του service είναι να παίρνει την απάντηση από τον προσομοιωτή φωτοβολταϊκών που είναι σε μορφή json, να το μετατρέπει σε μορφή που καταλαβαίνει το Prometheus και να την επιστρέφει. Ακούει στο port 19999 και το URL /api/v1/allmetrics το οποίο είναι και το default endpoint που παρακολουθεί το Prometheus. Ακόμα είναι υπεύθυνο να δηλώσει την διεύθυνση του στον consul server με ένα POST request, που περιέχει ένα json, έτσι ώστε να είναι προσβάσιμο από το Prometheus. Το πρόγραμμα παίρνει τα ακόλουθα 8 arguments. Το FRONIUS_URL: "PVsim:4567" το οποίο περιέχει την διεύθυνση του Fronius Api, στην δική μας περίπτωση είναι η διεύθυνση του simulated Fronius (Pvsim), το FRONIUS_SENSORS_PATH:"solar_api/v1/GetSensorRealtimeData.cgi?DataCollection=NowSensorData&Scope=System" το οποίο είναι το path των sensors στο api του Fronius, το FRONIUS_PV_PATH: "solar_api/v1/GetPowerFlowRealtimeData.fcgi" που είναι το path των μετρικών στο api του Fronius, το PREFIX: "cyprus__nicosia__ucy__dc1__fronius_" το οποίο θα είναι το prefix των μετρικών του συγκεκριμένου service για αποθήκευση στο Prometheus, το FRONIUS_TIMEOUT: 15 ο οποίος είναι ο αριθμός τον δευτερολέπτων που θα περιμένουμε ανάμεσα από κάθε request, το FRONIUS_CONTAINER_NAME: "fronius" όπου είναι το όνομα του container, την μεταβλητή CONSUL_ENABLED: "true" για να δηλώσουμε ότι θέλουμε να κάνουμε register στο consul στην αρχή της εκτέλεσης και την μεταβλητή CONSUL_SERVER: "consul-server" όπου είναι το όνομα του consul server που θέλουμε να κάνουμε register.

Επόμενο service είναι ο consul server ο οποίος κρατά τα registered services και πιο συγκεκριμένα τα instances του Fronius-to-Prometheus. Είναι εγκατεστημένο στο port 8500 και έχει το ακόλουθο configuration:

```
{
   "node_name": "consul-server",
   "server": true,
   "bootstrap" : true,
   "enable_script_checks" : true,
   "ui_config": {
      "enabled" : true
   },
   "data_dir": "/consul/data",
   "addresses": {
      "http" : "0.0.0.0"
   },
   "connect": {
      "enabled" : true
   }
}
```

Σημαντικό είναι να κάνουμε true την μεταβλητή enable_script_checks για να μπορούμε να δηλώσουμε τα δικά μας services στο consul. Για την αρχικοποίηση του Prometheus, το οποίο ακούει στο port 9000, δίνουμε το ακόλουθο configuration:

global:

```
  scrape_interval: 15s
  scrape_timeout: 10s

rule_files:
  - alert.yml

scrape_configs:
  - job_name: consul-server
    consul_sd_configs:
      - server: consul-server:8500
    relabel_configs:
      - source_labels: [__meta_consul_tags]
        regex: .*,fronius,.*
        action: keep
    metrics_path: '/api/v1/allmetrics'
    params:
      format: [prometheus]
    honor_labels: true
    scrape_interval: 15s
```

Θέτουμε το scrape interval σε 15 δευτερόλεπτα δηλαδή το Prometheus θα παίρνει δεδομένα από τους στόχους του κάθε 15 δευτερόλεπτα ενώ θέτουμε το scrape timeout σε 10 δευτερόλεπτα έτσι ώστε αν δεν πάρει κάποια απάντηση από τους στόχους σε 10 δευτερόλεπτα μπορούμε να στείλουμε κάποιο alert στον χρήστη ότι το συγκεκριμένο service δεν λειτουργεί. Μπορούμε να δηλώσουμε διαφορά alerts στο αρχείο alert.yml μαζί με τα υφιστάμενα τα οποία ειδοποιούν τον χρήστη για την κατάσταση των services που παρακολουθεί το Prometheus. Στο σημείο scrape configs δηλώνουμε τα services που θέλουμε να παρακολουθεί το Prometheus. Στην περίπτωση μας έχουμε βάλει το Prometheus να παρακολουθεί τον consul server που αναφέραμε πιο πάνω ο οποίος περιέχει τις διευθύνσεις των διάφορων jobs που έγιναν register δυναμικά με path '/api/v1/allmetrics' ενώ παρακολουθεί τις μετρικές που περιέχουν στα label τους την λέξη Fronius.

Μαζί με το Prometheus εγκαταστήσαμε και το Grafana το οποίο συνεργάζεται με το Prometheus έτσι ώστε να έχουμε μια γραφική απεικόνιση των δεδομένων μέσω κάποιων διαγραμμάτων. Στο επόμενο μέρος θα δούμε το τελευταίο service πιο αναλυτικά το οποίο είναι ο προσομοιωτής του Fronius - φωτοβολταϊκών.

Ακολουθεί ένα διάγραμμα για το πως αλληλοεπιδρούν τα διάφορα services κατά την εκτέλεση του συστήματος.

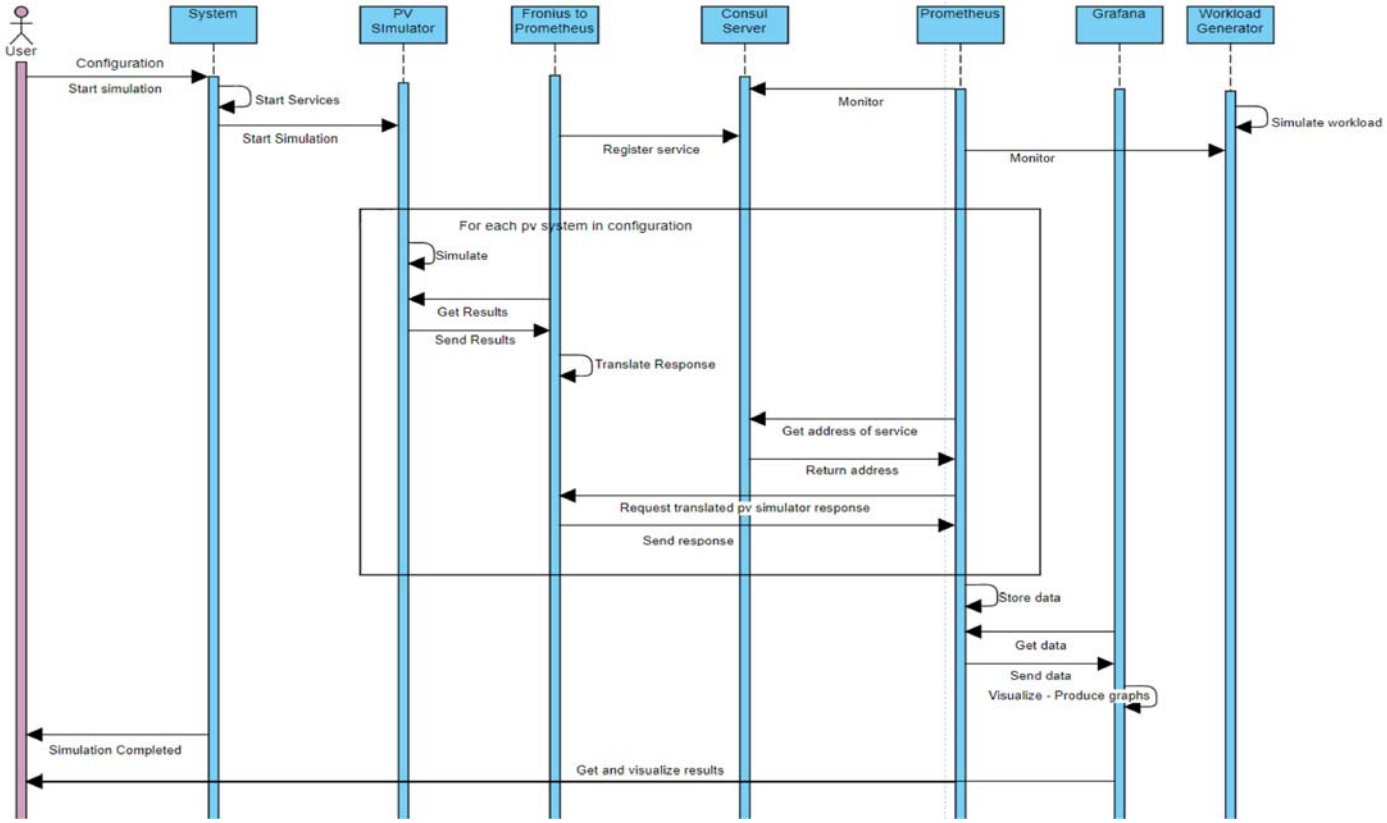


Figure 10, Sequence Diagram of how all the services communicate (Created using https://online.visual-paradigm.com/)

Κατά την εκκίνηση του συστήματος ο χρήστης δίνει το configuration file που περιγράφει το σύστημα που θέλει να προσομοιώσει και τις υπηρεσίες που θέλει να χρησιμοποιήσει όπως είδαμε πιο πάνω. Το σύστημα ξεκινά τις υπηρεσίες του συστήματος και τις αρχικοποιεί με το ανάλογο configuration τους και ξεκινά την προσομοίωση των φωτοβολταϊκών όπως περιγράφετε στο επόμενο κεφάλαιο. Μόλις τελειώσει η προσομοίωση η υπηρεσία Fronius-to-Prometheus ζητά τα δεδομένα από το simulated Fronius και τα μεταφράζει σε απάντηση που καταλαβαίνει το Prometheus. Ακόμα κάνει εγγραφή στον Consul Server έτσι ώστε να είναι ορατό από το Prometheus. Κατά την αρχικοποίηση του Prometheus το βάζουμε να παρακολουθεί τον Consul Server και τον workload generator που επιλέξαμε για το συγκεκριμένο σύστημα. Αυτό με την σειρά του κάνει ένα αίτημα στον Consul Server έτσι ώστε να πάρει τις διευθύνσεις τις οποίες βρίσκονται οι υπηρεσίες που θέλουμε να γίνουν scraped. Αυτό γίνετε κάθε ένα χρονικό διάστημα το οποίο έχει δηλωθεί στο configuration του κατά την αρχικοποίηση. Στην συνέχεια κάνει αιτήματα στις υπηρεσίες που του επέστρεψε το Consul και παίρνει δεδομένα από τις υπηρεσίες Fronius-to-Prometheus οι οποίες έχουν μεταφράσει τις

απαντήσεις από τα simulated Fronius ενώ παίρνει και διάφορες μετρικές από τον workload generator που έχουμε δηλώσει. Στην δική μας περίπτωση ο workload generator είναι ένα συνθετικό stress test αλλά θα μπορούσε να είναι όποια υπηρεσία θέλουμε να προστεθεί στο σύστημα. Στην συνέχεια αφού αποθηκεύσει τα δεδομένα δέχεται διάφορες αιτήσεις από το Grafana έτσι ώστε να μπορούμε να κάνουμε visualize τα αποτελέσματα μας. Το σύστημα συνεχίζει να παράγει δεδομένα με την ίδια διαδικασία και επιστρέφει στον χρήστη τις πληροφορίες που αιτείται μέσω του Prometheus και του Grafana.

3.4 Σύνδεση με το Fogify και παράδειγμα χρήσης

Για να επαληθεύσουμε την αξιοπιστία του συστήματος και ότι ο προσομοιωτής μαζί με την υπόλοιπη υποδομή δουλεύει, ενώσαμε το σύστημα με ένα Fog Computing Emulator, συγκεκριμένα το Fogify. Όλη η διαδικασία έγινε σε έναν εικονικό μηχάνημα πάνω στην υποδομή του εργαστηρίου, το οποίο είχε 16 εικονικούς πυρήνες και 16Gb μνήμη.

Το σενάριο το οποίο θέλαμε να επαληθεύσουμε είναι αν όντως κάποιος μπορεί να χρησιμοποιήσει το σύστημα του ΕΝΕΔΗ μαζί με το προσομοιωτή των φωτοβολταϊκών για εξακριβώσει την παραγωγή και κατανάλωση ενέργειας σε κέντρα δεδομένων που μπορεί να βρίσκονται ακόμα και σε διαφορετικές ηπείρους. Για το σκοπό αυτό διαλέξαμε τρεις διαφορετικές περιοχές σε τρεις διαφορετικές ηπείρους, συγκεκριμένα το Λονδίνο, το Τόκιο, και το Οχάιο. Για κάθε κέντρο δεδομένων δημιουργήσαμε ένα ξεχωριστό αρχείο με τα configurations των συστοιχιών των φωτοβολταϊκών. Φυσικά σε κάθε αρχείο ορίσαμε την εκάστοτε τοποθεσία καθώς και τον αριθμό των panels, τον αριθμό των συστοιχιών, την ποιότητα των panels, και γενικά οποιοδήποτε στοιχείο θα επηρέαζε τη προσομοίωση. Για παράδειγμα, η παρακάτω εικόνα δείχνει ένα μέρος του configuration του φωτοβολταϊκού πάρκου που βρίσκεται στο Οχάιο. Έχουμε επισημάνει τη παράμετρο location καθώς είναι η μόνη που αλλάζει και επηρεάζει την παραγωγή ενέργειας των φωτοβολταϊκών σε πραγματικό χρόνο, με βάση τις καιρικές συνθήκες εκείνης της χρονικής στιγμής.

Figure 11, Screenshot from the configuration of the pv simulator

Έχοντας λοιπόν δημιουργήσει τα αρχεία που χρειάζεται ο προσομοιωτής για να αναπαράξει τις τιμές για τις παραπάνω τοποθεσίες, τώρα πρέπει να δημιουργήσουμε το docker-compose

file που θα περιγράφει τα services που να γίνουν emulate από το Fogify. Για λόγους απλότητας, αποφασίσαμε να έχουμε ένα κεντρικοποιημένο Prometheus Server και ένα Consul τοποθετημένα και τα δύο στο κέντρο δεδομένων του Λονδίνου. Οπότε ξεκινώντας η προσομοίωση, τα Fronius από τις διαφορετικές γεωγραφικές τοποθεσίες γράφονται στο Consul και το Prometheus μπορεί να μαζεύει περιοδικά τις μετρήσεις από αυτά. Πρέπει να αναφέρουμε πως ο κάθε Fronius προσομοιωτής έχει και το δικό του container (φυσικά δημιουργήσαμε αυτά τα containers με ελάχιστες υπολογιστικές δυνατότητες). Το παραπάνω ήταν προϋπόθεση για να εκτελεστεί το πείραμα μέσα από το Fogify. Επίσης, πρέπει να αναφέρουμε ότι για κάθε μία από τις συστοιχίες των φωτοβολταϊκών υπάρχει ένα service του ΕΝΕΔΗ και μετατρέπει την δομή των δεδομένων του Fronius σε δομή ικανή να διαβαστεί από το σύστημα. Τελειώνοντας την περιγραφή των βασικών υπηρεσιών, έχοντας ως στόχο την επαλήθευση της υλοποίησης, δημιουργήσαμε φόρτο σε κάθε κέντρο δεδομένων με την χρήση containers που έτρεχαν την εντολή για cpu stress των linux (https://linux.die.net/man/1/stress).

Έχοντας δημιουργήσει το docker-compose file, το επόμενο βήμα είναι να προσθέσουμε την περιγραφή της υποδομής ακολουθώντας το μοντέλο του Fogify. Για αυτό το σκοπό, έπρεπε να εξετάσουμε τα χαρακτηριστικά του δικτύου μεταξύ των διαφορετικών κέντρων δεδομένων και να βρούμε τιμές για την καθυστέρηση του δικτύου. Η παρακάτω εικόνα δείχνει τις τιμές της καθυστέρησης μεταξύ των διαφόρων κέντρων δεδομένων της Amazon όπως μετρήθηκαν από την σελίδα https://www.cloudping.co/grid για τον Ιούνιο του 2021. Χρησιμοποιήσαμε τις τιμές μεταξύ των τοποθεσιών που έχουμε αναφέρει πιο πάνω (Λονδίνο, Οχάιο, Τόκιο) για να έχουμε πιο ρεαλιστικά αποτελέσματα. Η εικόνα 12 δείχνει ένα τμήμα του μοντέλου του Fogify για την προσομοίωση του δικτύου μεταξύ των κέντρων δεδομένων.

Destination Region / Source Region	Africa (Cape Town) af-south-1	Asia Pacific (Hong Kong) ap-east-1	Asia Pacific (Tokyo) ap-northeast-1	Asia Pacific (Seoul) ap-northeast-2	Asia Pacific (Mumbai) ap-south-1	Asia Pacific (Singapore) ap-southeast-1	Asia Pacific (Sydney) ap-southeast-2	Canada (Central) ca-central-1	EU (Frankfurt) eu-central-1	EU (Stockholm) eu-north-1	EU (Milan) eu-south-1	EU (Ireland) eu-west-1	EU (London) eu-west-2	EU (Paris) eu-west-3	Middle East (Bahrain) me-south-1	SA (São Paulo) sa-east-1	US East (N. Virginia) us-east-1	US East (Ohio) us-east-2	US West (N. California) us-west-1	US West (Oregon) us-west-2
Africa (Cape Town) af-south-1	9.14	248.87	363.2	387.3	174.48	206.81	427.13	240.42	160.12	182.53	190.99	157.18	147.77	157.29	154.33	336.84	220.38	229.89	287.4	277.31
Asia Pacific (Hong Kong) ap-east-1	248.11	4.59	53.67	40.21	97.39	42.61	144.69	198.37	199.41	224.85	194.65	223.09	216.45	205.57	115.6	314.67	195.13	183.44	160.89	143.27
Asia Pacific (Tokyo) ap-northeast-1	360.57	54.89	3.38	34.19	133.41	80.17	116.46	144.23	232.1	241.3	223.25	200.97	210.04	217.07	158.81	258.21	145.5	131.73	107.85	100.23
Asia Pacific (Seoul) ap-northeast-2	389.16	37.67	38.48	1.93	120.96	72.13	149.19	172.65	222.96	267.91	214.78	229.51	238.23	244.76	151.41	285.65	176.77	163.38	136.13	125.75
Asia Pacific (Mumbai) ap-south-1	165.7	98.47	133.16	121.84	3.3	60.56	154.32	199.9	134.76	158.88	125.96	121.94	114.01	106.29	39.35	301.22	185.89	199.63	236.9	235.33
Asia Pacific (Singapore) ap-southeast-1	208.76	40.12	81.85	73.46	59.87	1.47	94.89	219.73	157.08	180.83	149.15	180.38	173.06	164.23	80.77	334.54	228.87	210.71	178.06	170.49
Asia Pacific (Sydney) ap-southeast-2	415.58	139.63	114.83	146.52	152.4	95.9	3.57	198.16	286.56	295.49	240.28	256.72	265.52	278.91	179.25	312.54	199.78	189.08	139.06	141.58
Canada (Central) ca-central-1	243.69	194.35	144.4	172.85	199.63	220.71	198.3	4.51	99.41	121.25	113.13	72.61	88.19	96.08	218.55	146.03	25.44	33.28	81.49	66.37
EU (Frankfurt) eu-central-1	166.13	197.98	231.13	224.35	134.18	156.53	288.42	103.46	1.59	25.89	11.01	25.31	15.66	9.91	122.46	203.6	90.01	100.13	150.3	145.95
EU (Stockholm) eu-north-1	182.26	225.87	243.59	268.79	160.49	182.42	295.8	122.07	29.35	4.4	36.22	41.6	32.51	40.58	147.92	217.89	105.52	115.14	167.99	164.36
EU (Milan) eu-south-1	186.56	188.48	221.62	213.94	125.56	147.8	239.88	115.61	11.53	33.78	1.34	36.47	25.98	20.5	114.63	212.27	100.35	109.41	159.75	154.02
EU (Ireland) eu-west-1	163.23	222.1	200.85	232.27	122.73	181.3	257.04	70.57	26.63	42.2	36.2	2.5	12.17	18.54	142.55	178.89	70.4	86.33	138.42	126.35
EU (London) eu-west-2	149.12	210.16	209.85	238.28	112.74	171.53	266.67	85.83	16.42	34.55	26.41	12.6	1.55	9.92	132.41	188.14	76.97	86.68	138.74	133.01
EU (Paris) eu-west-3	156.81	204.73	216.45	244.54	106.5	164.46	280.63	96.95	10.21	38.22	19.12	18.92	9.59	1.47	125.22	197.14	82.58	92.77	143.46	140.64
Middle East (Bahrain) me-south-1	168.52	119.52	161.09	149.66	38.67	82.43	186.83	217.81	125.11	147.38	122.47	139.2	143.74	142.63	1.2	323.86	203.57	214.19	270.42	251.62
SA (São Paulo) sa-east-1	341.14	309.21	257.74	286.02	299.68	332.24	313.69	146.18	204.64	217.22	216.15	179.08	188.54	196.85	316.87	3.6	114.79	125.81	175.24	181.23
US East (N. Virginia)																				

Figure 12, Grid comparing ping between regions (https://www.cloudping.co/grid)

Με υλοποιημένη την εφαρμογή και το μοντέλο του Fogify, θα μπορούσαμε να δούμε αν όντως το σύστημά μας δουλεύει κανονικά. Ωστόσο, καθώς το Fogify υλοποιεί το emulation μέσω containers, δεν θα μπορούσαμε να έχουμε μετρήσεις επίδοσης από την υποδομή. Για να ξεπεράσουμε αυτό το πρόβλημα, χρειάστηκε να συνδέσουμε το monitoring agent του Fogify με το monitoring συστημα του ΕΝΕΔΗ, και συγκεκριμένα με το Prometheus. Το Fogify χρησιμοποιεί ως monitoring agent το cAdvisor(https://github.com/google/cadvisor) που έχει την δυνατότητα να μοντελοποιεί τα δεδομένα που έχει συλλέξει με τρόπο που το Prometheus μπορεί να τα αποθηκεύσει. Η διασύνδεση αυτών των συστημάτων χρειάστηκε: (1) να κάνουμε expose την πόρτα του cAdvisor API από το container του Fogify Agent ώστε να είναι προσβάσιμο στο Promitheus, και (2) να προσθέσουμε την IP του server πάνω στον οποίο έτρεχε το emulation ως "στόχο" στο Prometheus του ΕΝΕΔΗ. Μετά από τα configurations τελικά καταφέραμε να τρέξουμε το σύστημα και να πάρουμε δεδομένα και από την επεξεργαστική ισχύ και από την προσομοίωση των φωτοβολταϊκών.

Το επόμενο βήμα ήταν να δημιουργήσουμε κάποια dashboard που θα μπορούσαν να μας δώσουν πληροφορίες για την υποδομή σε πραγματικό χρόνο. Για το λόγο αυτό χρησιμοποιούμε το Graphana, ένα dashboard-as-a-service εργαλείο που χρησιμοποιείται από το ΕΝΕΔΗ για την αναπαράσταση και την απεικόνιση των αποθηκευμένων δεδομένων. Έτσι λοιπόν, δημιουργήσαμε ένα dashboard με όλα τα χρήσιμα δεδομένα που θα ήθελε να δει κάποιος διαχειριστής. Η παρακάτω εικόνα δείχνει το dashboard, που περιέχει δύο καρτέλες, την καρτέλα PVs, που έχει πληροφορίες για την παραγωγή ενέργειας των φωτοβολταϊκών, και την καρτέλα workload stats, που έχει πληροφορίες για το φόρτο εργασίας του κέντρου δεδομένων. Οι γραφικές της εικόνας παράγονται με βάση ερωτήματα τα οποία υποβάλλονται στο Promitheus περιοδικά από το Grafana και έχουν διατυπωθεί σε PromQL (Promitheus Query Language) κατα την σχεδίαση του dashboard.

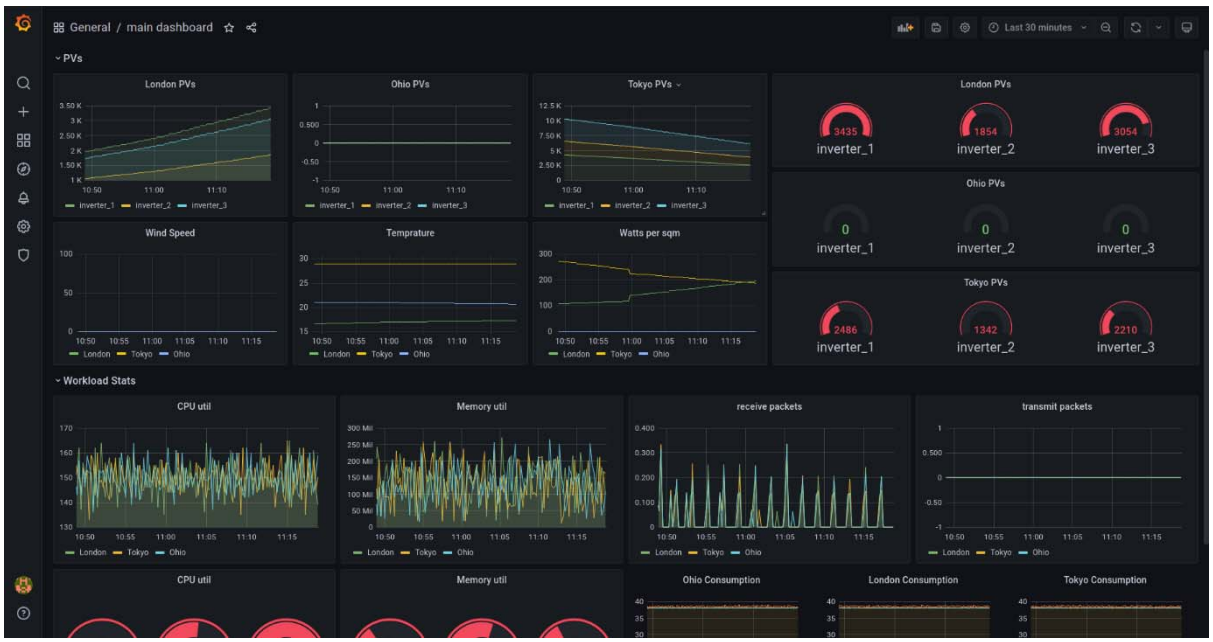


Figure 13, Grafana dashboard

Πιο συγκεκριμένα, στη παρακάτω εικόνα βλέπουμε την παραγωγή ενέργειας και τις τιμές από τους σένσορες από τις συστοιχίες των φωτοβολταϊκών της κάθε περιοχής. Συγκεκριμένα βλέπουμε να έχουμε μία ανοδική τάση στη παραγωγή ενέργειας στα φωτοβολταϊκά του Λονδίνου, μηδενική παραγωγή στα φωτοβολταϊκά του Οχάιο, και καθοδική τάση στα φωτοβολταϊκά του Τόκιο. Αυτό συμβαίνει λόγο της διαφοράς ώρας μεταξύ των περιοχών. Συγκεκριμένα, την στιγμή που πήραμε το στιγμιότυπο του dashboard, στο Λονδίνο ήταν πρωί, στο Οχάιο ήταν νύχτα, και στο Τόκυο ήταν απόγευμα. Δεξιότερα βλέπουμε την παραγωγή ανά inverter για κάθε μία περιοχή. Εδώ πρέπει να σημειώσουμε ότι είχαμε σε όλες τις περιοχές παρόμοια είσοδο ως configuration file, και έτσι σε όλες τις περιοχές η πρώτη συστοιχία φωτοβολταϊκών έχει τα περισσότερα panels. Έτσι λοιπόν βλέπουμε ότι στις περιοχές που έχουμε παραγωγή ενέργειας, από τον inverter 1 περνάνε περισσότερα kw, δεύτερη μεγαλύτερη συστοιχία είναι αυτή με το νούμερο 3, και η συστοιχία 2 έχει τα λιγότερα panels και συνεπώς την μικρότερη παραγωγή ενέργειας. Τα παραπάνω μας δείχνουν ότι όντως το σύστημά μας δουλεύει παράγει τιμές για τα PVs. Σχετικά με τις τιμές από τους σένσορες, και εδώ φαίνεται να παίρνουμε μετρήσεις με τη θερμοκρασία στο Τόκιο να είναι λίγο κάτω από τους 30 βαθμούς, στο Οχάιο κοντά στους 20, και στο Λονδίνο να έχει αυξητική τάση με λίγο πάνω από τους 15. Μετρήσεις παίρνουμε και για τα watts per sqm, όπως φαίνεται στην εικόνα. Το μόνο που δεν παρήγαγε μετρήσεις την στιγμή του snapshot ήταν η ταχύτητα του ανέμου. Ωστόσο, όπως έχουμε δει και από τους πραγματικούς σένσορες, οι μετρήσεις της ταχύτητας του ανέμου είναι σποραδικές με διάφορα “καρφιά” (spikes) σε τυχαίες χρονικές στιγμές. Παρατηρώντας για μεγαλύτερο χρονικό διάστημα το dashboard, είδαμε τα ίδια spikes και στη προσομοίωση.

Figure 14, Grafana dashboard

Όπως είπαμε παραπάνω, για να εξετάσουμε τις μετρήσεις από τα emulated μηχανήματα, δημιουργήσαμε τεχνητό φόρτο μέσω της εντολής stress. Στην παρακάτω εικόνα φαίνονται οι μετρήσεις από τα διαφορετικά κέντρα δεδομένων σε πραγματικό χρόνο. Συγκεκριμένα, η πρώτη γραφική δείχνει το cpu utilization από κάθε κέντρο δεδομένων και η δεύτερη το memory utilization. Καθώς χρησιμοποιήσω την ίδια εντολή και όλα τα κέντρα δεδομένων έχουν τους ίδιους πόρους, βλέπουμε ότι οι γραφικές για τα διαφορετικά κέντρα δεδομένων είναι παρόμοιες. Μετά (αριστερότερα) παραθέτουμε την μεταφορά δεδομένων πάνω από το δίκτυο. Καθώς ο φόρτος που προσθέσαμε επηρεάζει μόνο τον επεξεργαστή και την μνήμη, βλέπουμε ότι έχουμε μηδενικά δεδομένα που έφυγαν από τους emulated κόμβους. Στα δεδομένα που δέχονται οι κόμβοι βλέπουμε κάποια "καρφιά" (spikes) που σε πρώτο επίπεδο δεν θα μπορούσαν να εξηγηθούν. Ωστόσο, αυτά τα πακέτα είναι μηδαμινά και προέρχονται από heartbeats του ίδιου του docker και του docker-swarm. Στη δεύτερη γραμμή του dashboard, βλέπουμε τις τιμές (σε πραγματικό χρόνο) χρήσης του επεξεργαστή και της μνήμης σε άλλη γραφική αναπαράσταση. Τέλος παραθέτουμε την κατανάλωση ενέργειας για τα διαφορετικά κέντρα δεδομένων. Συγκεκριμένα, καθώς για τις ανάγκες του πειράματος είχαμε μόνο έναν κόμβο σε κάθε κέντρο δεδομένων, δημιουργήσαμε την κατανάλωση ενέργειας βάση κάποιων γραμμικών εξισώσεων που υπάρχουν στο paper *Full-System Power Analysis and Modeling for Server Environments [5]*. Σε αυτή την έρευνα, οι συγγραφείς καταλήγουν σε δύο τύπους για δύο διαφορετικούς servers, διαλέξαμε τον τύπο ενεργειακής κατανάλωσης για τον server blade που είναι ο ακόλουθος:

$$P\_blade = 14.45 + 0.236 * cpu\_util - (4.47E\text{-}8) * mem\_util + 0.00281 * disk\_util + (3.1E\text{-}8) * net\_util$$

Μην μένοντας στον απλό υπολογισμό της κατανάλωσης, αποφασίσαμε να δημιουργήσουμε γραφικές που δείχνουν ξεχωριστά πως επηρεάζει κάθε μέτρηση ξεχωριστά την κατανάλωση του server. Όπως βλέπουμε στην παρακάτω εικόνα, το σημαντικότερο στοιχείο για το φόρτο εργασίας που έχουμε στο πείραμα είναι το CPU utilization, που είναι λογικό αν σκεφτεί κανείς

ότι στρεσάρεται μόνο η CPU. Ακολουθεί η σταθερά του τύπου και μετά η κατανάλωση της μνήμης. Όπως είπαμε και παραπάνω δεν έχουμε σχεδόν καθόλου μεταφορά δεδομένων μέσω δικτύου, και για αυτό δεν έχουμε και καθόλου κατανάλωση ενέργειας.

Figure 15, Grafana dashboard

3.5 Μελλοντικά σχέδια

Όπως είδαμε το σύστημα που υλοποιήσαμε έχει θετικά αλλά και αρνητικά στοιχεία αλλά είναι αρκετά προσαρμόσιμο έτσι μπορούμε εύκολα να το επεκτείνουμε και να το βελτιώσουμε. Στο μέλλον θα μπορούσαμε να κάνουμε τον προσομοιωτή πιο αποτελεσματικό και ρεαλιστικό με κάποιο άλλο api καιρού για να ξεπεράσουμε κάποιους περιορισμούς που μας επέφερε η επιλογή του, κυρίως επειδή η χρήση του είναι δωρεάν. Ακόμα θα μπορούσαμε να προσθέσουμε και άλλα components στο σύστημα ενώ θα μπορούσε να γίνει και μια διπροσωπία όπου ο χρήστης θα μπορούσε πιο εύκολα να αλλάζει τα διάφορα inputs και configurations του συστήματος για πιο εύκολη και γρήγορη εκτέλεση.

Στα πλαίσια αυτής την διπλωματικής εργασίας αντιμετώπισα αρκετές δυσκολίες λόγω αρκετών τεχνολογιών που χρησιμοποιήθηκαν οι οποίες ήταν καινούργιες σε εμένα και χρειάστηκε χρόνος για την κατανόηση και την υλοποίηση τους. Πιστεύω έχω μάθει αρκετά σε αυτό το διάστημα μέσω της επίλυσης αυτών των προβλημάτων και της δουλείας που έγινε και ελπίζω να έχει βγει ένα καλό αποτέλεσμα μετά από αυτή την προσπάθεια.

Βιβλιογραφία

1. Agarwal, Sachin. “Public Cloud Inter-Region Network Latency as Heat-Maps.” *Medium*, Medium, 23 Oct. 2019, medium.com/@sachinkagarwal/public-cloud-inter-region-network-latency-as-heat-maps-134e22a5ff19.
2. Aron P. Dobos. *PVWatts Version 5 Manual*. NREL, 2014.
3. “AWS Inter Region Latency¶.” *AWS Inter Region Latency - aviatrix_docs Documentation*, docs.aviatrix.com/HowTos/inter_region_latency.html.
4. “Consul Tutorials.” *HashiCorp Learn*, learn.hashicorp.com/consul.
5. Dimitris Economou, Suzanne Rivoire, Christos Kozyrakis. *Full-System Power Analysis and Modeling for Server Environments*. Department of Electrical Engineering, Stanford University, www-mount.ece.umn.edu/~jjyi/MoBS/2006/program/3A-Economou.pdf.
6. “Docker and System Monitoring Dashboard for GRAFANA.” *Grafana Labs*, grafana.com/grafana/dashboards/893.
7. “Docker Documentation.” *Docker Documentation*, 27 Aug. 2021, docs.docker.com/.
8. “Documentation.” *Documentation - Spark Framework: An Expressive Web Framework for Kotlin and Java*, sparkjava.com/documentation.
9. Athanasios Tryfonos, Andreas Andreou, Nicholas Loulloudes George Pallis, Marios D. Dikaiakos. *ENEDI:* ENEDI: Energy Saving in Datacenters. Department of Computer Science, University of Cyprus, 2019
10. “Fogify: A Fog COMPUTING Emulation Framework.” *Fogify: A Fog Computing Emulation Framework | Fogify: A Fog Computing Emulator Framework*, ucy-linc-lab.github.io/fogify/.
11. *Fronius Solar API V1*. 1.0 ed., Fronius, 2017.
12. “Get Started with Docker Compose.” *Docker Documentation*, 27 Aug. 2021, docs.docker.com/compose/gettingstarted/.

13. GiantSwarm, Matthias Lübken from. “Matthias Lübken From Giantswarm.” *Spark Framework: An Expressive Web Framework for Kotlin and Java*, 14 Apr. 2015, sparkjava.com/tutorials/docker.
14. Google. “Google/Cadvisor: Analyzes Resource Usage and Performance Characteristics of Running Containers.” *GitHub*, github.com/google/cadvisor.
15. “Grafana Documentation.” *Grafana Labs*, grafana.com/docs/grafana/latest/.
16. “Learn Netdata.” *Learn Netdata Blog RSS*, learn.netdata.cloud/.
17. Moysis Symeonides , Zacharias Georgiou , Demetris Trihinas† , George Pallis , Marios D. Dikaiakos. *Fogify: A Fog Computing Emulation Framework*. Department of Computer Science, University of Cyprus, 2019.
18. “Overview of Docker Compose.” *Docker Documentation*, 27 Aug. 2021, docs.docker.com/compose/.
19. Prometheus. “Overview: Prometheus.” *Prometheus Blog*, prometheus.io/docs/introduction/overview/.
20. “Prometheus.” *Grafana Labs*, grafana.com/docs/grafana/latest/datasources/prometheus/.
21. “Pvwatts.” *PVWatts Calculator*, pvwatts.nrel.gov/pvwatts.php.
22. *Regions and Zones*, docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-regions-availability-zones.html.
23. “Solar Forecasting & Solar Irradiance Data.” *Solcast - Api Documentation*, solcast.com/solar-data-api/api/.
24. “Solar Forecasting & Solar Irradiance Data.” *Solcast*, solcast.com/solar-data-api/api-toolkit/.
25. “Stress(1) - Linux Man Page.” *Stress(1): Impose Load on/Stress Test Systems - Linux Man Page*, linux.die.net/man/1/stress.

26. Yegulalp, Serdar. “Docker Tutorial: Get Started with Docker Compose.” *InfoWorld*, InfoWorld, 21 Feb. 2018, www.infoworld.com/article/3254689/docker-tutorial-get-started-with-docker-compose.html.
27. Δρ Γεωργίου, Γιώργος Η. *ΑΝΑΛΥΣΗ ΑΠΑΙΤΗΣΕΩΝ ΣΕ ΤΕΧΝΙΚΟ ΚΑΙ ΝΟΜΙΚΟ ΑΔΕΙΟΔΟΤΙΚΟ ΕΠΙΠΕΔΟ*. ΠΑΡΑΔΟΤΕΟ 3.3.2 ΑΝΑΛΥΣΗ ed., Πανεπιστήμιο Κύπρου, 2018.
28. Δρ. Νικόλας Λουλούδης. *ΑΝΑΦΟΡΑ ΛΕΙΤΟΥΡΓΙΑΣ ΣΥΣΤΗΜΑΤΟΣ ΕΞΥΠΝΗΣ ΔΙΑΧΕΙΡΙΣΗΣ ΦΟΡΤΙΩΝ ΚΕΝΤΡΩΝ ΔΕΔΟΜΕΝΩΝ*. ΠΑΡΑΔΟΤΕΟ 3.3.3γ ed., ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ, 2019.
29. Καθηγητής Γιώργος Η. Γεωργίου. *ΜΕΛΕΤΗ ΣΥΜΒΟΛΗΣ ΠΡΟΓΡΑΜΜΑΤΟΣ ΣΤΗΝ ΕΞΟΙΚΟΝΟΜΗΣΗ ΕΝΕΡΓΕΙΑΣ, ΌΦΕΛΟΣ ΑΝΑ ΊΔΡΥΜΑ*. ΠΑΡΑΔΟΤΕΟ 6.3.2 ed., Πανεπιστήμιο Κύπρου, 2020.
30. “‘Consul Documentation.” *Consul by HashiCorp*, www.consul.io/docs.
31. “‘Http Api Structure.” *Consul by HashiCorp*, www.consul.io/api-docs.

Παράρτημα

GitHub repository: https://github.com/tioaki02/ADE

froniusRequests.py (script that made requests to both the real Fronius and the simulated one so we can compare the results)

```
import requests
import json
import time

while True:
    error = False
    s1 = ""
    s2 = ""
    try:
        r = requests.get('http://localhost:4567/solar_api/v1/GetPowerFlowRealtimeData.fcgi')
        #print(r.status_code)
        if r.status_code != 200:
            raise Exception("status code error")
        tmp = r.text
        res = json.loads(tmp)
        #print(json.dumps(res, indent=4, sort_keys=True))
        s1 = s1 + str(res['Head']['Timestamp']) + ","
        s1 = s1 + str(res['Body']['Data']['Inverters']['1']['E_Day']) + ","
        s1 = s1 + str(res['Body']['Data']['Inverters']['1']['E_Total']) + ","
        s1 = s1 + str(res['Body']['Data']['Inverters']['1']['E_Year']) + ","
        s1 = s1 + str(res['Body']['Data']['Inverters']['1']['P']) + ","
        s1 = s1 + str(res['Body']['Data']['Inverters']['3']['E_Day']) + ","
        s1 = s1 + str(res['Body']['Data']['Inverters']['3']['E_Total']) + ","
        s1 = s1 + str(res['Body']['Data']['Inverters']['3']['E_Year']) + ","
        s1 = s1 + str(res['Body']['Data']['Inverters']['3']['P']) + ","
        s1 = s1 + str(res['Body']['Data']['Inverters']['2']['E_Day']) + ","
        s1 = s1 + str(res['Body']['Data']['Inverters']['2']['E_Total']) + ","
        s1 = s1 + str(res['Body']['Data']['Inverters']['2']['E_Year']) + ","
        s1 = s1 + str(res['Body']['Data']['Inverters']['2']['P']) + ","
        s1 = s1 + str(res['Body']['Data']['Site']['E_Day']) + ","
        s1 = s1 + str(res['Body']['Data']['Site']['E_Total']) + ","
        s1 = s1 + str(res['Body']['Data']['Site']['E_Year']) + ","
        s1 = s1 + str(res['Body']['Data']['Site']['P_PV'])+"\n"
        print("success s1")
    except:
        error = True
    try:
        r = requests.get('http://10.16.24.137/solar_api/v1/GetPowerFlowRealtimeData.fcgi')
        #print(r.status_code)
        if r.status_code != 200:
            raise Exception("status code error")
        tmp = r.text
        res = json.loads(tmp)
        #print(json.dumps(res, indent=4, sort_keys=True))
        s2 = s2 + str(res['Head']['Timestamp']) + ","
        s2 = s2 + str(res['Body']['Data']['Inverters']['1']['E_Day']) + ","
        s2 = s2 + str(res['Body']['Data']['Inverters']['1']['E_Total']) + ","
        s2 = s2 + str(res['Body']['Data']['Inverters']['1']['E_Year']) + ","
        s2 = s2 + str(res['Body']['Data']['Inverters']['1']['P']) + ","
        s2 = s2 + str(res['Body']['Data']['Inverters']['2']['E_Day']) + ","
        s2 = s2 + str(res['Body']['Data']['Inverters']['2']['E_Total']) + ","
        s2 = s2 + str(res['Body']['Data']['Inverters']['2']['E_Year']) + ","
        s2 = s2 + str(res['Body']['Data']['Inverters']['2']['P']) + ","
        s2 = s2 + str(res['Body']['Data']['Inverters']['3']['E_Day']) + ","
```

```
        s2 = s2 + str(res['Body']['Data']['Inverters']['3']['E_Total']) + ","
        s2 = s2 + str(res['Body']['Data']['Inverters']['3']['E_Year']) + ","
        s2 = s2 + str(res['Body']['Data']['Inverters']['3']['P']) + ","
        s2 = s2 + str(res['Body']['Data']['Site']['E_Day']) + ","
        s2 = s2 + str(res['Body']['Data']['Site']['E_Total']) + ","
        s2 = s2 + str(res['Body']['Data']['Site']['E_Year']) + ","
        s2 = s2 + str(res['Body']['Data']['Site']['P_PV']) + "\n"
        print("success s2")
    except:
        error = True

    if error:
        print("Not reachable...")
        time.sleep(300)
        print("Restarting...")
    else:
        print("Writing to files...")
        f = open("solarparkemulator.csv", "a+")
        f.write(s1)
        f.close()
        f = open("fronius.csv", "a+")
        f.write(s2)
        f.close()
        time.sleep(300)
        print("Restarting...")
```

Main.java (Main class of pv simulator project)

```
package com.solarparkemulator.app;

import static spark.Spark.*;

import java.io.FileWriter;
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.sql.Timestamp;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.time.Instant;
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.Date;
import java.util.TimeZone;
import java.util.stream.Stream;

import org.apache.log4j.BasicConfigurator;
import org.json.*;

public class Main {
	// method to read entire file contents and return it in a string
	public static String readFile(String file) throws Exception {
		StringBuilder contentBuilder = new StringBuilder();
		Stream<String> stream;
		if (file.startsWith("configs")) {
			stream = Files.lines(Paths.get(file), StandardCharsets.UTF_8);
		} else {
			stream = Files.lines(Paths.get("src/main/resources/" + file), StandardCharsets.UTF_8);

		}
		stream.forEach(s -> contentBuilder.append(s).append("\n"));
		stream.close();
		return contentBuilder.toString();
	}

	// write string to file - empties file if exists
	public static void writeToFile(String content, String file) throws Exception {
		FileWriter myWriter = new FileWriter(file);
		myWriter.write(content);
```

```
		myWriter.close();
	}

	// parse json file and return an arraylist containing all solar parks specified
	// in the configuration file
	private static ArrayList<solarPark> parseJson(String conf) throws JSONException, ParseException {
		// parse
		JSONArray obj = new JSONObject(conf).getJSONArray("parks");
		// init arraylist
		ArrayList<solarPark> ret = new ArrayList<solarPark>();
		// iterate parks in configuration file and set variables of each park
		for (int i = 0; i < obj.length(); i++) {
			JSONObject obj2 = (JSONObject) obj.get(i);
			// create solarpark object
			solarPark park = new solarPark(i);
			park.setPanels(Integer.parseInt(obj2.getString("panels")));
			park.setPanelsAge(Integer.parseInt(obj2.getString("panels_age")));
			park.setPanelsSize(Double.parseDouble(obj2.getString("panels_size")));
			park.setPanelsEfficency(Integer.parseInt(obj2.getString("panels_efficency")));
			park.setPanelsType(Integer.parseInt(obj2.getString("panel_type")));
			park.setSystemlosses(Integer.parseInt(obj2.getString("system_losses")));
			park.setLocation(obj2.getString("location").replace(" ", "+").toLowerCase());
			park.setInclination(Integer.parseInt(obj2.getString("inclination")));
			park.setInstallationType(Integer.parseInt(obj2.getString("installation_type")));
			park.setAzimuth(Integer.parseInt(obj2.getString("azimuth")));
			park.setElectricityRateType(obj2.getString("electricity_rate_type"));
			park.setElectricityRate(Double.parseDouble(obj2.getString("electricity_rate")));
			park.setType(obj2.getString("type"));
			park.setCapacity(Double.parseDouble(obj2.getString("capacity")));
			park.setPr(Double.parseDouble(obj2.getString("pr")));
			ret.add(park);
		}
		return ret;
	}

	// construct GetPowerFlowRealtimeData response
	public static String GetPowerFlowRealtimeData(ArrayList<solarPark> parks, String arg1, String arg2)
			throws Exception {
		// construct response
		JSONObject ar = new JSONObject();
		// construct head
		JSONObject head = new JSONObject();
		head.put("RequestArguments", new JSONObject());
		head.put("Status", new JSONObject().put("Code", "0").put("Reason", "").put("UserMessage", ""));
		DateTimeFormatter dtf = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");
		ar.put("Head", head);
		// construct body
		JSONObject body = new JSONObject();
		JSONObject data = new JSONObject().put("Version", "12");
		JSONObject site = new JSONObject();
		double edaytotal = 0.0;
		double eyeartotal = 0.0;
		double etotaltotal = 0.0;
		double ptotal = 0.0;
		site.put("Meter_Location", "unknown");
		site.put("Mode", "produce-only");
		site.put("P_Akku", "null");
		site.put("P_Grid", "null");
		site.put("P_Load", "null");
		site.put("rel_Autonomy", "null");
		site.put("rel_SelfConsumption", "null");
		data.put("Site", site);
		JSONObject inverters = new JSONObject();
		for (int i = 0; i < parks.size(); i++) {
			solarPark a = parks.get(i);

			double e_day = 0.0;
			double e_year = 0.0;
			double e_total = 0.0;
			double p = 0.0;

			long tnow = System.currentTimeMillis() / 1000L;
			if (arg1 != null && arg2 != null) {
				return "only 1 argument allowed";
			}
			if (arg1 != null) {
				try {
					SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd
hh:mm:ss");
					Date date = new Date();
```

```
                                        date = dateFormat.parse(arg1);
                                        tnow = (long) date.getTime() / 1000;
                                } catch (Exception e) {
                                        return "cant parse timestamp";
                                }
                        }
                        if (arg2 != null) {
                                try {
                                        tnow = Long.parseLong(arg2);
                                } catch (Exception e) {
                                        return "cant parse unixtimestamp";
                                }
                        }

                        LocalDateTime now = LocalDateTime.ofInstant(Instant.ofEpochSecond(tnow),
TimeZone.getDefault().toZoneId());
                        head.put("Timestamp", dtf.format(now).replace(" ", "T") + "+03:00");

                        long closest = Math.abs(a.getFinalt().get(0) - tnow);
                        int index = 0;
                        for (int k = 1; k < a.getFinalt().size(); k++) {
                                long dt = a.getFinalt().get(k);
                                if (Math.abs(dt - tnow) < closest) {
                                        closest = Math.abs(dt - tnow);
                                        index = k;
                                }
                        }
                        p = p + a.getFinalp().get(index);
                        if (closest > 120)
                                head.put("Status",
                                                new JSONObject().put("Code", "-1").put("Reason",
                                                                "No historical data added. Please wait to get new
data (Max waiting time 1hour)")
                                                                .put("UserMessage", "Try again in a few moments
or provide a valid timestamp"));
                        if (p < 1 || closest > 120)
                                p = 0;
                        SimpleDateFormat formatter = new SimpleDateFormat("yyyy-MM-dd");
                        Date date = new Date();
                        LocalDate localDate = LocalDate.parse(formatter.format(date));
                        LocalDateTime startOfDay = localDate.atStartOfDay();
                        String time = startOfDay.toString().replace('T', ' ').replace("00:00", "00:01");
                        SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd hh:mm");
                        date = dateFormat.parse(time);
                        long unixTime = (long) date.getTime() / 1000;
                        Timestamp timestamp = new Timestamp(System.currentTimeMillis());
                        tnow = timestamp.getTime() / 1000;

                        for (int j = 0; j < a.getFinalt().size(); j++) {
                                long t = a.getFinalt().get(j);
                                if (t <= tnow && t >= unixTime)
                                        e_day = e_day + a.getFinalp().get(j);
                        }

                        Calendar cal = Calendar.getInstance();
                        cal.add(Calendar.YEAR, -1); // to get previous year add -1
                        date = cal.getTime();
                        localDate = LocalDate.parse(formatter.format(date));
                        startOfDay = localDate.atStartOfDay();
                        time = startOfDay.toString().replace('T', ' ');
                        dateFormat = new SimpleDateFormat("yyyy-MM-dd hh:mm");
                        date = dateFormat.parse(time);
                        unixTime = (long) date.getTime() / 1000;
                        for (int j = 0; j < a.getFinalt().size(); j++) {
                                long t = a.getFinalt().get(j);
                                if (t <= tnow && t >= unixTime) {
                                        e_year = e_year + a.getFinalp().get(j);
                                        e_total = e_total + a.getFinalp().get(j);
                                }
                        }

                        JSONObject t = new JSONObject();
                        t.put("DT", a.getType());
                        t.put("E_Day", e_day);
                        edaytotal = edaytotal + e_day;
                        t.put("E_Total", e_total);
                        etotaltotal = etotaltotal + e_total;
                        t.put("E_Year", e_year);
                        eyeartotal = eyeartotal + e_year;
                        t.put("P", p);
```

```
                        ptotal = ptotal + p;
                        inverters.put(String.valueOf(i + 1), t);
                }

                site.put("E_Day", edaytotal);
                site.put("E_Total", etotaltotal);
                site.put("E_Year", eyeartotal);
                site.put("P_PV", ptotal);
                data.put("Inverters", inverters);
                body.put("Data", data);
                ar.put("Body", body);

                return ar.toString() + "\n";
        }

        public static ArrayList<solarPark> init() {
                // read configuration file
                String filename = "configs/solarpark.conf";
                String conf = "";
                try {
                        conf = readFile(filename);
                } catch (Exception e) {
                        e.printStackTrace();
                        System.out.println("error reading file");
                        System.exit(-1);
                }
                ArrayList<solarPark> parks = new ArrayList<solarPark>();
                try {
                        parks = parseJson(conf);
                } catch (Exception e) {
                        e.printStackTrace();
                        System.out.println("error parsing file");
                        System.exit(-1);
                }
                try {
                        // start simulation for each of the parks in the configuration file after
                        // successful initialize
                        for (int i = 0; i < parks.size(); i++) {
                                parks.get(i).startSimulation();
                        }
                } catch (Exception e) {
                        e.printStackTrace();
                        System.out.println("error simulating");
                        System.exit(-1);
                }
                return parks;

        }

        public static String GetSensorRealtimeData(String scope, String datacollection, String deviceid,
                        ArrayList<solarPark> parks, String arg1, String arg2) throws Exception {
                if (scope.equals("System") && datacollection.equals("NowSensorData")) {
                        // construct response
                        JSONObject ar = new JSONObject();
                        // construct head
                        JSONObject head = new JSONObject();
                        head.put("RequestArguments", new JSONObject().put("DataCollection", "NowSensorData")
                                        .put("DeviceClass", "SensorCard").put("Scope", "System"));
                        head.put("Status", new JSONObject().put("Code", "0").put("Reason", "").put("UserMessage", ""));
                        DateTimeFormatter dtf = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");
                        ar.put("Head", head);

                        long tnow = System.currentTimeMillis() / 1000L;
                        if (arg1 != null && arg2 != null) {
                                return "only 1 argument allowed";
                        }
                        if (arg1 != null) {
                                try {
                                        SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd
hh:mm:ss");
                                        Date date = new Date();
                                        date = dateFormat.parse(arg1);
                                        tnow = (long) date.getTime() / 1000;
                                } catch (Exception e) {
                                        return "cant parse timestamp";
                                }
                        }
                        if (arg2 != null) {
                                try {
                                        tnow = Long.parseLong(arg2);
```

```
                } catch (Exception e) {
                        return "cant parse unixtimestamp";
                }
        }

        LocalDateTime now = LocalDateTime.ofInstant(Instant.ofEpochSecond(tnow),
TimeZone.getDefault().toZoneId());
        head.put("Timestamp", dtf.format(now).replace(" ", "T") + "+03:00");

        // get airtemp and irradiance
        solarPark a = parks.get(0);

        ArrayList<Long> times = new ArrayList<Long>();
        ArrayList<Double> air = new ArrayList<Double>();
        ArrayList<Integer> ghit = new ArrayList<Integer>();

        long before = a.getTimestamps().get(0);
        double beforeair = a.getAir_temp().get(0);
        int beforeghi = a.getGhi().get(0);
        for (int i = 1; i < a.getTimestamps().size(); i++) {
                long after = a.getTimestamps().get(i);
                double afterair = a.getAir_temp().get(i);
                int afterghi = a.getGhi().get(i);
                int count = 0;
                if (after - before < 60) {
                        continue;
                }
                for (long k = before; k < after; k = k + 60) {
                        times.add(k);
                        count++;
                }
                double stepair = (afterair - beforeair) / count;
                int stepghi = ((int) ((afterghi - beforeghi) / count));
                for (int k = 0; k < count; k++) {
                        air.add(beforeair);
                        ghit.add(beforeghi);
                        beforeair = beforeair + stepair;
                        beforeghi = beforeghi + stepghi;
                }

                before = after;
                beforeair = afterair;
                beforeghi = afterghi;
        }

        long closest = Math.abs(times.get(0) - tnow);
        int index = 0;
        for (int k = 1; k < times.size(); k++) {
                long dt = times.get(k);
                if (Math.abs(dt - tnow) < closest) {
                        closest = Math.abs(dt - tnow);
                        index = k;
                }
        }
        index = index - 3 * 60;
        double airtemp = air.get(index);
        int ghi = ghit.get(index);

        if (closest > 120) {
                head.put("Status",
                                new JSONObject().put("Code", "-1").put("Reason",
                                                "No historical data added. Please wait to get new
data (Max waiting time 1hour)")
                                                .put("UserMessage", "Try again in a few moments
or provide valid timestamp"));
                airtemp = 0.0;
                ghi = 0;
        }
        // construct body
        JSONObject body = new JSONObject();
        JSONObject data = new JSONObject();

        JSONObject inverters = new JSONObject();

        JSONObject t = new JSONObject();
        t.put("Unit", "°C");
        t.put("Value", String.valueOf(airtemp));
        inverters.put("0", t);// airtemp

        JSONObject t1 = new JSONObject();
```

```
                        t1.put("Unit", "°C");
                        t1.put("Value", String.valueOf(airtemp));
                        inverters.put("1", t1);// airtemp

                        JSONObject t2 = new JSONObject();
                        t2.put("Unit", "W/m2");
                        t2.put("Value", String.valueOf(ghi));// irradiance
                        inverters.put("2", t2);

                        JSONObject t3 = new JSONObject();
                        t3.put("Unit", "km/h");
                        t3.put("Value", "0");// windspeed
                        inverters.put("3", t3);

                        data.put("1", inverters);
                        body.put("Data", data);
                        ar.put("Body", body);

                        return ar.toString() + "\n";
                } else {
                        return "Wrong arguments";
                }

        }

        public static void main(String[] args) {
                System.setProperty("org.eclipse.jetty.util.log.class", "org.eclipse.jetty.util.log.StdErrLog");
                System.setProperty("org.eclipse.jetty.LEVEL", "OFF");
                BasicConfigurator.configure();
                // port(8080);
                get("/solar_api/v1/GetPowerFlowRealtimeData.fcgi", (request, response) -> {
                        String timestamp = null;
                        try {
                                timestamp = request.queryParams("Timestamp").replace("T", " ");
                        } catch (Exception e) {
                                System.out.println();
                        }
                        String unixtimestamp = request.queryParams("UnixTimestamp");
                        return GetPowerFlowRealtimeData(init(), timestamp, unixtimestamp);
                });
                get("/solar_api/v1/GetSensorRealtimeData.cgi", (request, response) -> {
                        String timestamp = null;
                        try {
                                timestamp = request.queryParams("Timestamp").replace("T", " ");
                        } catch (Exception e) {
                                System.out.println();
                        }
                        String unixtimestamp = request.queryParams("UnixTimestamp");
                        String scope = request.queryParams("Scope");
                        String datacollection = request.queryParams("DataCollection");
                        String deviceid = request.queryParams("DeviceId");
                        return GetSensorRealtimeData(scope, datacollection, deviceid, init(), timestamp, unixtimestamp);
                });

        }
}
```