ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΙΓΑΙΟΥ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΑΚΩΝ ΚΑΙ ΕΠΙΚΟΙΝΩΝΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Δοκιμή Συναρτήσεων ως Υπηρεσία

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Θωμάς Τόμτσης

Επιβλέπων: Κυριάκος Κρητικός (Αναπλ. Καθηγητής)

Μέλη εξεταστικής επιτροπής: Σπύρος Κοκολάκης (Καθηγητής), Χρήστος Γκουμόπουλος (Αναπλ. Καθηγητής)

Σάμος, Παρασκευή 8 Μαρτίου 2024

Πίνακας Περιεχομένων

Λίστα Εικόνων

Λίστα Πινάκων

ΠΕΡΙΛΗΨΗ

Η δοκιμή serverless συναρτήσεων (functions) διαφέρει σημαντικά από εκείνη των συμβατικών προϊόντων λογισμικού. Η λειτουργία των συναρτήσεων καθώς και η κατανεμημένη τους φύση, καθιστούν την ανάπτυξη δοκιμών πολύπλοκη. Επιπρόσθετα, η κατασκευή των συναρτήσεων αλλά και των δοκιμών τους, εξαρτάται πολλές φορές από την χρήση εργαλείων και πακέτων ανάπτυξης λογισμικού ":":(Software Development Kits - SDKs) που βρίσκονται υπό την διαχείριση των παρόχων νέφους, με αποτέλεσμα τον ελάχιστο έλεγχο στο περιβάλλον εκτέλεσης των δοκιμών και την έλλειψη εργαλείων αποσφαλμάτωσης[1].

Στόχος της ανάπτυξης της RESTful υπηρεσίας Ίκαρος ήταν η δημιουργία ενός εργαλείου ανοιχτού κώδικα, του οποίου η λειτουργία θα ήταν διαφανής, προβλέψιμη και θα έδινε πλήρη έλεγχο στον χρήστη για την δημιουργία των δοκιμών και την εκτέλεση τους. Ταυτόχρονα, δεν θα επιβάρυνε τον χρήστη με τις λεπτομέρειες της διάταξης μιας συνάρτησης σε διαφορετικούς παρόχους ενώ για την σύνθεση των δοκιμών θα βασιζόταν αποκλειστικά στην χρήση γνωστών και οικείων εργαλείων. Πράγματι, η υπηρεσία Ίκαρος είναι ικανή να εκτελεί αυτοματοποιημένα λειτουργικές και μη-λειτουργικές δοκιμές για serverless συναρτήσεις στις πλατφόρμες AWS Lambda και Google Cloud Functions, ενσωματώνοντας τα πολύ γνωστά εργαλεία Rest Assured και JMeter, να διατάσει τις συναρτήσεις χρησιμοποιώντας το εργαλείο Terraform και να παράγει αναφορές που περιέχουν τα ευρήματα των δοκιμών αξιοποιώντας το εργαλείο BiRT.

Η υπηρεσία αυτή αναπτύχθηκε για ένα μεγάλο εύρος χρηστών που κυμαίνεται από τις επιχειρήσεις μέχρι και την ακαδημαϊκή/ερευνητική κοινότητα. Επίσης, διενεργεί με αυτοματοποιημένο τρόπο τις δοκιμές με βάση τις πραγματικές ανάγκες και προτιμήσεις του χρήστη. Τέλος, δεν απαιτεί ειδικές γνώσεις πάνω στο νέφος από τον χρήστη αλλά χρειάζεται να διαμορφωθεί με τα στοιχεία των λογαριασμών του χρήστη στους δύο παρόχους νέφους που υποστηρίζονται.

Λέξεις Κλειδιά: serverless, AWS Lambda, Google Cloud Functions, λειτουργική δοκιμή, δοκιμή απόδοσης, Terraform, REST Assured, JMeter, λογισμικό ανοικτού κώδικα

SUMMARY

Testing serverless functions differs significantly from that of conventional software products. The way serverless functions operate as well as their distributed nature make test development inherently complex. In addition, the development of functions and their tests often relies on the usage of tools and software development kits (SDKs) managed by cloud providers, resulting in minimal control over the test execution environment and a lack of debugging tools[1].

The goal of developing the Icarus RESTful service was to create an open-source tool whose operation would be transparent, predictable and give the user full control over the creation of tests and their execution. At the same time, it would not burden the user with details of a function's layout across different providers and would rely solely on the use of familiar and well-known tools to compose tests. Indeed, the Icarus service is capable of performing automated functional and non-functional tests for serverless functions on the AWS Lambda and Google Cloud Functions platforms, integrating the well-known Rest Assured and JMeter tools, deploying functions using the Terraform tool, and generating reports containing the test findings by leveraging the BiRT tool.

This service was developed for a wide range of users including businesses and the academic/research community. It also performs automated testing based on the actual needs and preferences of the user. Finally, it does not require any specific knowledge about the cloud from the user and only needs to be configured with the user's account details at the two supported cloud providers.

Keywords: serverless, AWS Lambda, Google Cloud Functions, functional testing, performance testing, Terraform, REST Assured, JMeter, open source software

1. ΕΙΣΑΓΩΓΗ

Το κλείδωμα σε προμηθευτή είναι ένα από τα μεγαλύτερα προβλήματα στην υπολογιστική νέφους. Το λειτουργικό κόστος μιας υπηρεσίας καθώς και η ποιότητα με την οποία προσφέρεται είναι συνυφασμένες με την επιχειρηματική στρατηγική του παρόχου, ευθυγραμμίζοντας έτσι την στρατηγική νέφους (Cloud strategy) ενός οργανισμού, με του παρόχου. Ενώ με αυτόν τον τρόπο προσφέρεται εύκολη μετάβαση σε και χρήση των υπηρεσιών νέφους, μακροπρόθεσμα μια επιχείρηση κινδυνεύει να αντιμετωπίσει υψηλό κόστος σε περίπτωση ανάγκης αλλαγής παρόχου (πχ. λόγω μη ευθυγράμμισης με την στρατηγική νέφους του πλέον), ή ανάπτυξης ιδιόκτητων υποδομών τεχνολογίας πληροφοριών. Η πολυπλοκότητα και το κόστος της μετανάστευσης των δεδομένων καθώς και η απαιτητική και κοστοβόρα μετατροπή των υπηρεσιών και προϊόντων της επιχείρησης έτσι ώστε να μπορούν να χρησιμοποιηθούν από μια άλλη πλατφόρμα είναι κρίσιμοι παράγοντες που συχνά έχουν ως αποτέλεσμα είτε την αποφυγή χρήσης της υπολογιστικής νέφους είτε την μη-βέλτιστη χρήση της (λόγω του κλειδώματος σε έναν πάροχο).

Στο μοντέλο συνάρτησης ως υπηρεσία ο χρήστης εξαρτάται άμεσα από έναν πάροχο αφού η υλοποίηση της συνάρτησης συνήθως χρησιμοποιεί τα πακέτα ανάπτυξης λογισμικού που διαθέτει ο πάροχος, πρέπει πάντοτε να πληροί συγκεκριμένες αρχιτεκτονικές προδιαγραφές καθώς και να διατάσσεται με συγκεκριμένο τρόπο στις υποδομές που προσφέρει ο πάροχος. Ταυτόχρονα, υπάρχει μεγάλη διαφοροποίηση στον τρόπο υλοποίησης του μοντέλου Συνάρτηση ως υπηρεσία σε κάθε πάροχο, επηρεάζοντας άμεσα το κόστος λειτουργίας και την κλιμακωσιμότητα κάθε συνάρτησης. Τέλος, σε πολλές περιπτώσεις, η χρήση μιας συνάρτησης συνδέεται με λειτουργικότητες που παρέχονται με την μορφή μιας ιδιόκτητης υπηρεσίας, όπως λειτουργικότητες ουρών μηνυμάτων και οριζόντια κλιμακώσιμων βάσεων δεδομένων.

Συνεπώς, ενώ η επιλογή του σωστού παρόχου είναι υψίστης σημασίας για έναν χρήστη ή οργανισμό, η διαδικασία επιλογής συχνά είναι σύνθετη και απαιτεί την χειρωνακτική υλοποίηση δοκιμών που θα κρίνουν την καταλληλότητα ενός παρόχου. Οι δοκιμές αυτές πρέπει είτε να εκτελεστούν σε ένα περιβάλλον που προσωμοιώνει εκείνο του παρόχου, είτε να χρησιμοποιούν τις υποδομές του παρόχου και να αναπτυχθούν κάνοντας χρήση των εργαλείων που προσφέρονται από τους ίδιους τους προμηθευτές υπηρεσιών νέφους. Η προσομοίωση των βασικών εγκαταστάσεων ενός παρόχου είναι ιδιαίτερα προβληματική, καθώς οι πραγματικές συνθήκες δεν δύναται να αναδημιουργηθουν με ακρίβεια και η υλοποίηση προσομοιώσεων είναι μια σύνθετη διαδικασία. Ταυτόχρονα, η συγγραφή και διάταξη των συναρτήσεων σε νέφη διαφορετικών παρόχων αλλά και η συγγραφή δοκιμών προϋποθέτουν εξοικείωση με τον τρόπο λειτουργίας διαφορετικών παρόχων και των εργαλείων τους, δημιουργώντας ανάγκες πρόσληψης ή ενσωμάτωσης εξειδικευμένων επαγγελματιών.

Η διπλωματική αντιμετωπίζει το συγκεκριμένο ζήτημα με την υλοποίηση ενός απλού και εύχρηστου εργαλείου με την μορφή μιας RESTful υπηρεσίας ιστού (web service). Η υπηρεσία αυτή, η οποία ονομάζεται Ίκαρος, απλοποιεί την συγγραφή και την εκτέλεση αυτοματοποιημένων δοκιμών για συναρτήσεις, ενώ ταυτόχρονα μπορεί να διατάξει τις συναρτήσεις σε διαφορετικούς παρόχους. Οι χρήστες εισάγουν τη τοποθεσία του κώδικα της συνάρτησης και τις παραμέτρους των δοκιμών και ο Ίκαρος διατάσει τις συναρτήσεις, εκτελεί τις δοκιμές και αποθηκεύει τα ευρήματα των δοκιμών σε μια αναφορά. Η υπηρεσία μας υποστηρίζει την εκτέλεση τόσο λειτουργικών δοκιμών όσο και δοκιμών απόδοσης.

Οι δοκιμές απόδοσης έχουν διττό στόχο: (α) την εύρεση της βέλτιστης διαμόρφωσης κάθε συνάρτησης για τον πάροχο που επιλέγει ο χρήστης και (β) την διερεύνηση σε ποιο νέφος/πλατφόρμα μια συνάρτηση έχει την καλύτερη δυνατή απόδοση. Ο χρήστης εισάγει τις παραμέτρους της δοκιμής, τις επιθυμητές διαμορφώσεις (πόρων/υποδομής) καθώς και τις μετρικές που θα πρέπει να συλλεγούν. Μία διαμόρφωση περιλαμβάνει το μέγεθος της μνήμης, την περιοχή που θα διαταχθεί η συνάρτηση και στην περίπτωση του Google Cloud των αριθμών πυρήνων του επεξεργαστή. Η RESTful υπηρεσία μας διατάσει την συνάρτηση, εκτελεί την δοκιμή (είτε λειτουργική είτε μη λειτουργική είτε και τα δύο είδη δοκιμών), συλλέγει τα αποτελέσματα των μετρικών από τους παρόχους (σε περίπτωση μη λειτουργικής δοκιμής), παράγει ένα παραμετρικό μοντέλο απόδοσης της συνάρτησης με την χρήση γραμμικής αναδρομής και παρουσιάζει τα ευρήματα και το μοντέλο σε μια ευανάγνωστη μορφή για τον χρήστη με την μορφή μιας αναφοράς. Η παραγωγή ενός μοντέλου απόδοσης βοηθάει τον χρήστη να κατανοήσει καλύτερα την μη λειτουργική συμπεριφορά της συνάρτησης (κατά μήκος διαφορετικών διαμορφώσεων και φόρτων εργασίας) και συνεισφέρει στην επιλογή του κατάλληλου παρόχου.

Οι λειτουργικές δοκιμές αξιολογούν την σωστή λειτουργία μιας συνάρτησης. Ο χρήστης παραμετροποιεί τις λειτουργικές δοκιμές αντιστοιχίζοντας σε ένα σύνολο παραμέτρων εισόδου ένα σύνολο αναμενόμενων παραμέτρων εξόδου. Εφόσον μια συνάρτηση δοκιμαστεί με βάση αυτό το σύνολο (παραμέτρων εισόδου), παράγεται το μέρος της αναφοράς που αφορά αυτή τη λειτουργική δοκιμή, προσδιορίζοντας αν όντως η συνάρτηση παρήγαγε την αναμενόμενη έξοδο και άρα είχε τη σωστή συμπεριφορά.

Για την διενέργεια των δοκιμών εφαρμόστηκε το μοντέλο του μαύρου κουτιού (Black Box Testing). Η χρήση ενός τέτοιου μοντέλου κατά την υλοποίηση των δοκιμών εξασφαλίζει απλότητα και ανεξαρτησία από τις υποκείμενες τεχνολογίες υλοποίησης, ενώ ταυτόχρονα αντιμετωπίζει λειτουργικά και μη-λειτουργικά ζητήματα νωρίς στον κύκλο ανάπτυξης χωρίς να προϋποθέτει εξειδικευμένη γνώση τεχνολογιών νέφους και συνυπάρχοντας με πιο εξειδικευμένα μοντέλα δοκιμών.

Ο Ίκαρος υποστηρίζει την εκτέλεση δοκιμών και τη διάταξη συναρτήσεων στις πλατφόρμες AWS Lambda και Google Cloud Functions, έτσι ώστε ο χρήστης να

μπορεί να συγκρίνει τις ευρέως διαδεδομένες υπηρεσίες της Amazon με τις πολύ λιγότερο μελετημένες υπηρεσίες της Google[1]. Ο Ίκαρος υποστηρίζει τις καινούργιες Google Cloud Functions V2 που επιτρέπουν στον χρήστη την χειροκίνητη ρύθμιση των πυρήνων επεξεργαστή για μια συνάρτηση. Ο χρήστης έχει την δυνατότητα με αυτόν τον τρόπο να συγκρίνει την συμπεριφορά των συναρτήσεων και την αποτελεσματικότητα διαφόρων τεχνικών βελτιστοποίησης που προσφέρουν οι δύο αυτοί πάροχοι, βοηθώντας τον στην τελική επιλογή του καταλληλότερου παρόχου από τους δύο. Οι πάροχοι αυτοί επιλέχθηκαν προς υποστήριξη διότι είναι αρκετά γνωστοί και αξιόπιστοι και κατέχουν σημαντικό μερίδιο στην αγορά του υπολογιστικού νέφους.

Ο Ίκαρος, ακολουθώντας τις αξίες του ανοιχτού λογισμικού, μπορεί εύκολα να τροποποιηθεί για να καλύψει πληθώρα αναγκών χρήσης. Το γεγονός αυτό ενισχύεται από την ενσωμάτωση ευρέως διαδεδομένων εργαλείων ανοιχτού λογισμικού, όπως το Terraform. Η αρχιτεκτονική της εφαρμογής είναι αρκετά ευέλικτη έτσι ώστε να επιτρέψει την υποστήριξη επιπλέον προμηθευτών υπηρεσιών νέφους και παρέχεται στο ευρύ κοινό με την ιδιαίτερα ευέλικτη και ανεκτική άδεια MPL 2.0.

Η υπόλοιπη αναφορά της διπλωματικής δομείται ως εξής. Στο 2ο Κεφάλαιο θα αναλύσουμε τα κύρια χαρακτηριστικά του υπολογιστικού νέφους, του μοντέλου παράδοσης Συνάρτηση ως Υπηρεσία (Function-as-a-Service - FaaS) που αυτό προσφέρει καθώς και των δοκιμών λογισμικού. Στο 3ο Κεφάλαιο θα δοθούν λεπτομέρειες αναφορικά με την όλη ανάπτυξη της προτεινόμενης εφαρμογής, όπως οι απαιτήσεις που τέθηκαν, τα πλαίσια (ανάπτυξης) που χρησιμοποιήθηκαν και τα σχεδιαστικά μοντέλα που ακολουθήθηκαν. Στο 4ο Κεφάλαιο γίνεται επίδειξη της εφαρμογής με βάση ορισμένα σενάρια χρήσης της καθώς και παρέχονται οδηγίες εγκατάστασης της. Στο 5ο Κεφάλαιο παρουσιάζονται τα συμπεράσματα και η εμπειρία που αποκομίσθηκε από την περάτωση της διπλωματικής και δίνονται ορισμένες μελλοντικές κατευθύνσεις για την εξέλιξη του Ίκαρου.

2. ΥΠΟΒΑΘΡΟ

2.1 Υπολογιστικό Νέφος

Το Υπολογιστικό Νέφος είναι ένα μοντέλο για ενεργοποίηση πανταχού παρούσας, βολικής, κατ'απαίτηση πρόσβασης από το δίκτυο μιας διαμοιραζόμενης δεξαμενής διαμορφώσιμων υπολογιστικών πόρων που μπορούν να παρέχονται και να αποδεσμεύονται γρήγορα, με ελάχιστη προσπάθεια διαχείρισης ή αλληλεπίδρασης από τον πάροχο της υπηρεσίας[2].

Τα θεμελιώδη χαρακτηριστικά της υπολογιστικής νέφους συνοψίζονται ως εξής[3]:

1) **Χρησιμοποίηση Κατ' Απαίτηση (On Demand Usage / Self-Service)**
   Ένας καταναλωτής νέφους μπορεί να προσπελαύνει μονομερώς πόρους βασιζόμενος στο Υπολογιστικό Νέφος, όποτε τους χρειάζεται. Η χρήση των πόρων αυτών μπορεί να αυτοματοποιηθεί, οδηγώντας σε ένα περιβάλλον χρήσης αυτό-εξυπηρέτησης κατ'απαίτηση.

2) **Πανταχού Παρούσα Πρόσβαση (Ubiquitous Access)**
   Δυνατότητα μιας υπηρεσίας νέφους να είναι ευρέως προσπελάσιμη μέσω πρότυπων μηχανισμών από ετερογενείς πλατφόρμες λεπτών και πυκνών πελατών (thin and thick clients). Απαιτεί την υποστήριξη μιας ευρείας ποικιλίας συσκευών, πρωτοκόλλων μεταφοράς, διεπαφών και τεχνολογιών ασφάλειας. Επιπλέον, η αρχιτεκτονική της υπηρεσίας νέφους θα πρέπει να προσαρμόζεται στις ανάγκες των διαφόρων καταναλωτών νέφους.

3) **Πολλαπλή Μίσθωση (Multi-tenancy)**
   Ικανότητα εξυπηρέτησης πολλαπλών καταναλωτών, που είναι απομονωμένοι ο ένας από τον άλλον, από έναν μόνο πόρο νέφους (όπως είναι ένα φυσικό μηχάνημα). Ένας πάροχος νέφους υποστηρίζει την συνεκμετάλλευση των πόρων του χρησιμοποιώντας μοντέλα πολλαπλής μίσθωσης που βασίζονται στην τεχνολογία εικονικοποίησης. Με αυτό τον τρόπο, οι πόροι εκχωρούνται δυναμικά σύμφωνα με τις απαιτήσεις του κάθε καταναλωτή.

4) **Ελαστικότητα (Elasticity)**

Αυτοματοποιημένη δυνατότητα ενός νέφους να κλιμακώνει γρήγορα και διαφανώς πόρους ανάλογα με την ζήτηση, ώστε να ανταποκρίνεται σε συνθήκες χρόνου εκτέλεσης (ή άλλων μετρικών) που προκαθορίζονται από τον καταναλωτή ή πάροχο νέφους. Η ελαστικότητα θεωρείται βασικός λόγος υιοθέτησης του Υπολογιστικού Νέφους. Οι πάροχοι νέφους με τεράστιους πόρους μπορούν να προσφέρουν τη μεγαλύτερη δυνατή ελαστικότητα. Ταυτόχρονα, οι καταναλωτές έχουν την αίσθηση ότι οι δυνατότητες δέσμευσης πόρων είναι απεριόριστες και μπορούν να χρησιμοποιηθούν για τη δέσμευση οποιασδήποτε ποσότητας πόρων οποιαδήποτε χρονική στιγμή

5) **Μετρούμενη Χρήση (Measured Usage)**

Δυνατότητα μιας πλατφόρμας νέφους να παρακολουθεί τη χρήση των πόρων του νέφους από τους καταναλωτές και να τους χρεώνει αναλόγως με βάση το εκάστοτε μοντέλο χρέωσης. Η χρέωση βασίζεται στην πραγματική χρήση των πόρων, συνήθως με βάση ένα συγκεκριμένο χρονικό διάστημα μοναδιαίας χρέωσης. Η μετρούμενη χρήση δεν περιορίζεται στην παρακολούθηση στατιστικών στοιχείων για λόγους τιμολόγησης αλλά περιλαμβάνει τη γενική παρακολούθηση των πόρων του νέφους και την παραγωγή αντίστοιχων αναφορών σε ένα επίπεδο αφαίρεσης που είναι κατάλληλο για το είδος του παρεχόμενου πόρου / υπηρεσίας. Μέσω της παρακολούθησης, δίνεται η δυνατότητα του ελέγχου της χρήσης από τον διαχειριστή/πάροχο του νέφους.

6) **Ανθεκτικότητα (Resilience)**

Δυνατότητα μεταγωγής σε εφεδρικό σύστημα που διανέμει πλεονάζουσες υλοποιήσεις πόρων πάνω σε φυσικές τοποθεσίες. Η συγκρότηση πόρων μπορεί να γίνει εκ των προτέρων έτσι ώστε αν κάποιος παρουσιάσει κάποιο πρόβλημα να αντικαθίσταται από μια άλλη πλεονάζουσα υλοποίηση. Στο Υπολογιστικό Νέφος, η ανθεκτικότητα μπορεί να υποστηριχθεί είτε σε ένα νέφος ή κατά μήκος πολλαπλών νεφών. Η ανθεκτικότητα επιτρέπει την αναβάθμιση της αξιοπιστίας και της διαθεσιμότητας των εφαρμογών πάνω από Υπολογιστικό Νέφος.

Στην υπολογιστική νέφους συναντώνται οι εξής βασικοί ρόλοι[4], [3]:

- **Πάροχος νέφους (Cloud provider)**
  Παρέχει πόρους βασιζόμενους στο νέφος και καθιστά τις υπηρεσίες νέφους διαθέσιμες με βάση τις εγγυήσεις των συμφωνιών επιπέδου υπηρεσιών (Service Level Agreement, SLA) που υπόσχεται. Εκτελεί καθήκοντα διοίκησης και διαχείρισης για την διασφάλιση της συνεχής λειτουργίας της συνολικής υποδομής Υπολογιστικού Νέφους. Συνήθως, ο πάροχος είναι ιδιοκτήτης των πόρων που προσφέρονται αλλά σε ορισμένες περιπτώσεις, δύναται πάροχοι νέφους να μεταπωλούν τους πόρους που έχουν εκμισθώσει από άλλους παρόχους Υπολογιστικού Νέφους

- **Καταναλωτής νέφους (Cloud consumer)**
  Είναι ιδιώτης ή οργανισμός που έχει επίσημο συμβόλαιο ή διακανονισμό με έναν πάροχο Υπολογιστικού Νέφους για να χρησιμοποιεί του πόρους που ο δεύτερος παρέχει. Χρησιμοποιεί συνήθως έναν καταναλωτή υπηρεσίας (cloud service consumer), ένα πρόγραμμα λογισμικού που αλληλεπιδρά με την τεχνική διεπαφή / API μιας υπηρεσίας νέφους, για την πρόσβαση σε αυτήν.

- **Ιδιοκτήτης υπηρεσίας νέφους (Cloud service owner)**
  Ιδιώτης ή οργανισμός στον οποίο ανήκει νομικά μια υπηρεσία νέφους. Μπορεί να είναι καταναλωτής ή πάροχος του νέφους μέσω του οποίου προσφέρεται η υπηρεσία. Ο ιδιοκτήτης υπηρεσίας νέφους δεν καλείται ιδιοκτήτης πόρου νέφους επειδή ο ρόλος του ιδιοκτήτη υπηρεσίας νέφους εφαρμόζεται μόνο σε υπηρεσίες νέφους που είναι εξωτερικά προσπελάσιμοι πόροι ευρισκόμενοι μέσα σε ένα νέφος.

- **Διαχειριστής πόρου νέφους (Cloud resource administrator)**
  Άτομο ή οργανισμός υπεύθυνος για τη διαχείριση ενός πόρου βασιζόμενο στο νέφος. Μπορεί να είναι ο πάροχος νέφους ή ένα τρίτο μέρος που διαχειρίζεται με βάση μια σύμβαση, ανάμεσα σε αυτό και τον πάροχο, τον πόρο.

- **Ελεγκτής νέφους (Cloud auditor)**
  Ένα τρίτο μέρος (συχνά πιστοποιημένο) που εκτελεί ανεξάρτητες αποτιμήσεις περιβαλλόντων νέφους. Οι αρμοδιότητες ενός ελεγκτή νέφους περιλαμβάνουν την αξιολόγηση μηχανισμών και διαδικασιών ασφάλειας και την αποτίμηση ρίσκου και επίπτωσης στην απόδοση του συστήματος / περιβάλλοντος. Το αποτέλεσμα των ενεργειών αυτών είναι η ανεξάρτητη και αντικειμενική πιστοποίηση του περιβάλλοντος νέφους και η ενίσχυση της εμπιστοσύνης ως προς τον πάροχο αυτού του περιβάλλοντος από τους καταναλωτές.

- **Μεσίτης νέφους (Cloud broker)**
  Ένα τρίτο μέρος που αναλαμβάνει τη διαχείριση και τη διαπραγμάτευση της χρήσης υπηρεσιών νέφους μεταξύ καταναλωτών και παρόχων νέφους. Κύρια χαρακτηριστικά ενός μεσίτη νέφους είναι η ικανότητα παροχής μιας συνεπής διεπαφής σε πολλαπλούς διαφορετικούς παρόχους νέφους και η πλήρης διαφάνεια ως προς το ποιος οργανισμός παρέχει τις υπηρεσίες νέφους που ενισχύονται ή συσσωματώνονται.

- **Φορέας νέφους (Cloud carrier)**
  Τρίτο μέρος υπεύθυνο για την ενοποίηση δικτύων ευρείας περιοχής και άλλων χαρακτηριστικών τηλεπικοινωνιακών δικτύων υψηλής αξιοπιστίας για τη ολοκλήρωση της σύνδεσης ανάμεσα σε καταναλωτές και παρόχους νέφους. Αυτός ο ρόλος αναλαμβάνεται συχνά από παρόχους δικτύων και τηλεπικοινωνιών.

Τα κύρια μοντέλα παράδοσης νέφους (cloud delivery models) (ή αλλιώς μοντέλα/είδη υπηρεσιών νέφους)[2], [3] είναι τα εξής:

- **Υποδομή ως Υπηρεσία (Infrastructure as a Service - IaaS)**
  Παρέχει ποικιλία εικονικών μηχανών (Virtual Machines - VMs) πάνω από την ίδια φυσική υποδομή. Ειδικότερα, ο πάροχος προσφέρει προδιαμορφωμένες εικονικές μηχανές, με προκαθορισμένα λειτουργικά συστήματα, διαμόρφωση δικτύου και αποθηκευτικά μέσα. Επιπλέον, διατίθενται περαιτέρω τρόποι διαμόρφωσης στους χρήστες κατά την δημιουργία των σχετικών στιγμιότυπων (εικονικών μηχανών). Αυτό το μοντέλο παράδοσης παρέχει σε καταναλωτές νέφους ένα υψηλό επίπεδο ελέγχου και ευθύνης επί της παραμετροποίησης και της χρήσης. Οι παρεχόμενοι πόροι από μια υπηρεσία IaaS δεν συγκροτούνται εκ των προτέρων και η ευθύνη διαχείρισης εμπίπτει στον καταναλωτή. Οι πάροχοι νέφους μπορεί να λαμβάνουν με βάση SLAs υπηρεσίες IaaS από άλλους παρόχους για να κλιμακώσουν τα δικά τους περιβάλλοντα νέφους.

- **Πλατφόρμα ως Υπηρεσία (Platform as a Service - PaaS)**
  Παριστά ένα προκαθορισμένο περιβάλλον που αποτελείται από ήδη ανεπτυγμένους και συγκροτημένους πόρους. Το προσφερόμενο περιβάλλον μπορεί να ειδωθεί ως μια υπολογιστική πλατφόρμα που τυπικά περιλαμβάνει ένα λειτουργικό σύστημα, ένα περιβάλλον εκτέλεσης βασιζόμενο σε συγκεκριμένη γλώσσα προγραμματισμού (runtime, πλαίσια εφαρμογών, κλπ.), μια βάση δεδομένων και ένα διαδικτυακό εξυπηρετητή (web server). Οι προγραμματιστές εστιάζουν κυρίως στην ανάπτυξη και εκτέλεση του λογισμικού τους πάνω σε αυτή

τη πλατφόρμα. Τα διάφορα προϊόντα/υπηρεσίες PaaS προσφέρονται με διαφορετικές στοίβες ανάπτυξης.

- **Λογισμικό ως Υπηρεσία (Software as a Service - SaaS)**
  Πρόγραμμα λογισμικού προσφερόμενο ως διαμοιραζόμενη υπηρεσία νέφους. Μπορεί να προσφέρεται είτε από τον πάροχο του νέφους είτε από τρίτο μέρος που είναι είτε άμεσα είτε μη συνεργαζόμενο με τον πάροχο. Ενέχει πολύ περιορισμένο διαχειριστικό έλεγχο. Υπάρχει μια ολόκληρη αγοράς για προϊόντα SaaS, τα οποία προσφέρονται για διαφορετικούς σκοπούς και με διαφορετικούς όρους χρήσης.

Τα θεμελιώδη μοντέλα ανάπτυξης νέφους (cloud development models)[2], [3] είναι τα ακόλουθα:

- **Δημόσιο νέφος (Public cloud)**
  Δημόσια προσβάσιμο περιβάλλον νέφους που ανήκει σε ένα πάροχο νέφους, όπως μία επιχείρηση, ένα ακαδημαϊκό/ερευνητικό ίδρυμα, ή ένα κυβερνητικό οργανισμό. Οι πόροι σε ένα δημόσιο νέφος παρέχονται με βάση τα προαναφερόμενα μοντέλα παράδοσης, ενώ είτε προσφέρονται με κάποια χρέωση είτε εμπορευματοποιούνται μέσω άλλων οδών, όπως μέσω της διαφήμισης. Ο πάροχος είναι υπεύθυνος για την δημιουργία και συντήρηση του νέφους και των πόρων του. Το δημόσιο νέφος είναι αρκετά κλιμακώσιμο, προσιτό οικονομικά, λιγότερο ασφαλές, περισσότερο διαθέσιμο από οπουδήποτε ενώ χρησιμοποιεί αυστηρά και ανταγωνιστικά SLAs.

- **Κοινοτικό νέφος (Community cloud)**
  Παρόμοιο με δημόσιο νέφος αλλά η πρόσβαση σε αυτό περιορίζεται σε μια κοινότητα καταναλωτών νέφους. Το κοινοτικό νέφος μπορεί να ανήκει από κοινού σε μέλη της κοινότητας ή να παρέχεται από ένα τρίτο μέρος. Τα μέλη της κοινότητας συνήθως μοιράζονται την ευθύνη ορισμού και ανάπτυξης του κοινοτικού νέφους ενώ η συμμετοχή στην κοινότητα δεν εξασφαλίζει την πρόσβαση ή τον έλεγχο όλων των πόρων του νέφους. Η περιορισμένη πρόσβαση σε εξωτερικά μέλη επιτρέπεται μόνο σε ορισμένες περιπτώσεις.

- **Ιδιωτικό νέφος (Private cloud)**
  Ανήκει σε έναν μόνο οργανισμό, μπορεί να είναι μικρό σε μέγεθος και επιτρέπει σε έναν οργανισμό να χρησιμοποιεί τεχνολογία Υπολογιστικού Νέφους ως τρόπο κεντρικοποίησης της πρόσβασης σε πόρους Τεχνολογίας Πληροφοριών από διαφορετικά μέρη, τοποθεσίες ή τμήματα του οργανισμού. Μπορεί να λειτουργεί ως ένα πλήρως ελεγχόμενο περιβάλλον έτσι ώστε ορισμένες από τις προκλήσεις και ζητήματα/προβλήματα του νέφους (πχ. ασφάλειας, περιορισμένου

ελέγχου) είτε να περιορίζονται είτε να εξαλείφονται. Η χρήση ενός ιδιωτικού νέφους μπορεί να αλλάξει τον τρόπο που ορίζονται και εφαρμόζονται τα όρια οργανισμού & εμπιστοσύνης. Η διαχείριση του περιβάλλοντος ιδιωτικού νέφους μπορεί να γίνει από υπαλλήλους του οργανισμού ή να ανατεθεί σε τρίτα, εξωτερικά μέρη. Σε ένα ιδιωτικό νέφος, ο οργανισμός παίζει τόσο το ρόλο του καταναλωτή όσο και του παρόχου του νέφους.

- **Υβριδικό νέφος (Hybrid cloud)**
  Περιβάλλον νέφους που αποτελεί σύνθεση δύο ή περισσότερων ξεχωριστών υποδομών νέφους (ιδιωτικές, κοινοτικές ή δημόσιες), οι οποίες παραμένουν μοναδικές οντότητες αλλά δεσμεύονται μαζί από πρότυπες ή ιδιόκτητες τεχνολογίες που επιτρέπουν την φορητότητα δεδομένων και εφαρμογών. Πρόκειται ουσιαστικά για ένα υβριδικό μοντέλο ανάπτυξης νέφους. Η συνήθης μορφή του υβριδικού νέφους περιλαμβάνει ένα ιδιωτικό νέφος και την προαιρετική χρήση επιπλέον πόρων κατ'απαίτηση (πχ. όταν εξαντληθούν αυτοί του ιδιωτικού νέφους) από το δημόσιο νέφος. Με αυτό τον τρόπο, εξασφαλίζεται η εκμετάλλευση της υπολογιστικής δύναμης του δημόσιου νέφους διατηρώντας σημαντικές ιδιότητες του ιδιωτικού νέφους, όπως η ασφάλεια. Οι υβριδικές αρχιτεκτονικές ανάπτυξης νέφους μπορεί να είναι σύνθετες καθώς και δύσκολες στην υλοποίηση και συντήρηση εξαιτίας της πιθανής ανομοιότητας των περιβαλλόντων νέφους που ενοποιούνται ή της δυσκολίας στην διαίρεση των ευθυνών μεταξύ των παρόχων των διαφορετικών ειδών νέφους.

2.2 Συνάρτηση ως Υπηρεσία (Function as a Service)

Η υπολογιστική χωρίς διακομιστή (Serverless Computing) είναι ένα μοντέλο υπολογιστικού νέφους που περιλαμβάνει μια κατηγορία πλατφορμών υπολογιστικού νέφους που επιτρέπουν την ανάπτυξη και την εκτέλεση των εφαρμογών (ή των συστατικών τους) στο νέφος χωρίς την κατανομή και την διαχείριση εικονικών διακομιστών και πόρων ή άλλων λειτουργικών πτυχών από τη μεριά του ιδιοκτήτη της εφαρμογής (και ειδικότερα της ομάδας DevOps του). Η ευθύνη διαχείρισης των λειτουργικών πτυχών, όπως η ανοχή σε σφάλματα ή η ελαστική κλιμάκωση των υπολογιστικών, αποθηκευτικών και επικοινωνιακών πόρων, ώστε να ανταποκρίνονται στις ποικίλες εφαρμογές και απαιτήσεις, μεταβιβάζεται στον πάροχο του νέφους. Οι πάροχοι με την σειρά τους εφαρμόζουν τιμολόγηση βάσει πραγματικής χρήσης και χρεώνουν τους χρήστες του νέφους με λεπτομερή ανάλυση, ανάλογα με τους πόρους που οι εφαρμογές καταναλώνουν πραγματικά από την υποδομή του νέφους, όπως ο χρόνος που εκτελούνται υπολογισμοί, το ποσό της μνήμης και του αποθηκευτικού χώρου που χρησιμοποιούνται[5].

2.2.1 Ορισμός

Η Συνάρτηση ως Υπηρεσία (Function as a Service - FaaS) μπορεί να θεωρηθεί ως το πιο χαρακτηριστικό παράδειγμα του serverless computing. Ειδικότερα, η θεμελιώδης μονάδα υπολογισμού σε αυτή την περίπτωση ονομάζεται συνάρτηση (function) και εκτελείται σε απόκριση ενός εναύσματος, όπως ένα αίτημα HTTP. Οι προγραμματιστές/λειτουργοί (operators) υλοποιούν την επιχειρηματική λογική μιας εφαρμογής ή συστατικού της ως ένα σύνολο απομονωμένων συναρτήσεων, προσδιορίζουν τους υπολογιστικούς πόρους, όπως το μέγεθος της μνήμης και τους πυρήνες του επεξεργαστή, με τους οποίους κάθε συνάρτηση πρέπει να εκτελεστεί καθώς και ορίζουν τα εναύσματα (function triggers) που θα εκκινούν την εκτέλεση της συνάρτησης. Από την άλλη μεριά, ο πάροχος εγγυάται για την εκτέλεση της συνάρτησης όποτε το γεγονός με το οποίο συνδέεται το αντίστοιχο έναυσμα συμβεί. Σε αντίθεση με συμβατικά μοντέλα υπολογιστικής νέφους, οι χρήστες χρεώνονται βάση του χρόνου που οι πόροι χρησιμοποιούνται και όχι του χρόνου που οι πόροι έχουν δεσμευτεί[6].

Οι συναρτήσεις δίχως διακομιστή (Serverless Functions) υπόσχονται ομαλή κλιμάκωση ανεξάρτητα του εισερχόμενου φόρτου εργασίας, του υποκείμενου κώδικα και του τρόπου υλοποίησης.

2.2.2 Χαρακτηριστικά συναρτήσεων

Οι συναρτήσεις έχουν τα εξής χαρακτηριστικά:

- **Εφήμερες (Ephemeral)**
  Το περιβάλλον εκτέλεσης μιας συνάρτησης είναι προσωρινό και δεν δύναται πάντοτε να παραμένει διαθέσιμο με το πέρας της εκτελέσεώς της. Κάθε περιβάλλον εκτέλεσης είναι μοναδικό και απόλυτα απομονωμένο από όλα τα υπόλοιπα.

- **Οδηγούμενες από γεγονότα (Event driven)**
  Οι συναρτήσεις εκτελούνται ως απάντηση σε κάποιο εξωτερικό ερέθισμα (function trigger) ή γεγονός.

- **Ανεξάρτητες από προηγούμενες καταστάσεις (Stateless)**
  Εξαιτίας της ύπαρξης ενός προσωρινού περιβάλλοντος εκτέλεσης καθώς και της εκτέλεσης μονάχα μετά την ύπαρξη κάποιου γεγονότος, η εκτέλεση των συναρτήσεων δεν πρέπει να εξαρτάται από προηγούμενες καταστάσεις.

- **Εκτελούμενες σε ένα περιβάλλον εκτέλεσης διαχειρίσιμο από τον πάροχο και όχι τον χρήστη (Provider managed runtime environment)**
  Ο χρήστης επιλέγει ένα από τα διαθέσιμα περιβάλλοντα εκτέλεσης (runtime environment) που υποστηρίζει ο πάροχος έτσι ώστε να εκτελεστεί η συνάρτηση. Δεν υποστηρίζουν όλοι οι πάροχοι τα ίδια περιβάλλοντα, πάρα ταύτα τα πιο ευρέως χρησιμοποιούμενα υποστηρίζονται από όλους. Ορισμένοι πάροχοι, όπως η Amazon Lambda, επιτρέπουν στον χρήστη την εκτέλεση μιας συνάρτησης σε ένα προσαρμοσμένο από τον χρήστη περιβάλλον εκτέλεσης, αλλά η διαδικασία είναι σύνθετη.

- **Κλιμακώσιμες (Scalable)**
  Οι συναρτήσεις είναι κλιμακώσιμες και μπορούν να ανταποκριθούν σε οποιοδήποτε φόρτο εργασίας. Την ευθύνη για την κλιμάκωση την αναλαμβάνει ο πάροχος. Η πιο συνηθισμένη μορφή κλιμάκωσης είναι *'μία εκτέλεση συνάρτησης ανά αίτηση εξυπηρέτησης'*. Οι πάροχοι συνήθως οριοθετούν τον μέγιστο αριθμό παράλληλων εκτελέσεων μιας συνάρτησης. Συνεπώς, επιπλέον ταυτόχρονες αιτήσεις πάνω από αυτό το όριο απορρίπτονται.

- **Κατανεμημένες (Distributed)**
  Οι serverless εφαρμογές είναι από την φύση τους κατανεμημένες. Κάθε συστατικό της εφαρμογής υλοποιείται με την μορφή μιας συνάρτησης, η

οποία διατάσσεται στις υποδομές του παρόχου. Η επικοινωνία μεταξύ των συναρτήσεων συνήθως επιτυγχάνεται είτε με σύγχρονο τρόπο (HTTP αιτήματα), είτε με ασύγχρονο τρόπο (ουρές μηνυμάτων όπως Kafka και RabbitMQ) είτε με την χρήση εναυσμάτων, όπου μία συνάρτηση προκαλεί ένα γεγονός το οποίο έχει ως αποτέλεσμα την εκτέλεση μιας άλλης συνάρτησης. Αυτοί οι τρόποι επικοινωνίας προσφέρονται από τον κάθε πάροχο συνήθως με τη μορφή ιδιόκτητων υπηρεσιών που υλοποιούνται με ιδιόκτητες τεχνολογίες. Για παράδειγμα, η Amazon προσφέρει τις υπηρεσίες Simple Notification Service - SNS & Simple Queue Service - SQS και η Microsoft τις υπηρεσίες Azure Event Grid & Azure Service Bus. Ταυτόχρονα είναι εφικτή η υλοποίηση ολόκληρων εφαρμογών με την σύνθεση συναρτήσεων κάνοντας χρήση ιδιόκτητων τεχνολογιών, όπως AWS Step Functions από την Amazon και Azure Durable functions από την Microsoft.

2.2.3 Πλεονεκτήματα υπολογιστικής χωρίς διακομιστή

Είναι εμφανές πως το FaaS μοντέλο προσφέρει πληθώρα πλεονεκτημάτων:

- **Έμφαση στον κώδικα**
  Δεν απαιτείται η διαχείριση υποδομών από την πλευρά των χρηστών. Συνεπώς, υπάρχει περισσότερος χρόνος για την ανάπτυξη των εφαρμογών με αποτέλεσμα την ταχύτερη συγγραφή εφαρμογών ανώτερης ποιότητας και απόδοσης ή νέων χαρακτηριστικών και λειτουργιών τους.

- **Μειωμένο κόστος**
  Στο FaaS μοντέλο κοστολογούνται οι πόροι αναλόγως την χρήση τους και όχι την δέσμευση τους. Το αποτέλεσμα είναι πως υπάρχει διαφάνεια στον τρόπο κοστολόγησης και πληρωμή των πόρων που όντως χρησιμοποίησε η συνάρτηση. Ταυτόχρονα, το γεγονός πως δεν απαιτείται η κατοχή και η συντήρηση υποδομών μειώνει δραματικά το συνολικό κόστος ανάπτυξης εφαρμογών και επιτρέπει την γρήγορη παραγωγή τους με αποτέλεσμα την ταχύτερη αντίδραση μιας επιχείρησης ή ενός οργανισμού στις αλλαγές του επιχειρηματικού της περιβάλλοντος.

- **Καλύτερη απόδοση**
  Η κλιμάκωση των serverless εφαρμογών χαίρει ιδιαίτερης ευκολίας, καθώς ο πάροχος στον οποίο έχει διαταχθεί η εφαρμογή κλιμακώνει οριζόντια όλες τις συναρτήσεις που απαρτίζουν την εφαρμογή. Κατ’αυτόν τον τρόπο, μια serverless εφαρμογή απολαμβάνει ανώτερη απόδοση σε σύγκριση με μια συμβατική εφαρμογή, για την κλιμάκωση της οποίας απαιτείται μεγαλύτερη πολυπλοκότητα, κόστος και απευθείας επέμβαση από την ομάδα ανάπτυξης (σε αντίθεση με το FaaS μοντέλο όπου ο πάροχος αναλαμβάνει

την κλιμάκωση). Ταυτόχρονα, η εφαρμογή μπορεί να παρέχεται πιο κοντά στην φυσική τοποθεσία του χρήστη, μειώνοντας τον χρόνο απόκρισης. Είναι προφανές πως το υπολογιστικό αυτό μοντέλο είναι ιδανικό για εφαρμογές μαζικής παραλληλίας (massively-parallel applications) διότι είναι δυνατή η δημιουργία εκατοντάδων παράλληλων στιγμιοτύπων μιας συνάρτησης για την υλοποίηση λειτουργιών όπως την επεξεργασία εικόνας. Με αυτό τον τρόπο, είναι δυνατή η τάχιστη εκτέλεση τέτοιων λειτουργιών σε σχέση με τη χρήση παραδοσιακών εξυπηρετητών (είτε φυσικών είτε εικονικών).

2.2.4 Μειονεκτήματα υπολογιστικής χωρίς διακομιστή

Παρόλα τα πλεονεκτήματα, το FaaS μοντέλο υποφέρει από τα εξής μειονεκτήματα:

- **Καινούργιες προκλήσεις ασφάλειας**
  Όταν η διαχείριση των υποδομών μιας εφαρμογής εμπίπτει στον πάροχο, δεν είναι εφικτή η επικύρωση της ασφάλειας των δεδομένων της εφαρμογής. Το γεγονός αυτό δημιουργεί προβλήματα στην περίπτωση εφαρμογών που διαχειρίζονται ευαίσθητα ή προσωπικά δεδομένα και μπορεί να υπόκεινται σε νόμους που καθορίζουν συγκεκριμένες διαδικασίες για την διαχείριση και την διαφύλαξη αυτών.

- **Αυξημένο κόστος για μερικές εφαρμογές**
  Η αρχιτεκτονική που ακολουθεί η Συνάρτηση ως Υπηρεσία δεν είναι σχεδιασμένη για εφαρμογές που πρόκειται να εκτελούνται για μεγάλο χρονικό διάστημα, εξαιτίας της κοστολόγησης ανά δευτερόλεπτο χρήσης των πόρων. Λόγω αυτού του ζητήματος, το κόστος μιας τέτοιας αρχιτεκτονικής μπορεί να είναι αρκετά περισσότερο σε σχέση με άλλες μοντέρνες αρχιτεκτονικές νέφους, όπως αυτή των μικρο-υπηρεσιών. Επίσης, πολλές φορές μπορεί να είναι δύσκολο το “σπάσιμο” μιας εφαρμογής σε συναρτήσεις λόγω των σημαντικών περιορισμών που επιβάλλονται από τις serverless πλατφόρμες. Για παράδειγμα, όλες οι πλατφόρμες εφαρμόζουν ένα πλαφόν στη διάρκεια εκτέλεσης μιας συνάρτησης, πέρα από το οποίο η εκτέλεση της συνάρτησης διακόπτεται. Σε συνάρτηση με το γεγονός πως οι συναρτήσεις είναι ακαταστατικές (serverless), αυτό σημαίνει πως σε μια τέτοια περίπτωση ο όλος υπολογισμός (της συνάρτησης) θα έχει χαθεί, οδηγώντας σε αχρείαστη σπατάλη πόρων

- **Μειωμένη απόδοση για μερικές εφαρμογές**
  Κατά την πρώτη εκτέλεση μιας συνάρτησης, ο πάροχος παραμετροποιεί το περιβάλλον στο οποίο η συνάρτηση θα εκτελεστεί. Ο χρόνος

παραμετροποίησης ονομάζεται κρύο ξεκίνημα (Cold start) και είναι ο κύριος παράγοντας που επηρεάζει την απόδοση μιας συνάρτησης. Η αρχική αυτή παραμετροποίηση προκύπτει κάθε φορά που η συνάρτηση σταματάει να χρησιμοποιείται για ένα χρονικό διάστημα οπότε το αρχικό περιβάλλον εκτέλεσής της καταστρέφεται από τον πάροχο. Το χρονικό αυτό διάστημα εξαρτάται από κάθε πάροχο.

- **Κίνδυνος κλειδώματος σε πάροχο**
  Η χρήση των υποδομών ενός παρόχου και η συγγραφή συναρτήσεων γύρω από αυτές μοιραία δημιουργεί εξάρτηση σε αυτόν τον πάροχο, καθώς κάθε πάροχος προσφέρει διαφορετικές υποδομές και υπηρεσίες που λειτουργούν με διαφορετικό τρόπο. Ταυτόχρονα, η ασφαλής μετανάστευση δεδομένων που έχουν αποθηκευτεί στις βάσεις δεδομένων ενός παρόχου είναι μια σύνθετη και χρονοβόρα διαδικασία που απαιτεί μεγάλη προσοχή.

- **Μεγάλη δυσκολία στην περάτωση δοκιμών**
  Η συγγραφή και η εκτέλεση δοκιμών για τις συναρτήσεις είναι μια σύνθετη διαδικασία. Μάλιστα, δεν υπάρχουν συγκεκριμένα πρότυπα για την κατασκευή δοκιμών για serverless συναρτήσεις[7].
  Η εκτέλεση μοναδιαίων δοκιμών (Unit Tests) προϋποθέτει την πλήρη απομόνωση από εξωτερικούς παράγοντες καθώς και έλεγχο του περιβάλλοντος δοκιμής, συνεπώς η εκτέλεση μοναδιαίων δοκιμών συχνά πραγματοποιείται τοπικά και όχι στις υποδομές ενός παρόχου. Το γεγονός αυτό καθιστά αναγκαία την προσομοίωση των υποδομών του παρόχου καθώς και το περιβάλλον εκτέλεσης της συνάρτησης (πχ. οι υπηρεσίες στις οποίες εξαρτάται η συνάρτηση, όπως η βάση δεδομένων, άλλες συναρτήσεις, η ουρά μηνυμάτων κλπ). Αυτή η διαδικασία είναι σύνθετη και απαιτεί την χρήση έξτρα εργαλείων ενώ δεν οδηγεί στην δοκιμή της συνάρτησης σε ένα ρεαλιστικό περιβάλλον.

  Επιπλέον, η διαδικασία εκτέλεσης δοκιμών ενσωμάτωσης (Integration Tests) για serverless εφαρμογές διαφέρει σημαντικά από εκείνη του συμβατικού λογισμικού, καθώς όλοι οι πόροι βρίσκονται υπό την διαχείριση των παρόχων νέφους[7]. Όμως, η συγγραφή δοκιμών ενσωμάτωσης είναι πολύ σημαντική καθώς η πλειοψηφία των λαθών βρίσκεται στα σημεία σύνδεσης μεταξύ των συναρτήσεων. Αυτό προϋποθέτει την εκτέλεση των δοκιμών στο περιβάλλον του παρόχου (Cloud Integration Testing). Οι δοκιμές που εκτελούνται στο περιβάλλον του παρόχου πρέπει να παίρνουν υπόψη τις κρύες εκκινήσεις και τις αποτυχίες δικτύου ενώ ταυτόχρονα η ομάδα ανάπτυξης πρέπει να ανησυχεί για το κόστος που επιφέρει κάθε εκτέλεση δοκιμής.

  Ταυτόχρονα σε μεγάλες εφαρμογές που απαρτίζονται από πληθώρα συναρτήσεων, η κλήση μιας λειτουργίας της εφαρμογής μέσω του API

μπορεί να έχει σαν αποτέλεσμα την παραγωγή ενός υπέρογκου αριθμού μηνυμάτων στην ουρά μηνυμάτων καθώς και την παραγωγή μεγάλου αριθμού γεγονότων. Οι προγραμματιστές πρέπει να ελέγξουν ολόκληρη την ροή τόσο των μηνυμάτων όσο και των γεγονότων ενώ πρέπει να ληφθούν υπόψη και τα συμβάντα που προκύπτουν ασύγχρονα. Η δοκιμή των ασύγχρονων συμβάντων είναι ιδιαίτερα σύνθετη καθώς απαιτείται άμεση πρόσβαση στην ουρά γεγονότων (Event Bus) ενώ το σύνολο της διαδικασίας είναι ευάλωτο σε πληθώρα παρενεργειών εξαιτίας εξωτερικών συμβάντων[7].

Εξαιτίας των ασύγχρονων γεγονότων η μέτρηση του ποσοστού κάλυψης των δοκιμών είναι πολύ δύσκολη, καθώς απαιτείται λεπτομερής καταγραφή όλων των γεγονότων που θα μπορούσαν να προκαλέσουν εκτέλεση μιας συνάρτησης τόσο από άλλες συναρτήσεις όσο και από εξωτερικούς παράγοντες.

Η αναπαραγωγή σφαλμάτων είναι επίσης ιδιαίτερα σύνθετη καθώς στις serverless εφαρμογές δεν υπάρχει το ισοδύναμο της στοίβας λαθών, ενώ ταυτόχρονα η ανακατασκευή του σεναρίου εκτέλεσης πρέπει να επιτευχθεί από την συλλογή μηνυμάτων log. Αυτό δημιουργεί επίσης την ανάγκη για κατασκευή, διαχείριση και αποθήκευση κατανεμημένων μηνυμάτων log που παράγουν οι συναρτήσεις[7].

Η εκτενής χρήση των υποδομών ενός παρόχου τόσο για την εκτέλεση της συνάρτησης όσο και των αντίστοιχων δοκιμών, δημιουργούν μεγάλη εξάρτηση από τα εργαλεία που προσφέρει ο πάροχος για τον σκοπό αυτό, με αποτέλεσμα την δυσκολία μετανάστευσης των δοκιμών από έναν πάροχο σε έναν άλλο, ενώ ταυτόχρονα υπάρχει έλλειψη είτε ιδιόκτητων εργαλείων είτε εργαλείων ανοιχτού λογισμικού που να ανταποκρίνονται στα άνωθεν προβλήματα[1].

2.2.5 Βασικές serverless πλατφόρμες

Οι κύριες πλατφόρμες Συνάρτησης ως Υπηρεσία είναι[8]:

- **AWS Lambda:**
  Η πλατφόρμα αυτή προσφέρεται από την Amazon και είναι η πιο δημοφιλής παγκοσμίως. Προσφέρει πληθώρα υπηρεσιών και πακέτων ανάπτυξης serverless εφαρμογών. Στην πλατφόρμα αυτή ο χρήστης επιλέγει το μέγεθος της μνήμης καθώς και την τοποθεσία διάταξης της συνάρτησης ενώ ο αριθμός των πυρήνων του επεξεργαστή επιλέγεται αυτόματα αναλόγως το μέγεθος της μνήμης. Η πλατφόρμα Lambda υποστηρίζει τα περιβάλλοντα Node.js, Java, Python και .NET ενώ προσφέρει στον χρήστη την δυνατότητα

ενός εξατομικευμένου περιβάλλοντος εκτέλεσης που θα προσφέρει ο ίδιος (εφόσον πληροί ορισμένες προϋποθέσεις). Υλοποιεί τα ιδιόκτητα εναύσματα SNS, SNQ και EventBridge. Η πλατφόρμα AWS Lambda κοστολογεί την χρήση των συναρτήσεων με βάση τον συνολικό αριθμό αιτήσεων, τον συνολικό χρόνο περάτωσης κάθε αίτησης και τον όγκο των δεδομένων κλήσης. Ταυτόχρονα διαθέτει free quotas για όλα τα συναφή προϊόντα της και μια δοκιμαστική περίοδο που διαρκεί ένα έτος

- **Microsoft Azure Functions:**
  Το Azure προσφέρεται από την Microsoft και υποστηρίζει τα περιβάλλοντα εκτέλεσης C#, Java, JavaScript, Powershell, Python και TypeScript. Επιπλέον, υποστηρίζει την χρήση εξατομικευμένων περιβαλλόντων από τον χρήστη. Διαθέτει ευρεία γκάμα τοποθεσιών διάταξης των συναρτήσεων, όπως η πλατφόρμα Lambda, αλλά υλοποιεί διαφορετικό μηχανισμό διάταξης των συναρτήσεων. Οι συναρτήσεις του χρήστη παραμετροποιούνται σύμφωνα με το πλάνο χρέωσης που έχει επιλέξει ο χρήστης αυτόματα. Το μοντέλο εκτέλεσης συναρτήσεων είναι πιο σύνθετο από εκείνο της πλατφόρμας Lambda, προσφέροντας έναν μηχανισμό δεσμεύσεων (Binding) που χρησιμοποιείται συμπληρωματικά με τον μηχανισμό των εναυσμάτων. Η κοστολόγηση λειτουργεί σύμφωνα με πλάνα χρήσης και όχι με βάση την χρήση των πόρων, όπως είναι σύνηθες σε άλλες πλατφόρμες.

- **Google Cloud Functions:**
  Προσφέρεται από την εταιρεία Google, υποστηρίζει τα περιβάλλοντα Node.js, Python, Go, Java, Ruby, PHP και .NET. Επιπλέον, υποστηρίζει την χρήση εξατομικευμένων περιβαλλόντων από τον χρήστη καθώς και την δυνατότητα παραμετροποίησης των πυρήνων επεξεργαστή για μια συνάρτηση μέσω της υπηρεσίας Google Cloud Functions V2. Ο χρήστης κοστολογείται με βάση τον συνολικό χρόνο εκτέλεσης της συνάρτησης, τον αριθμό αιτήσεων στην συνάρτηση και τους πόρους που η συνάρτηση χρησιμοποιεί.

- **Alibaba Cloud Function Compute:**
  Προσφέρεται από την εταιρεία Alibaba και είναι από τους πιο δημοφιλείς παρόχους στην Ασία. Υποστηρίζει τα περιβάλλοντα χρήσης Node.js, Python, PHP, Java, .NET και Go. Επιπλέον, υποστηρίζει την χρήση εξατομικευμένων περιβαλλόντων από τον χρήστη. Η κοστολόγηση είναι ευέλικτη ενώ ο χρήστης μπορεί να επιλέξει ανάμεσα σε ‘pay-as-you-go’ χρεώσεις ή σε συγκεκριμένα πλάνα χρεώσεων.

Η παρακάτω εικόνα συγκρίνει τις προαναφερόμενες πλατφόρμες με βάση τα ακόλουθα κριτήρια: τον αριθμό των συναρτήσεων που υποστηρίζονται ανά λογαριασμό χρήστη, την προσφερόμενη διαθεσιμότητα και κλιμακωσιμότητα των

συναρτήσεων, τις γλώσσες προγραμματισμού που υποστηρίζονται, τις δυνατότητες ταυτόχρονης εκτέλεσης της κάθε συνάρτησης, τις περιοχές / αποθετήρια από όπου αντλείται ο κώδικας της συνάρτησης, το μέγιστο μέγεθος κώδικα συνάρτησης, τις εξαρτήσεις που απαιτούνται για τη σωστή διάταξη και λειτουργία μιας συνάρτησης, τους διάφορους τρόπους / μέσα κλήσης των συναρτήσεων, τις δυνατότητες παρακολούθησης και καταγραφής των συναρτήσεων και την υποστήριξη για προσωποποιημένα περιβάλλοντα εκτέλεσης (συναρτήσεων).

Πίνακας 1 - Σύγκριση των AWS Lambda, Azure Functions και Google Cloud Functions.

CLOUDWITHEASE
A Rendezvous with Cloud Technologies

COMPARISON TABLE: AWS Lambda vs Azure Functions vs Google Functions

Function	AWS Lambda	Azure Functions	Google Functions
Functions supported	unlimited functions supported	unlimited functions supported	1000 per project functions supported
Scalability and availability	It is transparent and automatic scaling	Supports automatic scaling	Supports automatic scaling
Languages supported	Node.js, Python, Java, C++, Go and Ruby	C#, JavaScript, F#, Python and TypeScript	Node.js, Python and Go
Execution	Per account per region - 1000 concurrent executions	Per function – 10 concurrent executions	Per function – 400 concurrent executions
Deployments	.ZIP to S3 or Lambda	GitHub, Visual Studio, Local git, Dropbox, Bitbucket	Google cloud source, .ZIP to cloud storage
Maximum code size	Compressed – 50 MB Uncompressed – 250 MB	User pays cost here. None	Compressed – 100 MB Uncompressed – 500 MB
Dependencies	Deployment packages per languages is required	Functions are written in dedicated editor and do not have dependencies	npm package.json
Triggers supported	Amazon services + API gateway	Microsoft cloud services and HTTP requests	Cloud Pub/Sub and cloud storage, HTTP requests
Logging and monitoring	Supported via CloudWatch logs and AWS x-ray	Has built in integration Azure application insights to monitor functions	Supported via StackDriver
Support for custom runtime	Using custom deployment packages or AWS Lambda layers	Using Azure functions custom handlers	Using custom docker images

Πηγή: https://web.archive.org/web/20240226115107/https://cloudwithease.com/lambda-vs-azure-functions-vs-google-cloud-functions/

2.2.6 Δεδομένα & Εφαρμογές της υπολογιστικής χωρίς διακομιστή

Η συντριπτική πλειοψηφία των serverless εφαρμογών[9]:

- Είναι διατεταγμένη στην πλατφόρμα AWS Lambda
- Χρησιμοποιεί ως περιβάλλον εκτέλεσης είτε Python είτε JavaScript

- Συνδυάζεται με τεχνολογίες αποθήκευσης νέφους ή βάσεις δεδομένων που βρίσκονται στο νέφος.
- Απαρτίζεται από μικρό αριθμό συναρτήσεων, συνήθως πέντε ή λιγότερες
- Χρησιμοποιείται για φόρτους εργασίας που εμφανίζονται ξαφνικά και απαιτούν έντονη υπολογιστική ισχύ ή για λειτουργίες που έχουν ρίσκο απότομης και παρατεταμένης αύξησης του φόρτου εργασίας
- Χρησιμοποιείται για ανάπτυξη εφαρμογών που εξυπηρετούν μεγάλο όγκο δικτυακής κίνησης
- Έχει φόρτο εργασίας που είναι σύντομος σε διάρκεια
- Έχει χρόνος εκτέλεσης των συναρτήσεών της που είναι κάτω του ενός λεπτού
- Αναπτύχθηκε κυρίως εξαιτίας του χαμηλού κόστους και του μειωμένου φόρτου διαχείρισης

Το πεδίο χρήσης του FaaS μοντέλου είναι ευρύ αλλά οι πιο κοινές περιπτώσεις χρήσης είναι η δημιουργία APIs, το streaming και η ασύγχρονη επεξεργασία, οι εργασίες παρτίδας (Batch tasks) καθώς και οι λειτουργικές εργασίες (Operation tasks). Η εφαρμογή της serverless αρχιτεκτονικής σε αυτές τις περιπτώσεις χρήσης μειώνει σημαντικά τα λειτουργικά κόστη και για αυτό και είναι ιδιαίτερα διαδεδομένη.

Γνωστό παράδειγμα αποτελεί η εταιρεία Netflix, η οποία χρησιμοποιεί την πλατφόρμα AWS Lambda για την εκτέλεση λειτουργικών εργασιών, όπως την κωδικοποίηση video, την δημιουργία αντιγράφων ασφαλείας, την διενέργεια audits σε EC2 στιγμιότυπα (instances) και την παρακολούθηση των υποδομών της.

Η χρήση συναρτήσεων για την ανάπτυξη APIs που διευκολύνουν την επικοινωνία σε εφαρμογές ιστού (web), κινητού (mobile) και διαδικτύου των πραγμάτων (Internet of Things - IoT) είναι επίσης ιδιαίτερα δημοφιλής. Ταυτόχρονα, το μοντέλο FaaS έχει χρησιμοποιηθεί και για επιστημονικό φόρτο εργασίας, όπως την πρόβλεψη σεισμών καθώς και την ανάλυση του (ανθρώπινου) γονιδιώματος.

2.3 Δοκιμή Λογισμικού

Η δοκιμή λογισμικού είναι μια διαδικασία που απαρτίζεται από όλες τις δραστηριότητες του κύκλου ζωής δοκιμών λογισμικού (Software Testing Life Cycle - STLC), τόσο τις στατικές όσο και τις δυναμικές. Αφορά τον σχεδιασμό, την ετοιμασία των δοκιμών και την αξιολόγηση προϊόντων λογισμικού, αλλά και συναφών προϊόντων, έτσι ώστε να εξακριβωθεί πως ικανοποιούνται συγκεκριμένες απαιτήσεις, επιτελείται ο σκοπός ανάπτυξης τους και ανιχνεύονται τυχόν ελαττώματα[10].

Η δοκιμή λογισμικού απαρτίζεται από τέσσερα βασικά επίπεδα δοκιμών[10]:

- **Δοκιμή Συστατικών (Component/Unit Testing):**
  Στόχος αυτού του επιπέδου είναι η αναζήτηση σφαλμάτων και η επικύρωση της ορθής λειτουργίας των συστατικών ή μεμονωμένων μονάδων λογισμικού (πχ προγράμματα, αντικείμενα, κλάσεις) που μπορούν να δοκιμαστούν σε απομόνωση από άλλα συστατικά. Οι δοκιμές συστατικών μπορούν να εκτελεστούν σε απομόνωση από το υπόλοιπο σύστημα, ανάλογα με τις ανάγκες του κύκλου ανάπτυξης και του συστήματος. Συχνά χρησιμοποιούνται mocks, stubs ή drivers για να αντικαταστήσουν τα συστατικά που έχουν αφαιρεθεί και να προσομοιώσουν την διασύνδεση μεταξύ συστατικών με απλό τρόπο. Η χρήση Test harnesses και η εικονικοποίηση υπηρεσιών μπορεί να παρέχει παρόμοια λειτουργικότητα.
  Κατά την πραγματοποίηση δοκιμών συστατικών, αναπτύσσονται περιπτώσεις δοκιμών (Test Cases) για κάθε συστατικό, έτσι ώστε να διαπιστωθεί η ορθή λειτουργία και συμπεριφορά του. Οι περιπτώσεις δοκιμής είναι αναγκαίες για να προσδιοριστεί εάν οι απαιτήσεις που έχουν διατυπωθεί για το συστατικό ικανοποιούνται. Εάν οι δοκιμές ολοκληρωθούν με επιτυχία τότε θεωρείται πως το συστατικό πληροί τις απαιτήσεις.

- **Δοκιμή ενσωμάτωσης (Integration Testing):**
  Οι δοκιμές ενσωμάτωσης εξετάζουν τις διεπαφές ανάμεσα σε συστατικά ή υπο-συστήματα και τις αλληλεπιδράσεις μεταξύ διαφορετικών μερών (διαφορετικών υποσυστημάτων ή/και συνθέσεις συστατικών που συγκροτούν μια λειτουργία) ενός συστήματος. Οι δοκιμές ενσωμάτωσης βασίζονται στην σχεδίαση του λογισμικού και του συστήματος, στην αρχιτεκτονική του λογισμικού/συστήματος και στις σχέσεις μεταξύ συστατικών ή αντικειμένων. Οι δοκιμές ενσωμάτωσης μπορούν να εκτελεστούν σε διάφορα επίπεδα, εξετάζοντας είτε τις διεπαφές και αλληλεπιδράσεις ανάμεσα στα συστατικά (Component Integration Testing) είτε τον συνδυασμό και τις αλληλεπιδράσεις (υπο-)συστημάτων (System Integration Testing).

- **Δοκιμή Συστήματος (System testing):**
  Οι δοκιμές συστήματος εστιάζουν στην συμπεριφορά ολόκληρου του συστήματος/προϊόντος (και όχι συγκεκριμένων μερών του), όπως αυτή προδιαγράφεται στις απαιτήσεις του. Ο κύριος στόχος αυτού του επιπέδου είναι η επικύρωση των χαρακτηριστικών και των απαιτήσεων του συστήματος, τόσο λειτουργικών όσο και μη λειτουργικών. Το περιβάλλον εκτέλεσης της δοκιμής συστήματος είναι ύψιστης σημασίας και πρέπει να αντικατοπτρίζει ένα ρεαλιστικό περιβάλλον, το οποίο θα φιλοξενήσει το τελικό έργο.

- **Δοκιμή αποδοχής (Acceptance testing):**
  Η δοκιμή αποδοχής επικυρώνει πως το σύστημα/εφαρμογή που αναπτύχθηκε ικανοποιεί τις ανάγκες και τις απαιτήσεις του τελικού χρήστη και πληροί τα κριτήρια αποδοχής που έχει αυτός θέσει. Οι δοκιμές αποδοχής αξιολογούν την ικανότητα ενός συστήματος να ανταπεξέλθει σε πραγματικά σενάρια χρήσης οπότε δεν εστιάζουν στην εύρεση σφαλμάτων (εάν όμως βρεθεί μεγάλος όγκος αυτών, τότε υπάρχουν σοβαρά λάθη σε ολόκληρη την εφαρμογή).

Κάθε επίπεδο δοκιμής απαιτεί την ύπαρξη ενός κατάλληλου περιβάλλοντος δοκιμής (Test Environment). Ένα περιβάλλον δοκιμής ορίζεται ως ένα περιβάλλον που περιέχει υλικό, λογισμικό, προσομοιωτές, εργαλεία λογισμικού και άλλα υποστηρικτικά στοιχεία κατάλληλα για την διεξαγωγή δοκιμών. Μερικές φορές πολλά επίπεδα μπορούν να χρησιμοποιούν το ίδιο περιβάλλον (πχ. οι δοκιμές συστήματος και ενοποίησης συνήθως μοιράζονται το ίδιο περιβάλλον δοκιμής - αν το περιβάλλον είναι αρκετά ρεαλιστικό, παρόμοιο με εκείνο του τελικού χρήστη, τότε μπορεί να αφορά και τις δοκιμές αποδοχής)[10].

Οι δραστηριότητες δοκιμών σχετίζονται και εξαρτώνται άμεσα από τις δραστηριότητες της ανάπτυξης λογισμικού. Οι δραστηριότητες ανάπτυξης λογισμικού και ιδιαίτερα το μοντέλο ανάπτυξης καθορίζουν τον τρόπο περάτωσης των δοκιμών και τις τεχνικές που θα χρησιμοποιηθούν. Κάθε επίπεδο δοκιμής έχει διαφορετικό στόχο και σκοπό που εξαρτάται από το στάδιο στο οποίο βρίσκεται το έργο που αναπτύσσεται. Ο τρόπος οργάνωσης των δοκιμών πρέπει να ταιριάζει στον εκάστοτε κύκλο ανάπτυξης[10].

Κάθε δοκιμή κατηγοριοποιείται με βάση τον τελικό της στόχο στις εξής κατηγορίες[10]:

- **Λειτουργική δοκιμή (Functional Testing):**
  Οι λειτουργικές δοκιμές αξιολογούν την συμμόρφωση ενός συστήματος ή συστατικού συστήματος στις λειτουργικές του απαιτήσεις. Οι λειτουργικές δοκιμές εστιάζουν στην συμπεριφορά του προϊόντος, η οποία περιγράφεται

στις λειτουργικές απαιτήσεις, στις περιπτώσεις χρήσης, ή στις ιστορίες των χρηστών (user stories).

Οι λειτουργικές δοκιμές μπορούν να πραγματοποιηθούν σε όλα τα επίπεδα δοκιμών και συνήθως ακολουθούν δύο διαφορετικές προσεγγίσεις. Η πρώτη προσέγγιση έγκειται στην δημιουργία δοκιμών με βάση τις λειτουργικές απαιτήσεις του συστήματος/συστατικού (Requirements Based Testing), ενώ η δεύτερη αναπτύσσει δοκιμές με βάση τις επιχειρηματικές διαδικασίες της εφαρμογής (Business Process Based Testing).

Η πληρότητα των λειτουργικών δοκιμών μπορεί να μετρηθεί με την χρήση της μετρικής κάλυψης (Coverage), και βασίζεται στα στοιχεία της λειτουργίας της εφαρμογής που μπορούμε να απαριθμήσουμε.

Η κάλυψη είναι μια βασική μετρική στη δοκιμή λογισμικού που αποτιμά αν οι δοκιμές εξετάζουν επαρκώς όσο γίνεται μεγαλύτερο μέρος του κώδικα και των λειτουργιών του. Για τον προσδιορισμό της κάλυψης υπολογίζουμε τα εξής:

- Κάλυψη δηλώσεων (Statement Coverage):
  Το ποσοστό των εκτελέσιμων δηλώσεων του κώδικα που έχουν εξετάσει οι δοκιμές

- Κάλυψη αποφάσεων (Branch/Decision Coverage):
  Το ποσοστό των διαφορετικών αποφάσεων που μπορεί να πάρει το σύστημα και έχουν εξετάσει οι δοκιμές

- Κάλυψη μονοπατιών (Path Coverage):
  Το ποσοστό των διαφορετικών μονοπατιών εκτέλεσης (execution paths) που έχουν εξετάσει οι δοκιμές

- **Μη-λειτουργική δοκιμή (Non-functional Testing):**
  Οι μη-λειτουργικές δοκιμές αξιολογούν την συμμόρφωση ενός συστήματος ή συστατικού στις μη-λειτουργικές του απαιτήσεις. Οι μη λειτουργικές δοκιμές περιλαμβάνουν δοκιμές ευχρηστίας, συντηρησιμότητας, αξιοπιστίας, φορητότητας και ασφαλείας.
  Οι δοκιμές απόδοσης αποτελούν χαρακτηριστικό παράδειγμα μη-λειτουργικών δοκιμών που εστιάζουν στην απόδοση (απόκριση) του συστήματος ή των επιμέρους συστατικών του όταν βρίσκονται υπό διαφορετικούς όγκους φόρτου εργασιών. Στις δοκιμές απόδοσης συμπεριλαμβάνονται οι δοκιμές φορτίου (load tests), οι δοκιμές αντοχής (endurance tests), οι δοκιμές καταπόνησης (stress tests), οι δοκιμές κλιμακωσιμότητας (scalability tests), οι δοκιμές αιχμής (spike tests), οι δοκιμές ταυτοχρονισμού (concurrency tests) και οι δοκιμές χωρητικότητας (capacity tests).

Η εκτέλεση των δοκιμών απόδοσης μπορεί να πραγματοποιηθεί δυναμικά, εκτελώντας ένα πλήθος περιπτώσεων δοκιμών με την χρήση κάποιου εργαλείου όπως το JMeter. Για τη διενέργεια των διαφόρων τύπων δοκιμών επιδόσεων που περιγράφονται, πρέπει[11]

1) Να μοντελοποιηθεί, να παραχθεί και να υποβληθεί στο υπό δοκιμή σύστημα αντιπροσωπευτικός φόρτος εργασίας συστήματος (System Load).
    - ❖ Ο φόρτος συστήματος είναι συγκρίσιμος με τα δεδομένα εισόδου που χρησιμοποιούνται για τις περιπτώσεις δοκιμής (test cases) των
      λειτουργικών δοκιμών. Με άλλα λόγια, ο αναμενόμενος φόρτος συστήματος είναι η είσοδος στην περίπτωση των δοκιμών απόδοσης.
    - ❖ Η δημιουργία φόρτου συστήματος εξαρτάται από την απουσία λειτουργικών ελαττωμάτων στο υπό δοκιμή σύστημα, τα οποία μπορεί να επηρεάσουν αρνητικά (πχ. να μην είναι εφικτή) την εκτέλεση της δοκιμής.
    - ❖ Η αποτελεσματική και αξιόπιστη παραγωγή φόρτου αποτελεί βασικό παράγοντα επιτυχίας κατά τη διεξαγωγή δοκιμών απόδοσης.
    - ❖ Για την παραγωγή φόρτου συστήματος απαιτείται η χρήση των λειτουργικών προφίλ (operational profiles) που έχουν καθοριστεί για την εφαρμογή. Ένα λειτουργικό προφίλ αναπαριστά κατά προσέγγιση πως θα χρησιμοποιηθεί ένα σύστημα σε ρεαλιστικές συνθήκες. Αποτελείται από το είδος των εργασιών που θα εκτελεστούν, την συχνότητα τους, την στατιστική τους κατανομή σε ένα διάστημα χρόνου και το πλήθος των δεδομένων που επεξεργάζονται.
    - ❖ Για κάθε λειτουργικό προφίλ θα δημιουργηθεί ένα πλήθος προφίλ φόρτου (load profile). Ένα προφίλ φόρτου απαρτίζεται από συγκεκριμένες κατηγορίες χρηστών και συγκεκριμένο αριθμό χρηστών, όπου κάθε κατηγορία εκτελεί συγκεκριμένες εργασίες με συγκεκριμένο τρόπο (πχ συγκεκριμένο χρόνο σκέψης (think time), εκτέλεσης, ροή εργασιών κλπ). Στόχος είναι η υλοποίηση ενός ρεαλιστικού και αντιπροσωπευτικού σεναρίου χρήσης που να προσομοιώνει το λειτουργικό προφίλ (συστήματος).

2) Να συγκεντρωθούν τα αποτελέσματα των μετρικών
    - ❖ Οι μετρικές που θα επιλεγούν ποικίλλουν και επιλέγονται ανάλογα με το επιχειρηματικό πλαίσιο και τους στόχους της δοκιμής.

- ❖ Οι μετρικές που θα επιλεγούν καθορίζονται από το τεχνικό περιβάλλον (πχ IoT, Web, Cloud) και ειδικότερα από τις δυνατότητες που προσφέρει (πχ. για τη διαίσθηση και συλλογή των μετρήσεων των μετρικών).

3) Να παρουσιαστούν τα αποτελέσματα σε μια ευανάγνωστη και κατανοητή μορφή.
   - ❖ Η σωστή παρουσίαση των αποτελεσμάτων είναι κρίσιμης σημασίας, καθώς η έννοια της καλής και κακής απόδοσης είναι δυσδιάκριτη και εξαρτάται από την ερμηνεία και τις απαιτήσεις του συστήματος. Σε αντίθεση με τις λειτουργικές δοκιμές όπου η έννοια της σωστής συμπεριφοράς είναι μονοσήμαντη.

Οι μη λειτουργικές δοκιμές καθορίζουν τα αναμενόμενα αποτελέσματα σε ότι αφορά την εξωτερική συμπεριφορά της εφαρμογής. Αυτό σημαίνει ότι συνήθως χρησιμοποιούμε τεχνικές δοκιμών μαύρου κουτιού (Black box testing). Η πληρότητα των μη λειτουργικών δοκιμών μπορεί να μετρηθεί χρησιμοποιώντας την μετρική της κάλυψης (coverage) των μη λειτουργικών στοιχείων (πχ ασφάλεια, απόδοση)[11].

Οι κύριες τεχνικές εκτέλεσης δοκιμών είναι[10]:

- **Δοκιμή μαύρου κουτιού (Black box testing):**
  Διαδικασία για την εξαγωγή και/ή επιλογή περιπτώσεων δοκιμής με βάση μια ανάλυση των προδιαγραφών, είτε λειτουργική είτε μη λειτουργική, ενός συστατικού ή συστήματος, χωρίς αναφορά στην εσωτερική του δομή ή τον υποκείμενο τρόπο υλοποίησης του. Όλες οι τεχνικές δοκιμών "μαύρου κουτιού" έχουν το κοινό χαρακτηριστικό ότι βασίζονται σε ένα μοντέλο (επίσημο ή ανεπίσημο) κάποιας πτυχής του συστήματος, το οποίο επιτρέπει τη δημιουργία περιπτώσεων δοκιμών με συστημικό τρόπο.

- **Δοκιμή λευκού κουτιού (White box testing):**
  Διαδικασία για την εξαγωγή και/ή επιλογή περιπτώσεων δοκιμής με βάση την ανάλυση της εσωτερικής δομής ενός στοιχείου ή συστήματος. Όλες οι τεχνικές που βασίζονται στη διαδικασία λευκού κουτιού έχουν το κοινό χαρακτηριστικό ότι βασίζονται στον τρόπο με τον οποίο κατασκευάζεται ή σχεδιάζεται το υπό δοκιμή λογισμικό. Αυτή η δομική πληροφορία χρησιμοποιείται για να εκτιμηθεί ποια μέρη του λογισμικού έχουν ήδη εξεταστεί από ένα σύνολο δοκιμών (που συχνά προέρχονται από διάφορες τεχνικές / είδη δοκιμών). Στη συνέχεια, μπορούν να προκύψουν πρόσθετες περιπτώσεις δοκιμών με συστημικό τρόπο για να καλυφθούν τα μέρη της

δομής της εφαρμογής που δεν έχουν αγγιχτεί προηγουμένως από καμία δοκιμή.

3. ΑΝΑΠΤΥΞΗ ΕΦΑΡΜΟΓΗΣ

Η ανάπτυξη της εφαρμογής ακολούθησε το μοντέλο ανάπτυξης του καταρράκτη (waterfall model) λόγω του γεγονότος πως οι απαιτήσεις ήταν καλά καθορισμένες και απίθανο να αλλάξουν. Υπήρξαν τέσσερις δραστηριότητες: ανάλυση απαιτήσεων, σχεδίαση, υλοποίηση και επικύρωση/δοκιμή. Τα αποτελέσματα των δραστηριοτήτων παρέχονται παρακάτω στις αντίστοιχες ενότητες του κεφαλαίου.

3.1 Λειτουργικές Απαιτήσεις

Οι κύριες οντότητες που απαρτίζουν το σύστημα είναι η συνάρτηση, η δοκιμή, η εκτέλεση δοκιμής, ο χρήστης και ο λογαριασμός πλατφόρμας χρήστη. Η δοκιμή περιλαμβάνει πληροφορίες για την κατάλληλη διαμόρφωση μιας δοκιμής συνάρτησης, αλλά και πληροφορίες για την ίδια την συνάρτηση. Μια δοκιμή μπορεί να εκτελεστεί πολλές φορές οπότε το σύστημα θα πρέπει να τη συσχετίζει με όλες τις εκτελέσεις της. Μια εκτέλεση έχει συγκεκριμένη κατάσταση και μπορεί να αντιστοιχεί σε συγκεκριμένα αποτελέσματα. Ένας χρήστης μπορεί να έχει πρόσβαση στις λειτουργίες της υπηρεσίας, εφόσον ταυτοποιηθεί, και έχει ένα προφίλ που καλύπτει την βασική πληροφορία που πρέπει να γνωρίζουμε για αυτόν τον χρήστη. Το προφίλ του χρήστη μπορεί να συσχετιστεί με όλους τους λογαριασμούς σε serverless / FaaS πλατφόρμες στις οποίες αυτός έχει πρόσβαση.

Παρακάτω αναλύουμε τις λειτουργικές απαιτήσεις του συστήματος / υπηρεσίας με βάση την οντότητα που αφορούν.

3.1.1 Διαχείριση Χρηστών

Το σύστημα θα πρέπει να είναι σε θέση να διαχειρίζεται τους χρήστες του. Οι λειτουργίες διαχείρισης περιλαμβάνουν τις εξής:

- ***εγγραφή χρήστη*:** θα πρέπει να επιτρέπεται σε έναν επισκέπτη να αιτείται την εγγραφή του ως χρήστης στο σύστημα. Κατά την αίτηση, θα πρέπει να παρέχονται το όνομα χρήστη (username), email του καθώς και ο κωδικός του (password). Θα πρέπει να ελέγχεται αν ο χρήστης με το ίδιο όνομα ή email υπάρχει. Σε αυτή την περίπτωση, η εγγραφή θα αποτυγχάνει και αντίστοιχο μήνυμα λάθους θα πρέπει να εμφανίζεται. Ο κωδικός του χρήστη θα πρέπει να κατακερματίζεται
    - Προαιρετικό: θα μπορούσε να υποστηρίζεται και εγγραφή χρήστη μέσω κάποιας άλλης υπηρεσίας, όπως Google Accounts.

- ***διαγραφή χρήστη*:** ο χρήστης θα πρέπει να μπορεί να αιτείται την διαγραφή του. Αυτό θα έχει ως επίπτωση την διαγραφή όλων των στοιχείων που αφορούν τον χρήστη, συμπεριλαμβανομένου στοιχείων δοκιμών.

- ***ενημέρωση (στοιχείων) χρήστη*:** ο χρήστης θα πρέπει να μπορεί να ανανεώνει βασικές πληροφορίες για το άτομό του όπως το ονοματεπώνυμό του, την ηλικία του, και το email του. Για λόγους επαναφοράς κωδικού, καλό είναι να αποθηκεύεται και η απάντηση σε μια από τις ερωτήσεις που εμφανίζει το σύστημα στον χρήστη (και ο χρήστης επιλέγει προς απάντηση). Επίσης, θα πρέπει να είναι σε θέση να διαχειρίζεται τους λογαριασμούς του σε FaaS πλατφόρμες. Η διαχείριση λογαριασμών θα περιλαμβάνει τις λειτουργίες προσθήκης, ενημέρωσης και διαγραφής στοιχείων λογαριασμών. Τα στοιχεία ενός λογαριασμού θα μπορούν να περιλαμβάνουν τίτλος/όνομα λογαριασμού, το όνομα της πλατφόρμας, το τελικό(ά) της σημείο(α), και πληροφορίες διαπίστευσης (πχ. όνομα και κωδικός χρήστη).

- ***επαναφορά/ενημέρωση κωδικού*:** εφόσον ο χρήστης επιθυμεί την επαναφορά κωδικού, το σύστημα θα του εμφανίζει μια σειρά από ερωτήσεις κι αυτός θα πρέπει να δώσει την απάντηση που έχει δώσει προηγουμένως σε μια από αυτές κατά την ενημέρωση των στοιχείων του. Εφόσον η απάντηση είναι σωστή, τότε θα δημιουργείται ένας προσωρινός ασφαλής κωδικός για τον χρήστη, ο οποίος θα αποστέλλεται στο email του. Έπειτα, ο χρήστης θα μπορεί να αιτηθεί την ενημέρωση του κωδικού του ώστε να τον αλλάξει. Σημειώνεται πως αν ο χρήστης δεν αλλάξει τον κωδικό μέσα σε συγκεκριμένο χρονικό διάστημα, τότε θα απενεργοποιείται ο λογαριασμός του. Για την ενημέρωση, ο χρήστης θα πρέπει να δώσει τόσο τον προηγούμενο όσο και τον νέο κωδικό (τον τελευταίο εις διπλούν). Εν γένει, οι κωδικοί θα εναπόκεινται σε ορισμένους περιορισμούς (πχ. τουλάχιστον 10 χαρακτήρων που να περιλαμβάνουν γράμματα, ψηφία και ειδικούς χαρακτήρες) ώστε να θεωρούνται έγκυροι και ασφαλείς. Για να επιτύχει η ενημέρωση, ο προηγούμενος κωδικός θα πρέπει να είναι σωστός και ο νέος κωδικός να είναι τόσο έγκυρος όσο και ασφαλής καθώς και να ταιριάζουν απόλυτα οι 2 τιμές του που παρέχονται από τον χρήστη.

- ***ταυτοποίηση***: ο χρήστης θα πρέπει να μπορεί να ταυτοποιείται στο σύστημα παρέχοντας το όνομα χρήστη & κωδικό του. Προαιρετικά, μπορεί να υποστηρίζεται και εξωτερική υπηρεσία ταυτοποίησης (πχ. Google Accounts). Έπειτα, θα παράγεται ένα μοναδικό token με συγκεκριμένο χρονικό όριο εγκυρότητας, το οποίο θα μπορεί να χρησιμοποιείται κατά την αίτηση εκτέλεσης των διαφόρων λειτουργιών που παρέχονται από το σύστημα προς τους χρήστες του.

3.1.2 Διαχείριση δοκιμών

Οι λειτουργίες διαχείρισης μιας δοκιμής περιλαμβάνουν:

- ***δημιουργία νέας δοκιμής***: ο χρήστης θα πρέπει να είναι σε θέση να δημιουργήσει μια νέα δοκιμή. Σε αυτή την περίπτωση, κατά την σχετική αίτησή του θα πρέπει να προσδιορίσει τις ακόλουθες πληροφορίες:

    - όνομα δοκιμής
    - περιγραφή δοκιμής: μια προαιρετική κειμενική περιγραφή της δοκιμής
    - όνομα συνάρτησης: το όνομα της συνάρτησης προς δοκιμή
    - URL εικόνας συνάρτησης: εφόσον η συνάρτηση αντιστοιχεί σε μια εικόνα δοχείου, μπορεί να δοθεί URL προς το αντίστοιχο αποθετήριο (πχ. στο DockerHub)
    - URL κώδικα συνάρτησης: εφόσον το πακέτο της συνάρτησης είναι διαθέσιμο, μπορεί να δοθεί το αντίστοιχο URL προς το πακέτο αυτό. Σημειώνεται πως ένα πακέτο συνάρτησης συνήθως είναι ειδικό προς μια serverless/FaaS πλατφόρμα
    - URL & HTTP μέθοδος συνάρτησης εφόσον η συνάρτηση έχει διαταχθεί σε κάποια serverless/FaaS πλατφόρμα
    - Περιπτώσεις δοκιμής: μια σειρά από περιπτώσεις (λειτουργικής) δοκιμής. Κάθε περίπτωση προσδιορίζει συγκεκριμένες τιμές για τις παραμέτρους εισόδου της συνάρτησης και συγκεκριμένη τιμή εξόδου (ή/και είδος εξαίρεσης με αντίστοιχο μήνυμα). Για κάθε περίπτωση δοκιμής, εφόσον δοθούν οι αντίστοιχες τιμές εισόδου κατά την εκτέλεση της συνάρτησης, αναμένεται η αντίστοιχη τιμή εξόδου (ή/και είδος εξαίρεσης με αντίστοιχο μήνυμα). Προαιρετικά, οι περιπτώσεις δοκιμής θα μπορούσαν να ομαδοποιούνται σε ομάδες με συγκεκριμένα ονόματα. Σε μια τέτοια περίπτωση, η δοκιμή για μια ομάδα θα αποτυγχάνει εφόσον τουλάχιστον μια από τις περιπτώσεις δοκιμών της ομάδας αποτύχει ενώ θα επιτυγχάνει εφόσον όλες οι περιπτώσεις δοκιμής της ομάδας επιτύχουν. Η αντίστοιχη αναφορά (που θα πρέπει να παραχθεί) θα πρέπει επομένως να προσδιορίζει το αποτέλεσμα της δοκιμής ιεραρχικά: καθολικά, έπειτα στο επίπεδο των ομάδων και τέλος στο επίπεδο των περιπτώσεων δοκιμών.
    - Τοποθεσία λειτουργικής δοκιμής: εφόσον το URL πρόσβασης στην συνάρτηση παρέχεται, υποθέτουμε πως η λειτουργική δοκιμή πραγματοποιείται προς αυτή την διάταξη/τοποθεσία της συνάρτησης. Εναλλακτικά, ο χρήστης μπορεί να προσδιορίσει πως η εκτέλεση της συνάρτησης μπορεί να γίνει τοπικά ή σε συγκεκριμένη FaaS/serverless πλατφόρμα. Στην δεύτερη περίπτωση, θα πρέπει είτε να έχει δοθεί το URL εικόνας συνάρτησης ή το URL κώδικα συνάρτησης.

- Εύρος ταυτόχρονων χρηστών: εύρος από ταυτόχρονους χρήστες για την μη λειτουργική δοκιμή (πχ. 10 έως 100). Το βήμα αύξησης μπορεί να προκύπτει προκαθορισμένα ως εξής: max / min (πχ. 100/10 = 10). Διαφορετικά, θα πρέπει να παρέχεται ρητά από τον αιτούντα της δοκιμής (αλλά θα πρέπει να είναι αριθμός μεταξύ 1 & max).
- Εύρος αιτήσεων ανά ταυτόχρονο χρήστη - το βήμα αύξησης μπορεί και πάλι να προκύπτει είτε προκαθορισμένα είτε με βάση ρητή τιμή που δίνει ο χρήστης.
- Εύρος αριθμού πυρήνων - ισχύει ότι και παραπάνω για το βήμα αύξησης
- Εύρος μεγέθους κύριας μνήμης - εδώ το βήμα αύξησης θα εξαρτάται από την πλατφόρμα και άρα μπορεί να είναι προκαθορισμένο
- Λίστα από τοποθεσίες: Ο χρήστης μπορεί να δίνει μια σειρά από τοποθεσίες όπου πρέπει να πραγματοποιηθεί η μη λειτουργική δοκιμή. Οι τοποθεσίες αυτές θα μπορούσαν να είναι ήπειροι για απλούστευση. Εναλλακτικά, θα μπορούσαν να αφορούν και συγκεκριμένες χώρες (όπου υπάρχουν κέντρα δεδομένων των παρόχων νέφους)
- Λίστα από πλατφόρμες: Ο χρήστης μπορεί να δίνει μια λίστα από FaaS/serverless πλατφόρμες όπου θα πρέπει να πραγματοποιηθεί η μη λειτουργική δοκιμή. Υπάρχει μια συσχέτιση μεταξύ πλατφορμών και τοποθεσιών. Επομένως, αν η εκτέλεση μιας μη λειτουργικής δοκιμής πραγματοποιείται σε μια πλατφόρμα, τότε θα λογαριάζονται από την παρεχόμενη λίστα τοποθεσιών μόνο οι τοποθεσίες που υποστηρίζονται από την σχετική πλατφόρμα.
- Λίστα από μετρικές: Ο χρήστης θα πρέπει να παρέχει μια λίστα από μετρικές για τις οποίες θα πρέπει να παραχθούν αντίστοιχες συναρτήσεις που να σχετίζονται με τις παραμέτρους (πχ. φόρτος εργασίας, ποσότητες πόρων υποδομής) του μη λειτουργικού μέρους της τρέχουσας δοκιμής.

Τονίζεται πως για καλύτερη διαχείριση των προαναφερόμενων πληροφοριών, αυτές μπορούν να οργανωθούν σε μέρη/τμήματα όπως γενικές πληροφορίες δοκιμής, πληροφορίες συνάρτησης, πληροφορίες λειτουργικής δοκιμής και πληροφορίες μη λειτουργικής δοκιμής. Επίσης, εφόσον η δημιουργία της δοκιμής είναι επιτυχής, η δοκιμή θα σχετίζεται με την ημερομηνία δημιουργίας της.

- ***εμφάνιση δοκιμής***: θα πρέπει να εμφανίζονται όλα τα στοιχεία της δοκιμής, χωριζόμενα στα 4 προαναφερόμενα μέρη, καθώς και μια λίστα από όλες τις εκτελέσεις της δοκιμής

- ***ανανέωση δοκιμής***: τα στοιχεία μιας δοκιμής μπορούν να τροποποιούνται από τον χρήστη. Εφόσον η τροποποίηση είναι επιτυχής, προαιρετικά η δοκιμή μπορεί να σχετίζεται με την τελευταία ημερομηνία τροποποίησής της (δηλαδή

την τρέχουσα). Τόσο η ημερομηνία δημιουργίας της δοκιμής και τελευταίας τροποποίησής της δεν μπορούν να αλλάζουν από τον χρήστη (παρά μόνο από το σύστημα).

- ***διαγραφή δοκιμής***: η δοκιμή (όλα τα στοιχεία της) μαζί με όλες τις εκτελέσεις της θα πρέπει να διαγραφούν σε αυτή την περίπτωση.

- ***αναζήτηση δοκιμής***: το σύστημα θα επιτρέπει στον χρήστη να αναζητά τις δοκιμές με βάση το όνομά τους ή το όνομα της συνάρτησης που δοκιμάζεται. Η αναζήτηση μπορεί να γίνεται με λέξεις κλειδιά. Προαιρετικά θα μπορεί να επιτρέπεται και πιο εξεζητημένη αναζήτηση όπου θα μπορεί να παρέχονται οι λέξεις κλειδιά και ένα εύρος ημερομηνιών που θα πρέπει να καλύπτει η ημερομηνία δημιουργίας της δοκιμής. Το αποτέλεσμα της αναζήτησης είναι μια λίστα από δοκιμές που ταιριάζουν με τα δεδομένα (εισόδου) της αναζήτησης. Εφόσον ο χρήστης επιλέξει ένα από αυτά τα αποτελέσματα, θα εμφανίζονται τα στοιχεία της αντίστοιχης δοκιμής.

- ***εκτέλεση δοκιμής***: ο χρήστης θα πρέπει να είναι σε θέση να εκκινεί μια νέα εκτέλεση μιας συγκεκριμένης δοκιμής. Η εκτέλεση θα πραγματοποιείται ασύγχρονα. Ο χρήστης μέσω σχετικών λειτουργιών, θα μπορεί να διαχειρίζεται την εκτέλεση αυτή. Το αποτέλεσμα εκτέλεσης δοκιμής θα πρέπει να είναι μια αναφορά που θα μπορεί να οπτικοποιείται στον χρήστη ενώ ο τελευταίος θα μπορεί να την εκφορτώνει στο δικό του τοπικό σύστημα. Η αναφορά θα πρέπει να προσδιορίζεται τα εξής:

    - ποσοστό επιτυχίας λειτουργικής δοκιμής: αριθμός επιτυχών περιπτώσεων δοκιμής / συνολικό πλήθος των περιπτώσεων δοκιμής.
      Σημειώνουμε πως εφόσον υποστηριχθεί το χαρακτηριστικό της ομαδοποίησης των περιπτώσεων δοκιμών, τότε θα πρέπει επιπλέον να προσδιορίζονται στατιστικά ανά ομάδα (δηλ. αριθμός επιτυχών περιπτώσεων δοκιμής / συνολικό πλήθος δοκιμών ομάδας).
    - αποτυχημένες δοκιμές: αριθμός μη επιτυχών περιπτώσεων δοκιμής και λίστα από αυτές.
      Εφόσον υποστηριχθεί το χαρακτηριστικό της ομαδοποίησης των περιπτώσεων δοκιμής, τότε οι αποτυχημένες δοκιμές θα πρέπει να εμφανίζονται και ανά ομάδα.
    - άνω όρια: αριθμός από ταυτόχρονους χρήστες και πλήθος αιτήσεων ανά χρήστη που οδηγεί σε μη αναμενόμενη ή λανθασμένη συμπεριφορά της συνάρτησης που δοκιμάζεται. Αυτό θα πρέπει να γίνεται σε συνάρτηση με τους διαθέσιμους πόρους που παρέχονται στην συνάρτηση (πχ. 50 ταυτόχρονοι χρήστες και 10 αιτήσεις ανά χρήστη στην περίπτωση 2 πυρήνων και 1 GB κύριας μνήμης). Άρα τα όρια θα πρέπει να παρουσιάζονται ανά configuration/διαμόρφωση.

- συναρτήσεις μετρικών: μια σειρά από συναρτήσεις για κάθε μετρική (πχ. μέσος χρόνος εκτέλεσης) που να την συσχετίζουν με αντίστοιχες παραμέτρους του μη λειτουργικού μέρους της δοκιμής. Κάθε συνάρτηση μετρικής θα πρέπει να είναι ειδική προς μια FaaS/serverless πλατφόρμα

Τονίζεται πως η εκτέλεση του μη λειτουργικού μέρους της δοκιμής θα πραγματοποιείται σε κάθε επιθυμητή FaaS/serverless πλατφόρμα. Σε αυτή την πλατφόρμα θα πραγματοποιείται διάταξη της σχετικής συνάρτησης σε διαφορετικές διαμορφώσεις. Για παράδειγμα, μια διαμόρφωση θα είναι: 2 πυρήνες CPU, 1 GB κύριας μνήμης και τοποθεσία Ευρώπη. Σε κάθε διαμόρφωση, θα πραγματοποιούνται μια σειρά από δοκιμές φόρτου (load testing) όπου θα ποικίλλει ο φόρτος εργασίας (ταυτόχρονοι χρήστες και αριθμός αιτήσεων ανά χρήστη) με σκοπό να συλλεχθούν αντίστοιχες μετρήσεις για κάθε επιθυμητή μετρική. Με βάση τις τιμές που θα συλλεχθούν, θα παραχθεί και η τελική συνάρτηση της μετρικής για την τρέχουσα πλατφόρμα. Για την παραγωγή της συνάρτησης, θα πρέπει να εξεταστούν τεχνικές, όπως αυτής της γραμμικής παρεμβολής (linear regression).

Σημειώνεται επίσης πως η τοποθεσία είναι περισσότερο σημαντική για μετρικές που εξαρτώνται από την δικτυακή καθυστέρηση, όπως είναι ο χρόνος απόκρισης (response time). Σε αυτή την περίπτωση, αυτό που μας ενδιαφέρει είναι η απόσταση της τοποθεσίας όπου δημιουργείται ο φόρτος εργασίας σε σχέση με την τοποθεσία που έχει διαταχθεί η συνάρτηση. Αυτός θα είναι ένας παράγοντας με τον οποίο εν τέλει θα σχετίζεται η συνάρτηση της αντίστοιχης μετρικής.

3.1.3 Διαχείριση εκτελέσεων δοκιμών

Μόλις ο χρήστης αιτηθεί την εκτέλεση μιας δοκιμής, μπορεί να του εμφανίζεται μια σελίδα όπου θα προσδιορίζεται η κατάσταση της εκτέλεσης. Ο χρήστης θα μπορεί έπειτα να ανανεώνει την σελίδα μέχρι να ολοκληρωθεί η εκτέλεση της δοκιμής όπου και θα εμφανιστούν τα αντίστοιχα αποτελέσματα αυτής της εκτέλεσης. Εναλλακτικά, ο χρήστης θα μπορεί να εξετάζει και να διαχειρίζεται τις εκτελέσεις μιας δοκιμής κατά την εμφάνισή της. Η διαχείριση των εκτελέσεων περιλαμβάνει τις εξής λειτουργίες:

- ***εμφάνιση εκτέλεσης δοκιμής*:** ο χρήστης θα μπορεί να βλέπει γενικές πληροφορίες, όπως όνομα δοκιμής, κατάσταση εκτέλεσης (εκκινημένη, σε εξέλιξη, ολοκληρωμένη, σταματημένη, τερματισμένη λόγω λαθών), ημερομηνία έναρξης εκτέλεσης, ημερομηνία λήξης εκτέλεσης(εφόσον η εκτέλεση έχει ολοκληρωθεί ή παυθεί από τον χρήστη) καθώς και τα αποτελέσματα της εκτέλεσης (ολοκληρωμένα εφόσον αυτή έχει ολοκληρωθεί ή ορισμένα/εν μέρει εφόσον αυτή έχει παυθεί από τον χρήστη). Επίσης, θα μπορεί να εκφορτώνει

τα αποτελέσματα της εκτέλεσης (δηλ. την αναφορά) σε μορφή PDF στο τοπικό του σύστημα.

- ***διαγραφή εκτέλεσης δοκιμής*:** ο χρήστης θα μπορεί να διαγράφει μια εκτέλεση δοκιμής και άρα και όλα τα δεδομένα που σχετίζονται με αυτήν.

- ***αναζήτηση εκτέλεσης δοκιμής*:** ο χρήστης θα μπορεί να αναζητά μια εκτέλεση δοκιμής με βάση την κατάστασή της. Επίσης, εκτός από την κατάσταση, θα μπορεί να επιτρέπεται προαιρετικά να παρέχεται ένα εύρος ημερομηνιών που θα πρέπει να καλύπτει είτε η ημερομηνία έναρξής ή λήξης της (αυτό θα μπορούσε να το επιλέγει ο χρήστης)

3.2 Μη Λειτουργικές Απαιτήσεις

- Κάθε λειτουργία δεν μπορεί να παίρνει πάνω από 5 δευτερόλεπτα. Υπενθυμίζεται πως η εκτέλεση δοκιμής πραγματοποιείται ασύγχρονα οπότε ο χρήστης απλώς μπορεί να αιτηθεί την έναρξη της εκτέλεσης, η οποία θα πρέπει να ικανοποιεί τον προαναφερόμενο περιορισμό και όχι η ίδια η αυτούσια/κύρια εκτέλεση της δοκιμής.

- Θα πρέπει να υποστηρίζονται τουλάχιστον 50 ταυτόχρονοι χρήστες

- Θα πρέπει να πραγματοποιηθεί δοχειοποίηση (containerization) της υπηρεσίας προς ανάπτυξη

- Κατάλληλος χειρισμός λαθών/εξαιρέσεων με παροχή αυτο-επεξηγούμενων μηνυμάτων

- Κατάλληλη επικύρωση δεδομένων

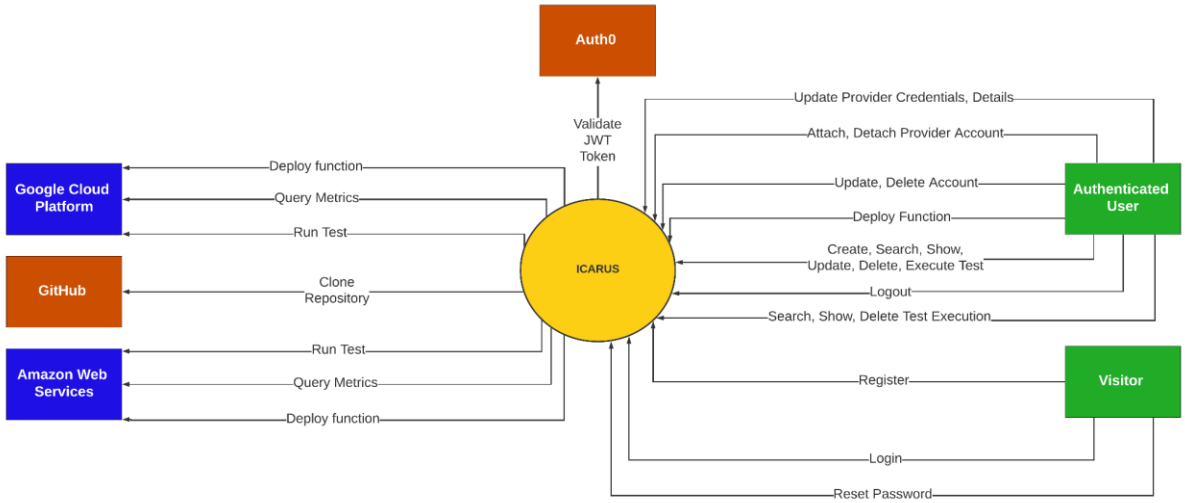
- Καλό επίπεδο ασφάλειας (ταυτοποίηση με βάση token, RBAC/ABAC, υποστήριξη TLS, προστασία CSRF, κατακερματισμός κωδικών)

3.3 Σχεδίαση

Κατά την σχεδίαση του συστήματος, παράχθηκαν ορισμένα μοντέλα / διαγράμματα σχεδίασης, τα οποία προσδιορίζουν διάφορες πτυχές της δομής και συμπεριφοράς της εφαρμογής υπό ανάπτυξη καθώς και οδήγησαν την υλοποίησή της.

3.3.1 Διάγραμμα Περιβάλλοντος

Ένα διάγραμμα περιβάλλοντος (context diagram) ορίζει το όριο μεταξύ του συστήματος υπό ανάπτυξη ή μέρους του συστήματος αυτού και του περιβάλλοντός του, δείχνοντας τις οντότητες που αλληλεπιδρούν με αυτό. Το σύστημα υπό εξέταση βρίσκεται στο κέντρο του διαγράμματος.



Εικόνα 1 - Διάγραμμα περιβάλλοντος

Εξωτερικές οντότητες της εφαρμογής:

- **Google Cloud Platform**
  Η πλατφόρμα Google Cloud Functions παρέχεται ως μέρος των υπηρεσιών του Google Cloud Platform που προσφέρει η Google

- **Amazon Web Services**
  Η πλατφόρμα AWS Lambda παρέχεται ως μέρος των υπηρεσιών του Amazon Web Services που προσφέρει η Amazon

- **Github**
  Ο χρήστης μπορεί να προσφέρει ένα δημόσιο αποθετήριο στο Github και όχι τον πηγαίο κώδικα της συνάρτησης άμεσα. Οπότε, το σύστημα θα πρέπει να εκφορτώσει τον κώδικα αυτό ώστε να μπορεί να το διατάσσει έπειτα σε κάποια FaaS πλατφόρμα.

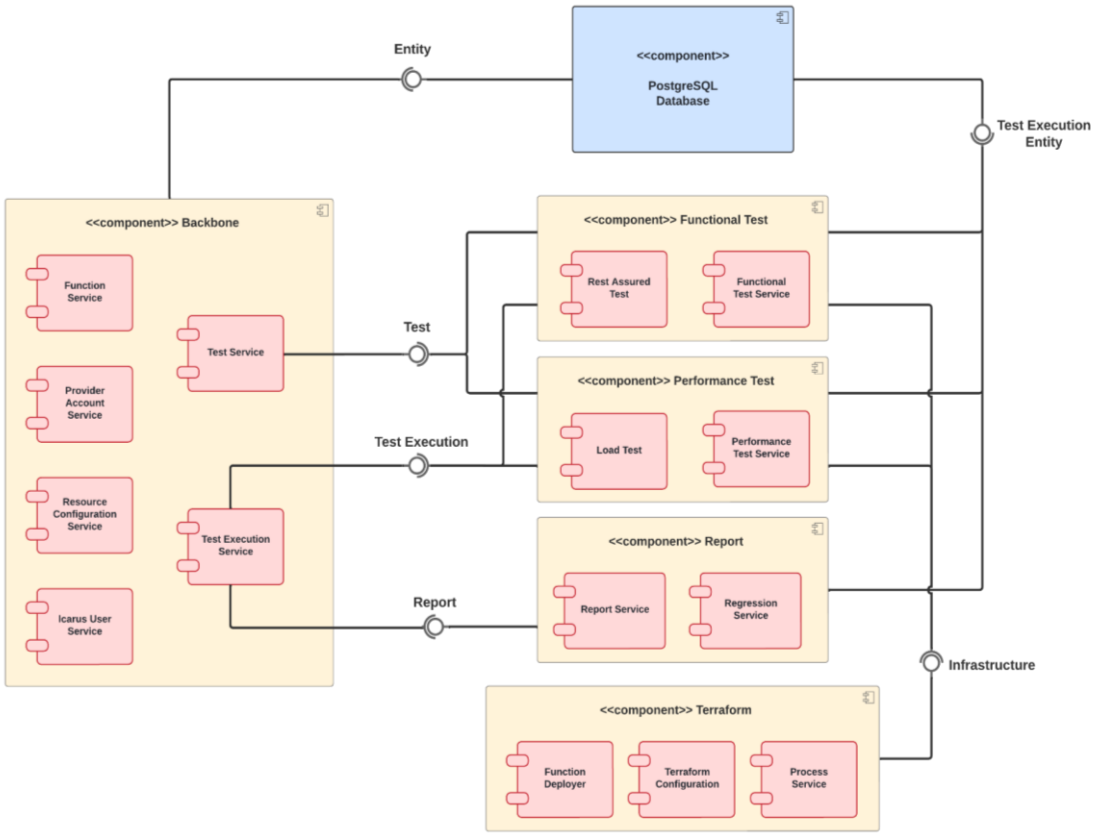
- **Auth0**
  Για την αυθεντικοποίηση με χρήση OAuth2 χρησιμοποιείται ο πάροχος Auth0

- **Authenticated User**
  Ένας εγγεγραμμένος στο σύστημα χρήστης, ο οποίος έχει αυθεντικοποιηθεί είτε με χρήση HTTP Basic είτε με χρήση Auth0

- **Visitor**
  Ένας χρήστης του συστήματος που δεν έχει εγγραφεί ακόμα στην εφαρμογή οπότε την χρησιμοποιεί ως επισκέπτης

3.3.2 Διάγραμμα Συστατικών

Ένα διάγραμμα συστατικών (component diagram) βοηθά στη μοντελοποίηση των συστατικών ενός συστήματος. Ειδικότερα, οπτικοποιεί την οργάνωση και την σύνδεση των συστατικών στο σύστημα. Κάθε συστατικό είναι ενθυλακωμένο και αλληλεπιδρά με άλλα συστατικά μέσω διεπαφών, διασφαλίζοντας έτσι ότι κάθε συστατικό είναι ένα μέρος μιας ενιαίας ολότητας, δηλ. του συστήματος.



Εικόνα 2 - Το διάγραμμα συστατικών του συστήματος

Συστατικά:

- **Backbone**
  Το κύριο συστατικό της εφαρμογής που παρέχει τα βασικά συστατικά, τα οποία είναι απαραίτητα για την εκτέλεση δοκιμών

    - **Function Service**
      Διαχειρίζεται τις οντότητες των Συναρτήσεων

    - **Provider Account Service**
      Διαχειρίζεται τους λογαριασμούς που έχουν οι χρήστες σε παρόχους νέφους

    - **Resource Configuration Service**
      Διαχειρίζεται τις παραμετροποιήσεις των συναρτήσεων

    - **Icarus User Service**
      Διαχειρίζεται τους λογαριασμούς των χρηστών στην εφαρμογή

    - **Test Service**
      Διαχειρίζεται βασικές λειτουργίες των δοκιμών που είναι κοινές τόσο στην περίπτωση της λειτουργικής δοκιμής όσο και στην περίπτωση της μη-λειτουργικής δοκιμής (πχ έλεγχος εάν ο πηγαίος κώδικας της συνάρτησης υπάρχει πριν την εκτέλεση της δοκιμής)

    - **Test Execution Service**
      Παρέχει βασικές λειτουργίες για την εκτέλεση των δοκιμών (πχ παραγωγή αναφοράς, διαγραφή των συναρτήσεων μετά την εκτέλεση των δοκιμών). Χρησιμοποιείται από τα συστατικά Functional και Performance Test για την επιτυχή περάτωση των δοκιμών.

- **PostgreSQL Database**
  Η βάση δεδομένων που χρησιμοποιείται για αποθήκευση των οντοτήτων και των αποτελεσμάτων των δοκιμών

- **Functional Test**
  Διαχειρίζεται την εκτέλεση λειτουργικών δοκιμών.

    - **Rest Assured Test**
      Παραμετροποιεί μια δοκιμή χρησιμοποιώντας το εργαλείο RestAssured

- **Functional Test Service**
  Υπεύθυνο για την ομαλή εκτέλεση μιας λειτουργικής δοκιμής με την χρήση του RestAssured

- **Performance Test**
  Διαχειρίζεται την εκτέλεση μη-λειτουργικών δοκιμών

  - **Load Test**
    Παραμετροποιεί μια μη-λειτουργική δοκιμή χρησιμοποιώντας το εργαλείο JMeter

  - **Performance Test Service**
    Υπεύθυνο για την ομαλή εκτέλεση μιας μη-λειτουργικής δοκιμής με την χρήση του JMeter

- **Report**
  Διαχειρίζεται την παραγωγή της τελικής αναφοράς με τα αποτελέσματα της εκάστοτε δοκιμής.

  - **Report Service**
    Υπεύθυνο για την δημιουργία της τελικής αναφοράς

  - **Regression Service**
    Υπεύθυνο για την εκτέλεση μιας γραμμικής παλινδρόμησης (linear regression), τα αποτελέσματα της οποίας προστίθενται στην τελική αναφορά (μόνο σε περίπτωση μη-λειτουργικής δοκιμής)

- **Terraform**
  Διαχειρίζεται την διάταξη των συναρτήσεων στους παρόχους νέφους με την χρήση του Terraform και την παραγωγή εγγραφών διατάξεων (Deployment Records), όπου ένα Deployment Record αντιστοιχίζει την διάταξη μιας συγκεκριμένης παραμετροποίησης της συνάρτησης σε ένα URL όπου αυτή είναι προσβάσιμη (πχ 128MB - 2 vCPU cores - Canada - http://example.com)

  - **Function Deployer**
    Διαχειρίζεται την διάταξη στους παρόχους και την παραγωγή deployment records

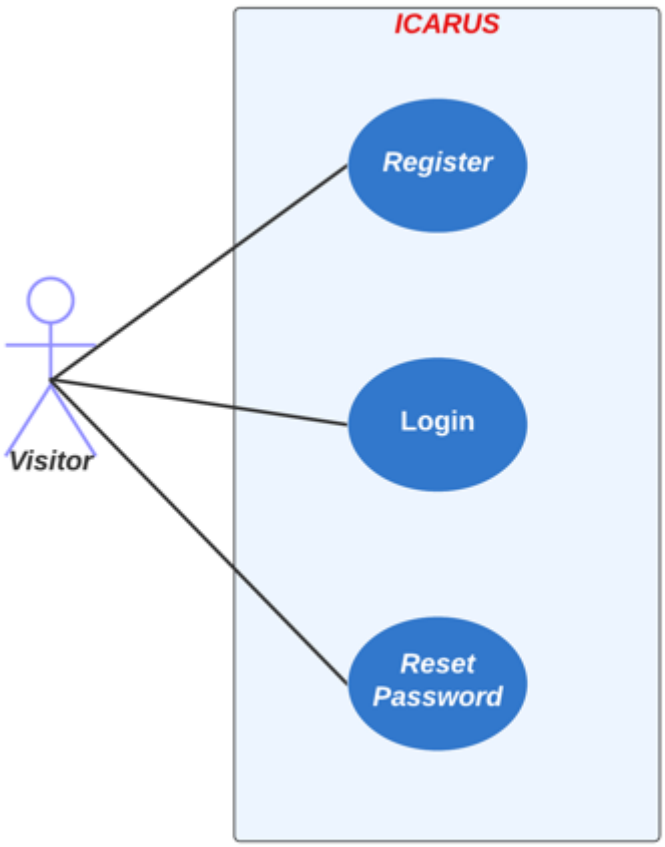
  - **Terraform Configuration**
    Παραμετροποιεί το εργαλείο Terraform

- **Process Service**
  Εκτελεί εντολές του εργαλείου Terraform στο σύστημα ή στο δοχείο που περιέχει την εφαρμογή

3.3.3 Διαγράμματα Περιπτώσεων Χρήσης

Ένα διάγραμμα χρήσης αναπαριστά τον τρόπο με τον οποίο μπορεί να αλληλεπιδράσει ένας χρήστης με ένα σύστημα και τις βασικές λειτουργίες (συστήματος) που μπορεί να εκτελέσει.

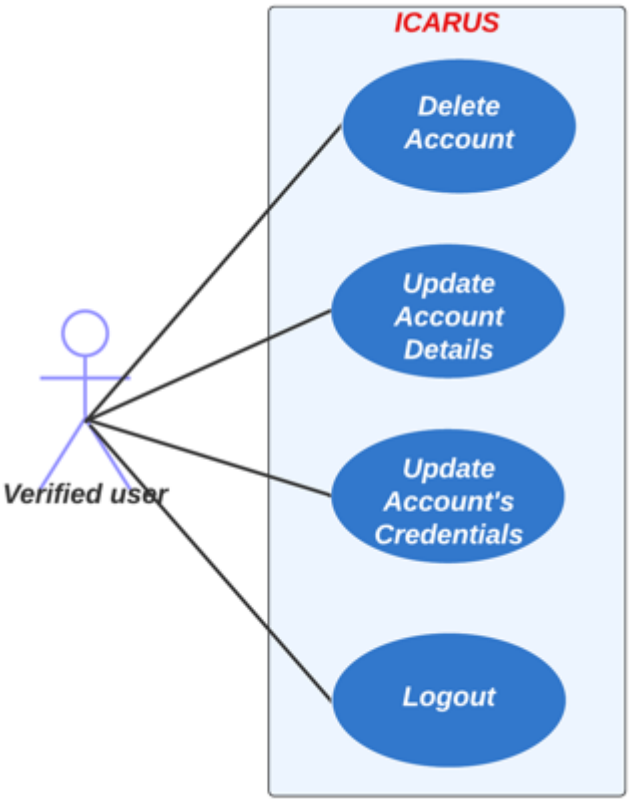
- **Περιπτώσεις χρήσης ενός επισκέπτη**



Εικόνα 3 - Διάγραμμα περιπτώσεων χρήσης - Επισκέπτης

Ένας επισκέπτης, μπορεί είτε να εγγραφεί στην εφαρμογή, είτε να πραγματοποιήσει είσοδο (εφόσον είναι ήδη εγγεγραμμένος) χρησιμοποιώντας Auth0 ή HTTP Basic είτε να επαναφέρει τον κωδικό του (εφόσον είναι εγγεγραμμένος αλλά δεν μπορεί να εισέλθει εξαιτίας ενός χαμένου κωδικού πρόσβασης).

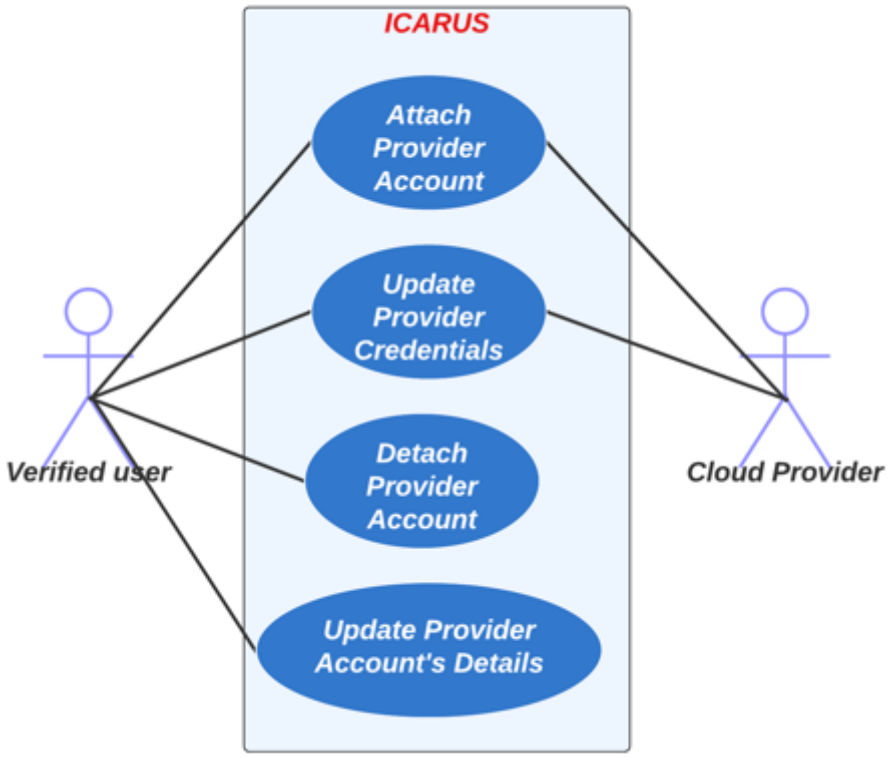
- **Περιπτώσεις χρήσης για διαχείριση λογαριασμών χρήστη**



Εικόνα 4 - Διάγραμμα περιπτώσεων χρήσης - Διαχείριση λογαριασμών

Ένας εγγεγραμμένος χρήστης μπορεί είτε να διαγράψει τον λογαριασμό του, είτε να ενημερώσει τα διαπιστευτήρια του και τις λεπτομέρειες του λογαριασμού του είτε να πραγματοποιήσει έξοδο από την εφαρμογή (logout) (Στην περίπτωση χρήσης HTTP Basic δεν υποστηρίζεται αυτή η λειτουργία ενώ στην περίπτωση που χρησιμοποιείτε Auth0 η έξοδος πραγματοποιείται αυτόματα με την λήξη του JWT Token).

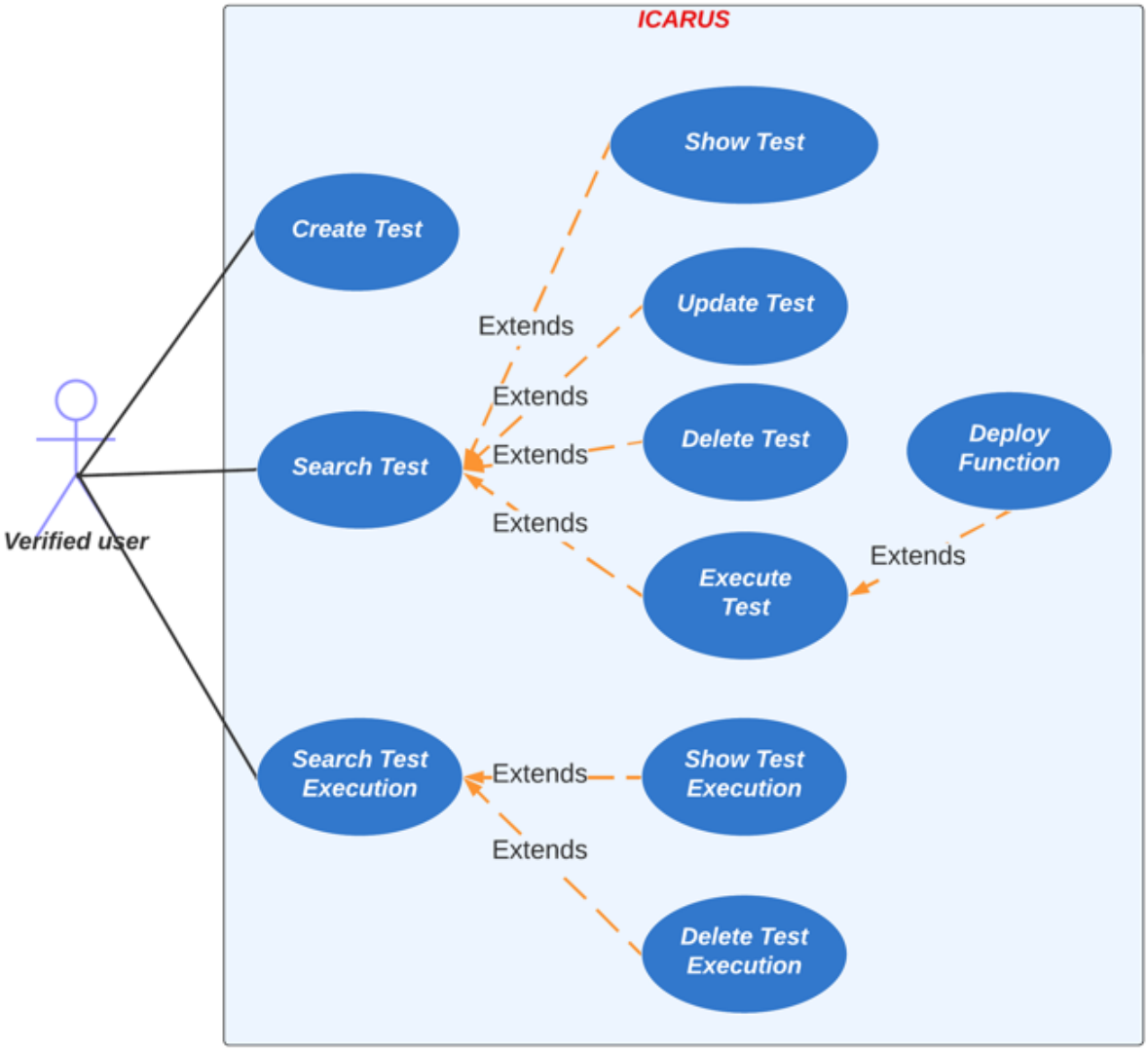
- **Περιπτώσεις χρήσης για διαχείριση συνδεδεμένων λογαριασμών παρόχου νέφους**



Εικόνα 5 - Διάγραμμα περιπτώσεων χρήσης - Διαχείριση λογαριασμού παρόχου νέφους

Ένας εγγεγραμμένος χρήστης μπορεί να συνδέσει με την εφαρμογή έναν λογαριασμό παρόχου νέφους. Από την στιγμή που η εφαρμογή μας, ο Ίκαρος, συνδέσει τον χρήστη, αποθηκεύει τα διαπιστευτήρια που παρέχει ο χρήστης για αυτόν τον λογαριασμό καθώς και μερικές λεπτομέρειες όπως το όνομα του λογαριασμού. Ο χρήστης μπορεί αργότερα να ενημερώσει τα διαπιστευτήρια του λογαριασμού νέφους που έχει αποθηκεύσει ο Ίκαρος (σε περίπτωση που δεν είναι πλέον έγκυρα), να ενημερώσει τις λεπτομέρειες του λογαριασμού στην εφαρμογή (όπως να αλλάξει για παράδειγμα το όνομα του λογαριασμού) ή να αποσυνδέσει τον λογαριασμό από την εφαρμογή και ταυτόχρονα να διαγράψει όλα τα άνωθεν.

- **Περιπτώσεις χρήσης για την βασική λειτουργικότητα της εφαρμογής**



Εικόνα 6 - Διάγραμμα περιπτώσεων χρήσης - Βασικές λειτουργίες Ίκαρου

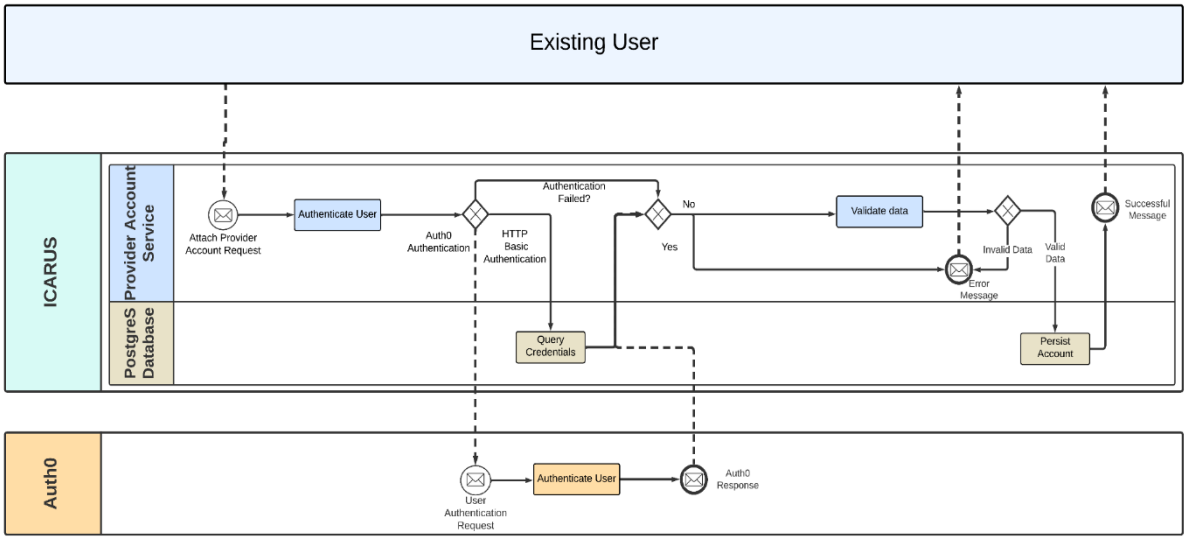
Ένας εγγεγραμμένος χρήστης μπορεί να δημιουργήσει δοκιμές, να τις εκτελέσει και να δει τα αποτελέσματα τους. Για την εκτέλεση μιας δοκιμής απαιτείται πρώτα το μοναδικό αναγνωριστικό της, συνεπώς ο χρήστης πρέπει πρώτα να αναζητήσει την συγκεκριμένη δοκιμή. Το ίδιο απαιτείται και για να δει τα αποτελέσματα της δοκιμής. Αντίστοιχα ο χρήστης μπορεί να διαγράψει, να ενημερώσει ή να δει την δοκιμή αλλά και να διαγράψει τα αποτελέσματα της (για όλες τις παραπάνω λειτουργίες απαιτείται πάλι το αναγνωριστικό της δοκιμής συνεπώς προηγείται η αναζήτηση της).

3.3.4 Διαγράμματα Επιχειρηματικών Διαδικασιών

Ένα διάγραμμα επιχειρηματικών διαδικασιών αναπαριστά μια επιχειρηματική διαδικασία με εύκολο και κατανοητό τρόπο δείχνοντας την ροή εκτέλεσης και δεδομένων της διαδικασίας καθώς και τους ρόλους/χρήστες/συστήματα που εμπλέκονται σε αυτήν.

Παρακάτω δίνονται τα διαγράμματα των πιο χαρακτηριστικών διαδικασιών:

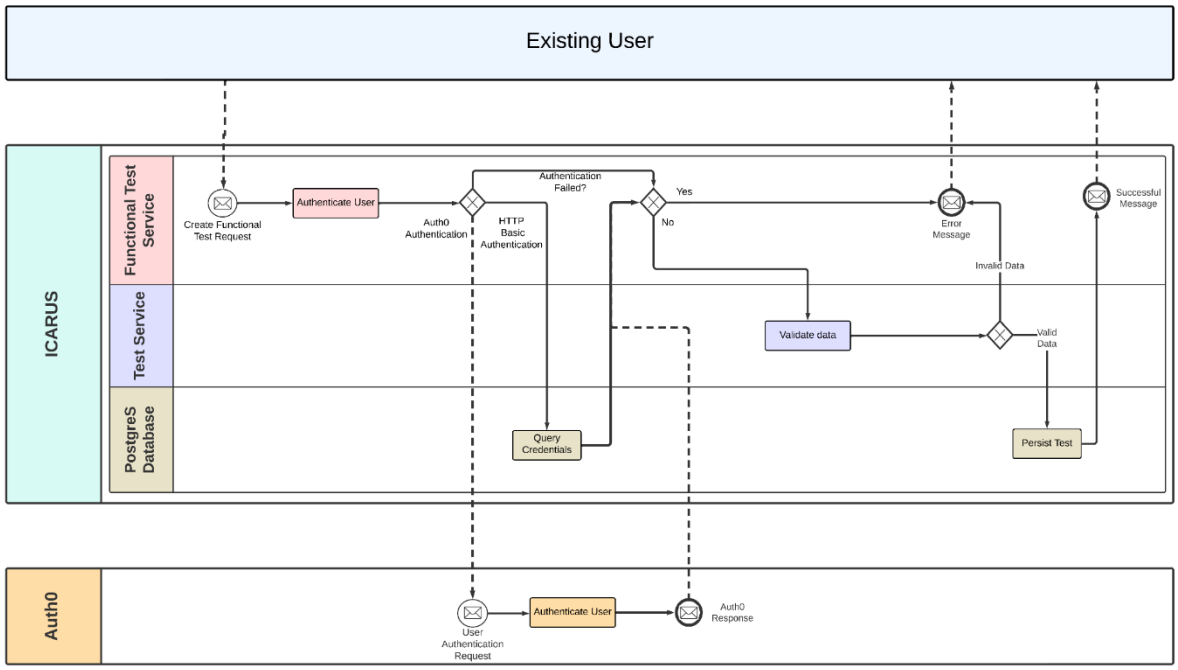
- **Διαδικασία σύνδεσης λογαριασμού παρόχου νέφους**



Εικόνα 7 - Διάγραμμα επιχειρηματικών διαδικασιών - Σύνδεση λογαριασμού παρόχου νέφους

Η διαδικασία αυτή πραγματοποιείται μόνο από εγγεγραμμένους χρήστες. Προτού ξεκινήσει η διαδικασία σύνδεσης λογαριασμού νέφους, ο χρήστης πρέπει να αυθεντικοποιηθεί από το σύστημα, είτε χρησιμοποιώντας HTTP Basic είτε χρησιμοποιώντας Auth0. Από εκεί και πέρα, εάν η αυθεντικοποίηση ήταν επιτυχής, τότε το συστατικό Provider Account Service ελέγχει εάν τα δεδομένα του νέου λογαριασμού είναι έγκυρα και σε περίπτωση που όντως είναι, τότε ο νέος λογαριασμός αποθηκεύεται στην βάση και συνδέεται με τον χρήστη. Η διαδικασία αυτή είναι η ίδια τόσο για λογαριασμούς AWS όσο και για λογαριασμούς GCP.

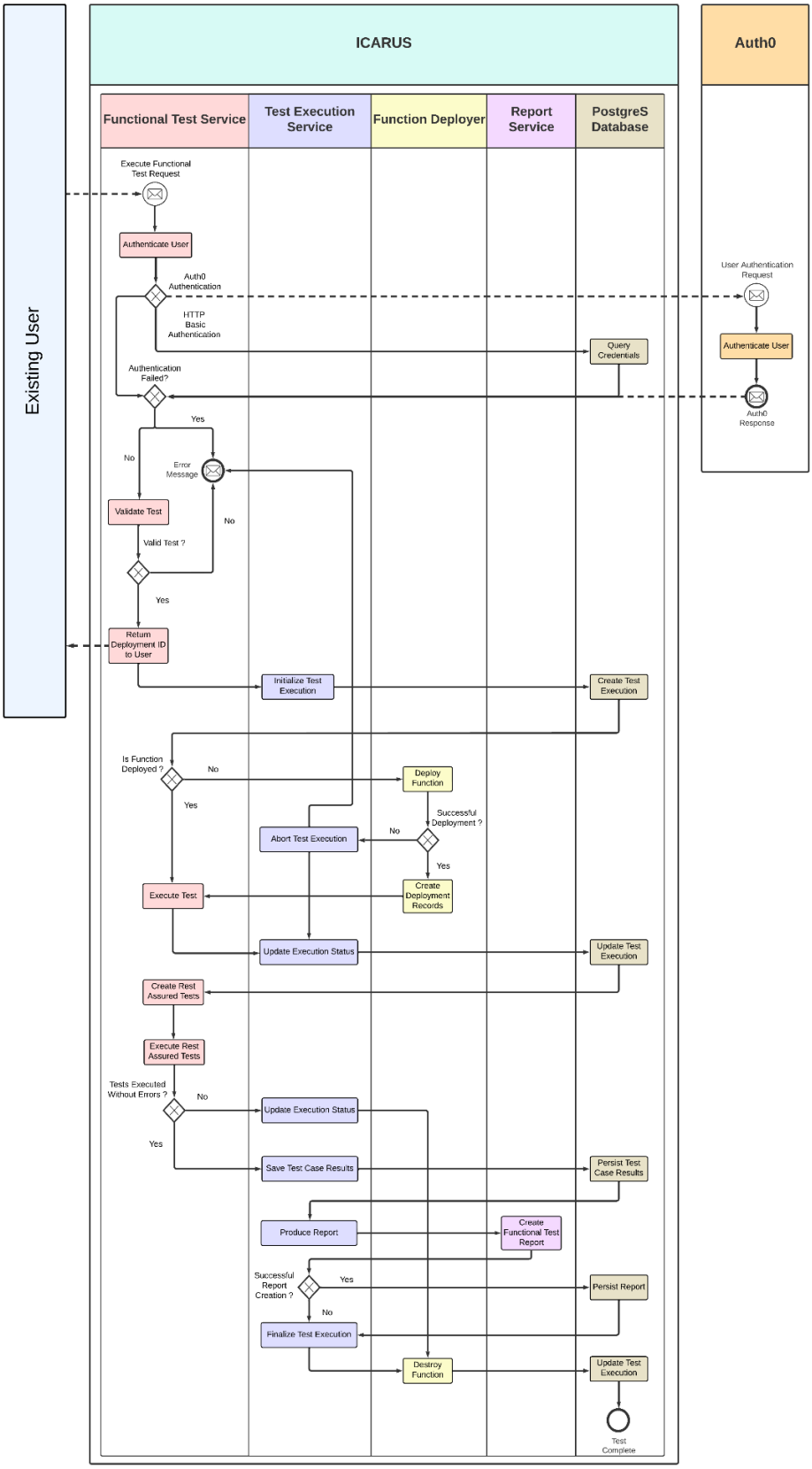
- **Διαδικασία δημιουργίας δοκιμής**



Εικόνα 8 - Διάγραμμα επιχειρηματικών διαδικασιών - Δημιουργία λειτουργικής δοκιμής

Η διαδικασία αυτή πραγματοποιείται μόνο από εγγεγραμμένους χρήστες. Αρχικά, ο χρήστης πρέπει να αυθεντικοποιηθεί από τον Ίκαρο, όπως παραπάνω. Έπειτα, το συστατικό Functional Test Service χρησιμοποιεί το συστατικό Test Service για να επικυρώσει την εγκυρότητα των δεδομένων της καινούργιας δοκιμής και σε περίπτωση που τα δεδομένα της νέας δοκιμής είναι έγκυρα τότε θα αποθηκεύσει την καινούργια δοκιμή στην βάση δεδομένων. Η ίδια διαδικασία χρησιμοποιείται και στην περίπτωση δημιουργίας μιας δοκιμής απόδοσης με μόνη διαφορά την χρήση του συστατικού Performance Test Service έναντι του Functional Test Service.

- **Διαδικασία εκτέλεσης δοκιμής**
  - **Λειτουργική Δοκιμή**



Εικόνα 9 - Διάγραμμα επιχειρηματικών διαδικασιών - Εκτέλεση λειτουργικής δοκιμής

Η διαδικασία αυτή πραγματοποιείται μόνο από εγγεγραμμένους χρήστες. Αρχικά, ο χρήστης αυθεντικοποιείται, όπως παραπάνω, και στην συνέχεια επικυρώνεται πως τα δεδομένα της δοκιμής είναι έγκυρα προτού αυτή εκτελεστεί (στο στάδιο αυτό γίνονται έλεγχοι όπως: ύπαρξη του πηγαίου κώδικα της συνάρτησης, ύπαρξη τουλάχιστον ενός συνδεδεμένου λογαριασμού κλπ).

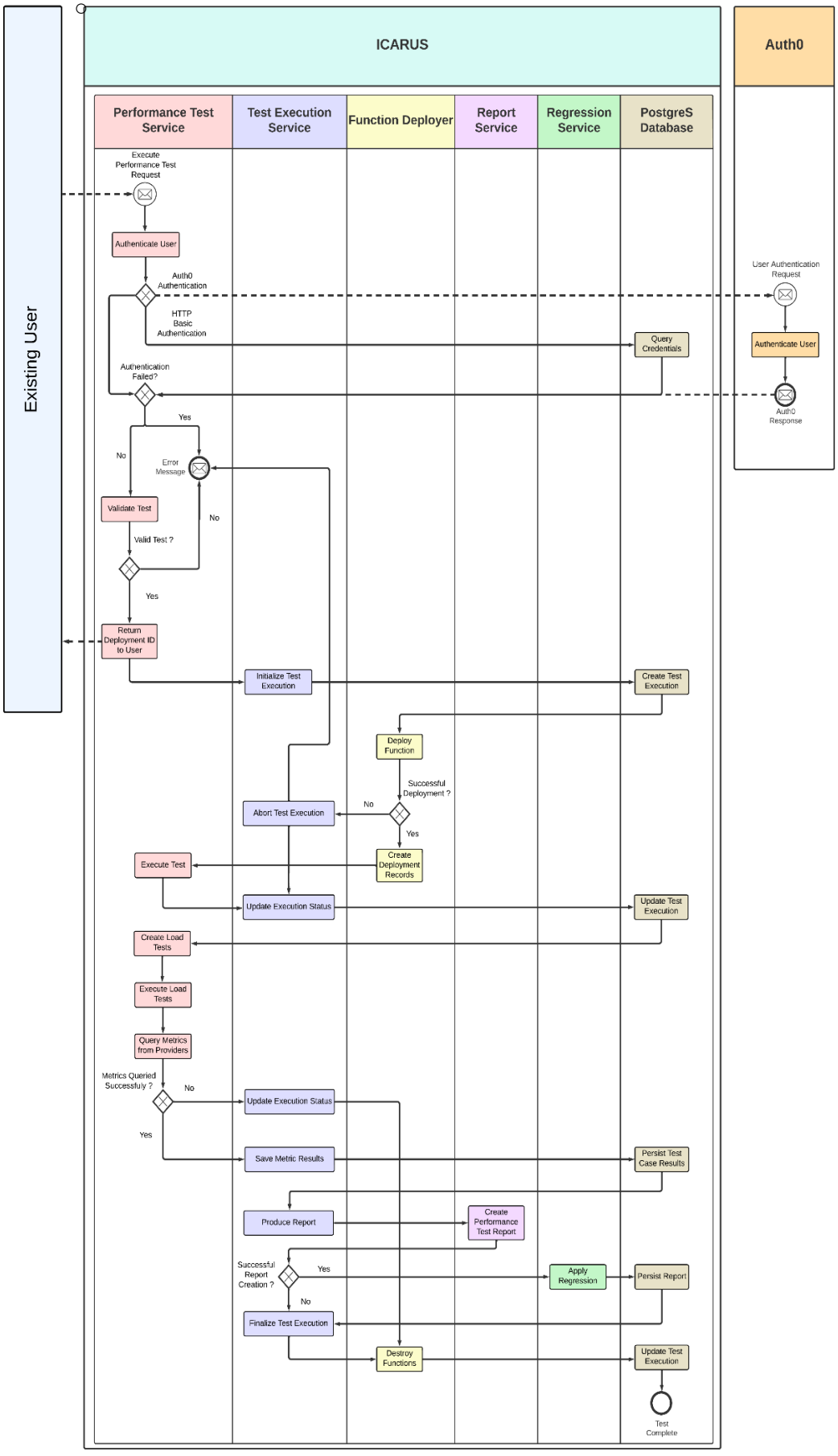
Στην περίπτωση όπου η δοκιμή είναι έγκυρη, τότε επιστρέφεται στον χρήστη ένα αναγνωριστικό που ονομάζεται deployment id και συμβολίζει την συγκεκριμένη εκτέλεση της δοκιμής (για διαφορετικές εκτελέσεις της ίδιας δοκιμής δημιουργούνται νέα deployment id).

Στην συνέχεια ξεκινάει η ασύγχρονη εκτέλεση της δοκιμής, η οποία περιλαμβάνει αρχικά την δημιουργία ενός Test Execution (οντότητα που συμβολίζει την εκτέλεση της δοκιμής) στην βάση δεδομένων και έπειτα τον έλεγχο εάν η συνάρτηση έχει ήδη διαταχθεί στον πάροχο από τον χρήστη (μόνο οι λειτουργικές δοκιμές υποστηρίζουν συναρτήσεις που έχουν διαταχθεί ήδη από τον χρήστη σε κάποιον πάροχο). Εάν η συνάρτηση δεν έχει διαταχθεί σε κάποιον πάροχο, τότε το συστατικό Function Deployer αναλαμβάνει την διάταξη. Σε περίπτωση επιτυχίας διάταξης, ενημερώνονται οι πληροφορίες της εκτέλεσης δοκιμής στην βάση και το σύστημα προχωράει στην δημιουργία και την εκτέλεση των αντίστοιχων λειτουργικών δοκιμών.

Εάν οι δοκιμές εκτελεστούν με επιτυχία, τότε το σύστημα προχωράει στην παραγωγή της αναφοράς. Εφόσον υπάρξει κάποιο σφάλμα κατά την εκτέλεση των δοκιμών, τότε ενημερώνονται οι πληροφορίες της εκτέλεσης δοκιμής (Test Execution) στην βάση και καταστρέφεται η συνάρτηση στον πάροχο.

Εάν η αναφορά δημιουργηθεί με επιτυχία, τότε το σύστημα ενημερώνει την κατάσταση της εκτέλεσης δοκιμής (Test Execution), αποθηκεύει την αναφορά και καταστρέφει την συνάρτηση. Σε περίπτωση σφάλματος κατά την δημιουργία της αναφοράς, το σύστημα ενημερώνει πάλι τις πληροφορίες της εκτέλεσης δοκιμής στην βάση και καταστρέφει την συνάρτηση. Απλώς δεν αποθηκεύει την αναφορά στην βάση και επιτρέπει στον χρήστη μεταγενέστερα την εκ νέου προσπάθεια δημιουργίας της αναφοράς χρησιμοποιώντας το deployment id που του έχει ήδη παραδώσει.

- **Δοκιμή Απόδοσης**



Εικόνα 10 - Διάγραμμα επιχειρηματικών διαδικασιών - Εκτέλεση Δοκιμής Απόδοσης

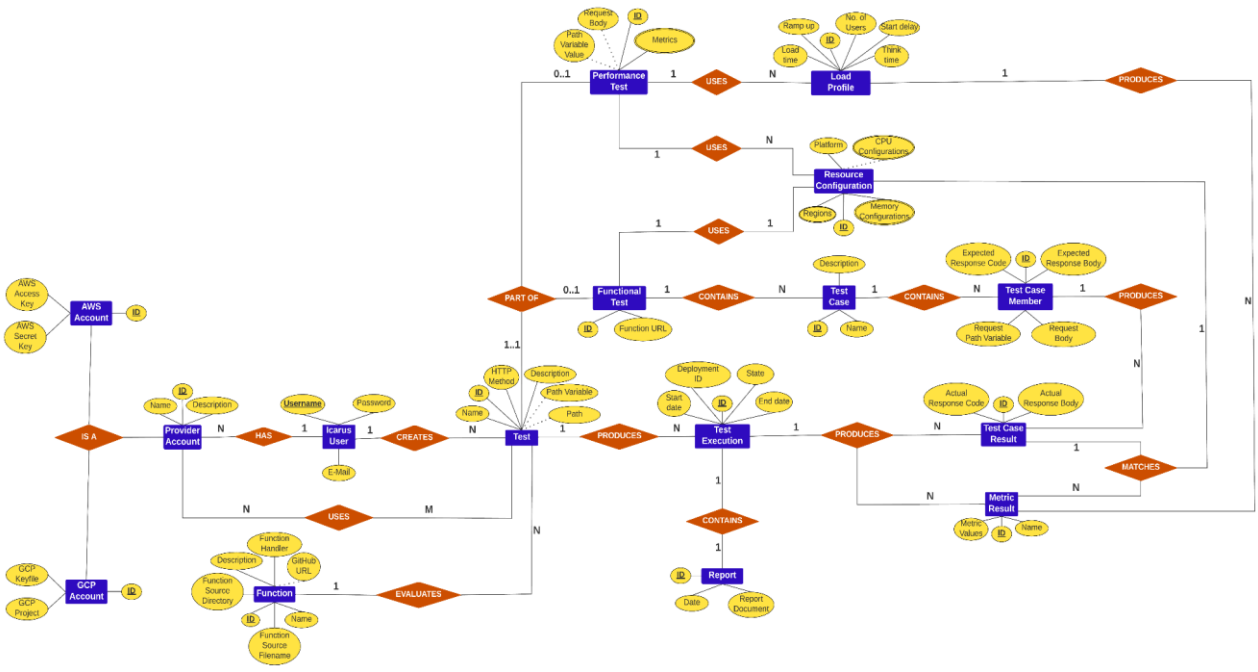
Η εκτέλεση της δοκιμής απόδοσης ακολουθεί ακριβώς την ίδια διαδικασία με την εκτέλεση της λειτουργικής δοκιμής με μικρές διαφορές. Στην θέση του συστατικού Functional Test Service συμμετέχει το συστατικό Performance Test Service ενώ αντί για Rest Assured Tests δημιουργούνται Load Tests οπότε μετά την εκτέλεση τους γίνεται συλλογή των μετρικών από τους παρόχους.

Στην περίπτωση της δοκιμής απόδοσης δεν πραγματοποιείται έλεγχος για το εάν μια συνάρτηση έχει ήδη διαταχθεί από τον χρήστη σε έναν πάροχο διότι ο Ίκαρος αναλαμβάνει την διαδικασία της διάταξης απευθείας.

Ταυτόχρονα, αφού δημιουργηθεί επιτυχώς η αναφορά, ο Ίκαρος εκτελεί μια γραμμική παλινδρόμηση στα αποτελέσματα των μετρικών και προσθέτει το αποτέλεσμά της στην αναφορά. Την παραγωγή της αναφοράς αναλαμβάνει το συστατικό Regression Service

3.3.5 Διάγραμμα Οντοτήτων Συσχετίσεων

Ένα Διάγραμμα Οντοτήτων Συσχετίσεων είναι ένας τύπος διαγράμματος δεδομένων σε λογικό επίπεδο που απεικονίζει τις βασικές οντότητες ενός συστήματος (άνθρωποι, αντικείμενα, έννοιες), τα χαρακτηριστικά τους καθώς και τον τρόπο με τον οποίο οι οντότητες σχετίζονται μεταξύ τους.



Εικόνα 11 - Διάγραμμα Οντοτήτων Συσχετίσεων

Το διάγραμμα Οντοτήτων-Συσχετίσεων (Entity-Relationship – ER) απεικονίζει τις ακόλουθες οντότητες και τις σχέσεις τους:

1. **AWS Account (Λογαριασμός AWS)**
    - Ιδιότητες: AWS access key (κλειδί πρόσβασης), AWS secret key (μυστικό κλειδί), id (αναγνωριστικό) (μοναδική, γνώρισμα κλειδί)
    - Σχέσεις:
        - Ένα Aws Account είναι ένα Provider Account (Λογαριασμός σε Πάροχο)

2. **GCP Account (Λογαριασμός GCP)**
    - Ιδιότητες: GCP Keyfile (αρχείο κλειδιού), GCP Project (έργο), id (μοναδική, γνώρισμα κλειδί)
    - Σχέσεις:
        - Ένα GCP Account είναι ένα Provider Account

3. **Provider Account**
    - Ιδιότητες: Name (όνομα), Description (περιγραφή), id (μοναδική, γνώρισμα κλειδί)
    - Σχέσεις:
        - Ένα GCP Account & ένα AWS Account είναι (ISA) Provider Account
        - Πολλά Provider Account ανήκουν σε έναν Icarus User (Χρήστη του Ίκαρου)

4. **Icarus User**
    - Ιδιότητες: Username (όνομα χρήστη) (μοναδική, γνώρισμα κλειδί), Password (κωδικός), E-mail
    - Σχέσεις:
        - Ένας Icarus User δημιουργεί πολλά Provider Account
        - Ένας Icarus User δημιουργεί πολλά Test (δοκιμές)

5. **Test (Δοκιμή)**
    - Ιδιότητες: Name, HTTP Method (μέθοδος HTTP), Description, Path Variable (μεταβλητή περιβάλλοντος) (προαιρετική), Path (προαιρετική), id (μοναδική, γνώρισμα κλειδί)
    - Σχέσεις:
        - Πολλά Test αξιολογούν μια Function (συνάρτηση)
        - Πολλά Test χρησιμοποιούν πολλά Provider Account
        - Ένα Test παράγει πολλές Test Execution (εκτελέσεις δοκιμών)
        - Ένα Functional Test (Λειτουργική Δοκιμή) & ένα Performance Test (Δοκιμή Απόδοσης) είναι Test

6. **Function (Συνάρτηση)**
    - Ιδιότητες: Function Handler (χειριστής συνάρτησης), Description, Function Source Directory (πηγαίος φάκελος συνάρτησης), Function Source Filename (όνομα πηγαίου αρχείων συνάρτησης), Name, Github Url (προαιρετική), id (μοναδική γνώρισμα κλειδί)
    - Σχέσεις:
        - Μια Function αξιολογείται από πολλά Test

7. **Performance Test (Δοκιμή Απόδοσης)**
    - Ιδιότητες: Path Variable Value (τιμή μεταβλητής path) (προαιρετική), Request Body (σώμα του αιτήματος) (προαιρετική), Metrics (μετρικές) (πλειότιμο γνώρισμα)
    - Σχέσεις:
        - Ένα Performance Test χρησιμοποιεί πολλά Load Profiles (Προφίλ Φόρτου)
        - Ένα Performance Test χρησιμοποιεί πολλά Resource Configuration (Διαμόρφωση Πόρου)

8. **Functional Test (Λειτουργική Δοκιμή)**
    - Ιδιότητες: function Url (url της συνάρτησης), id (μοναδική, γνώρισμα κλειδί)
    - Σχέσεις:
        - Ένα Functional Test χρησιμοποιεί ένα Resource Configuration
        - Ένα Functional Test περιέχει πολλά Test Case (Περιπτώσεις Δοκιμής)

9. **Test Execution (Εκτέλεση Δοκιμής)**
    - Ιδιότητες: Start Date (Ημερομηνία έναρξης, Deployment Id (αναγνωριστικό διάταξης), id (μοναδική, γνώρισμα κλειδί), State (κατάσταση), End Date (ημερομηνία λήξης)
    - Σχέσεις:
        - Ένα Test Execution περιλαμβάνει ένα Report (Αναφορά)
        - Ένα Test Execution παράγει πολλά Test Case Result (Αποτελέσματα περιπτώσεων Δοκιμής)
        - Ένα Test Execution παράγει πολλά Metric Result (Αποτελέσματα Μετρικών)

10. **Report (Αναφορά)**
    - Ιδιότητες: id (μοναδική, γνώρισμα κλειδί), Date (ημερομηνία), Report Document (έγγραφο αναφοράς)

11. **Load Profile (Προφίλ Φόρτου)**
    - Ιδιότητες: Load Time (χρόνος φόρτου), Ramp Up (περίοδος κλιμάκωσης), id (μοναδική, γνώρισμα κλειδί), No. of Users (αριθμός

χρηστών), Start Delay (αρχική καθυστέρηση), Think Time (χρόνος απόφασης)
- Σχέσεις:
    - Ένα Load Profile παράγει πολλά Metric Result (Αποτελέσματα Μετρικών)

12. Resource Configuration (Διαμόρφωση Πόρου)

- Ιδιότητες: Platform (πλατφόρμα), CPU Configurations (διαμορφώσεις CPU)(πλειότιμη, προαιρετική), Regions (τοποθεσίες)(πλειότιμη), id (μοναδική, γνώρισμα κλειδί), Memory Configurations (διαμορφώσεις μνήμης)(πλειότιμη)
- Σχέσεις:
    - Ένα Resource Configuration αντιστοιχίζεται σε ένα Test Case Result (Αποτέλεσμα Περίπτωσης Δοκιμής)
    - Ένα Resource Configuration αντιστοιχίζεται σε πολλά Metric Result (Αποτελέσματα Μετρικών)

13. Test Case (Περίπτωση Δοκιμής)

- Ιδιότητες: id (μοναδική, γνώρισμα κλειδί), name (όνομα), description (περιγραφή)
- Σχέσεις:
    - Ένα Test Case περιλαμβάνει πολλά Test Case Member (Μέλος Περίπτωσης Δοκιμής)

14. Test Case Member (Μέλος Περίπτωσης Δοκιμής)

- Ιδιότητες: Expected Response Code (αναμενόμενος κωδικός της απάντησης), id (μοναδική, γνώρισμα κλειδί), Expected Response Body (αναμενόμενο σώμα της απάντησης), Request Path Variable (μεταβλητή path του σταλθέντος αιτήματος), Request Body (σώμα του σταλθέντος αιτήματος)
- Σχέσεις:
    - Ένα Test Case Member παράγει πολλά Test Case Result (Αποτέλεσμα Περίπτωσης Δοκιμής)

15. Test Case Result (Αποτέλεσμα Περίπτωσης Δοκιμής)

- Ιδιότητες: Actual Response Code (πραγματικός κωδικός της απάντησης), Actual Response Body (πραγματικό σώμα της απάντησης), id (μοναδική, γνώρισμα κλειδί)

16. Metric Result (Αποτέλεσμα Μετρικής)

- Ιδιότητες: Metric Values (τιμές της μετρικής), id (μοναδική, γνώρισμα κλειδί), Name (όνομα της μετρικής)

3.4 Υλοποίηση

Η εφαρμογή υλοποιήθηκε με πλήρη διαφάνεια και ακολουθώντας τις αρχές του ανοιχτού λογισμικού (open source software). Είναι διαθέσιμη ως λογισμικό ανοιχτού κώδικα στο παρακάτω αποθετήριο GitHub:

https://github.com/ttomtsis/icarus

3.4.1 Αρχιτεκτονική

Η αρχιτεκτονική με την οποία υλοποιήθηκε η εφαρμογή είναι η N – tier αρχιτεκτονική ή διαστρωματική (layered) αρχιτεκτονική. Είναι ένα μοτίβο αρχιτεκτονικής λογισμικού στο οποίο μια εφαρμογή χωρίζεται σε ξεχωριστά στρώματα/επίπεδα και κάθε ένα εκτελεί μια συγκεκριμένη λειτουργία, η οποία μπορεί να καταναλώνεται από το αμέσως επόμενο επίπεδο (στην ιεραρχία της αρχιτεκτονικής). Κάθε στρώμα μπορεί να επικοινωνήσει μόνο με τα στρώματα ακριβώς πάνω και κάτω από αυτό.

Τα επίπεδα/στρώματα σε μια τυπική N – tier αρχιτεκτονική είναι:

- Presentation Layer (Στρώμα Παρουσίασης): Το ανώτερο επίπεδο που αλληλεπιδρά με τον τελικό χρήστη. Χειρίζεται τη λογική παρουσίασης της εφαρμογής και επικοινωνεί με το service layer (στρώμα υπηρεσιών) για ανάκτηση ή ενημέρωση δεδομένων.

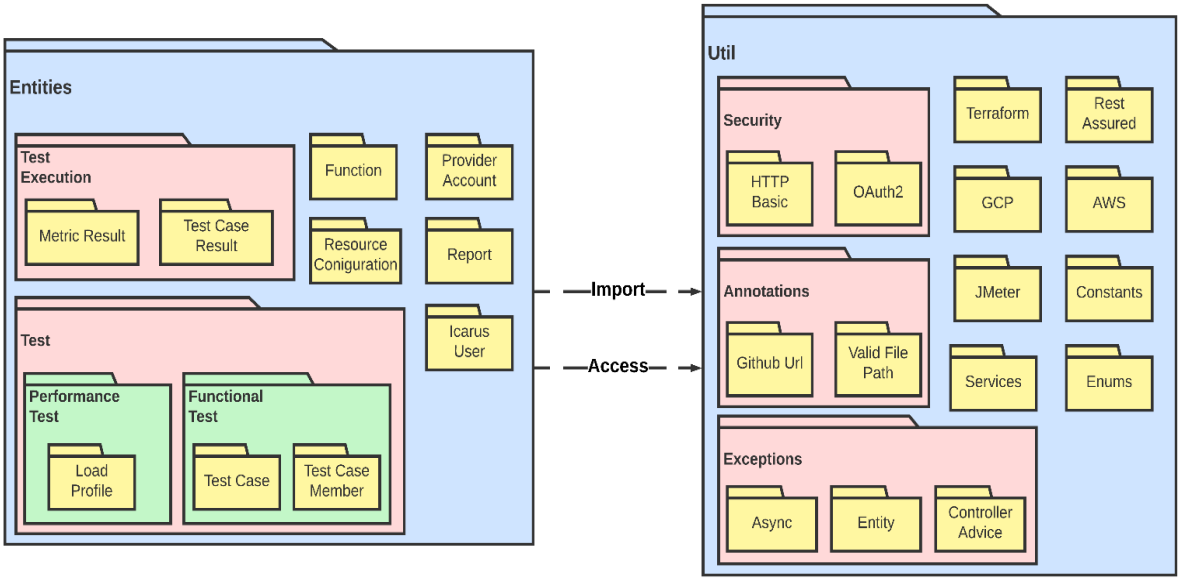
- Service Layer (Στρώμα Υπηρεσιών): Το μεσαίο στρώμα που χειρίζεται τη λογική της εφαρμογής. Λαμβάνει δεδομένα από το επίπεδο παρουσίασης, εκτελεί επιχειρηματική λογική (business logic) και αλληλεπιδρά με το data layer (στρώμα δεδομένων) για ανάκτηση ή ενημέρωση δεδομένων.

- Data Layer (Στρώμα Δεδομένων): Το κατώτατο επίπεδο που χειρίζεται τη διατήρηση δεδομένων. Συνήθως υλοποιείται μέσω της χρήσης ενός συστήματος διαχείρισης βάσεων δεδομένων που είναι υπεύθυνο για την αποθήκευση και την ανάκτηση δεδομένων στις βάσεις δεδομένων που διαχειρίζεται.

Διαχωρίζοντας την εφαρμογή σε ξεχωριστά επίπεδα/στρώματα, η αρχιτεκτονική παρέχει πλεονεκτήματα, όπως επεκτασιμότητα και δυνατότητα συντήρησης. Οι αλλαγές σε ένα επίπεδο έχουν ελάχιστο αντίκτυπο στα άλλα επίπεδα, καθιστώντας ευκολότερη την τροποποίηση και επέκταση της εφαρμογής.

3.4.2 Βασική Δομή & Λεπτομέρειες Υλοποίησης Κώδικα

Ένα διάγραμμα πακέτου απεικονίζει πώς το σύστημα χωρίζεται σε λογικές ομαδοποιήσεις
χρησιμοποιώντας πακέτα. Τα πακέτα χρησιμοποιούνται για την ομαδοποίηση σχετικών κλάσεων, διεπαφών ή άλλων στοιχείων σε μια ενιαία μονάδα. Με αυτό τον τρόπο, μια τέτοια ομαδοποίηση βοηθά στην οργάνωση και διαχείριση μεγάλων συστημάτων λογισμικού. Η δομή των πακέτων του συστήματός μας απεικονίζεται στην παρακάτω εικόνα.



Εικόνα 12 - Διάγραμμα Πακέτων

Η ανάπτυξη της εφαρμογής πραγματοποιήθηκε χρησιμοποιώντας την γλώσσα προγραμματισμού Java, συγκεκριμένα την έκδοση 21, ενώ το πλαίσιο ανάπτυξης (development framework) ήταν το Spring Boot έκδοση 3.1.5[1]. Το εργαλείο αυτοματοποίησης κατασκευής (build automation tool) που επιλέχθηκε είναι το Maven[2]. Είναι ένα αρκετά δημοφιλές εργαλείο αυτοματοποίησης κατασκευής και διαχείρισης εξαρτήσεων για έργα που βασίζονται σε Java καθώς απλοποιεί τις προαναφερόμενες διαδικασίες, παρέχοντας τη δυνατότητα χειρισμού των κατασκευών του έργου (λογισμικού) και των εξαρτήσεων του μέσω μιας προσέγγισης βασισμένης σε μοντέλα διαμόρφωσης κατασκευής.

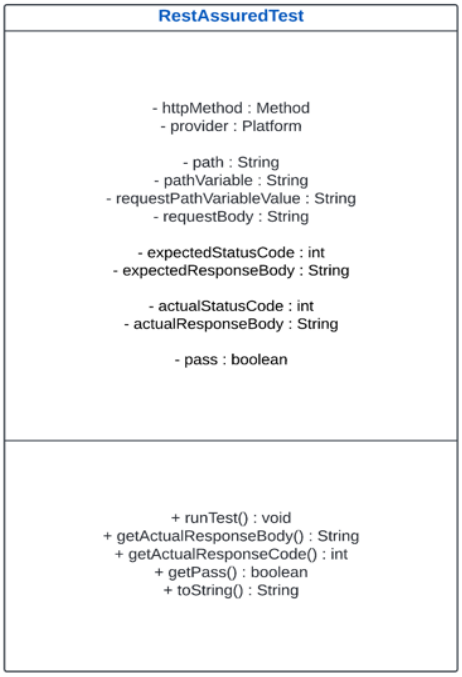
Για τη βάση δεδομένων χρησιμοποιήθηκε η PostgreSQL[3], έκδοση 15.5, που είναι ένα δωρεάν και ανοιχτού κώδικα σύστημα διαχείρισης σχεσιακών βάσεων

[1] Spring Boot - https://spring.io/projects/spring-boot
[2] Maven - https://maven.apache.org/
[3] PostgreSQL - https://www.postgresql.org/

δεδομένων. Για την διάταξη των συναρτήσεων χρησιμοποιήθηκε το εργαλείο Terraform[4], ένα εργαλείο αυτοματοποιημένης διαχείρισης υποδομών νέφους ανάμεσα σε πληθώρα παρόχων, συγκεκριμένα η έκδοση 1.6.5, ενώ για την εκτέλεση μη-λειτουργικών δοκιμών το εργαλείο JMeter[5], στην έκδοση 5.6.2. Η παραγωγή αναφορών επιτυγχάνεται με την χρήση του εργαλείου BiRT[6], έκδοση 4.8.0.

3.4.3 Ανάλυση Βασικών Κλάσεων

- **RestAssuredTest**



Εικόνα 13 - Διάγραμμα κλάσεων για την κλάση RestAssuredTest

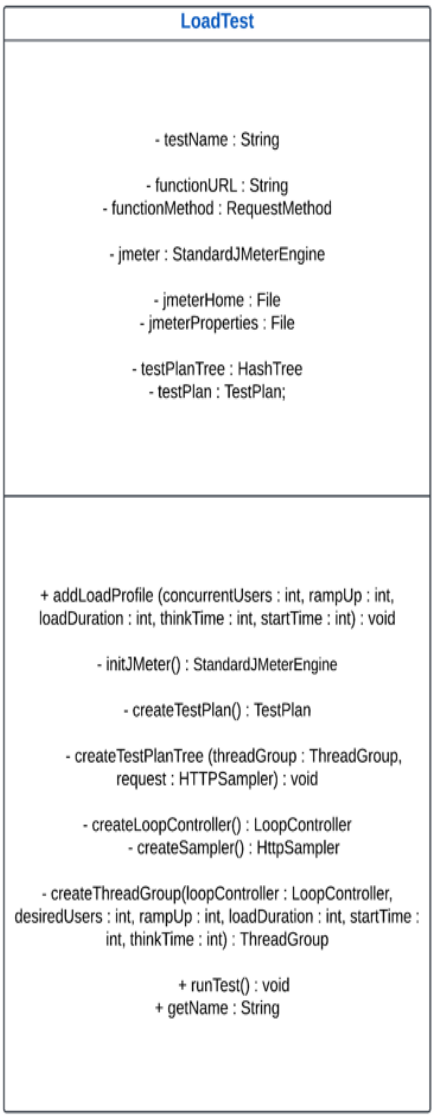
Η κλάση αυτή παραμετροποιεί μια λειτουργική δοκιμή που θα εκτελεστεί με το RestAssured. Αντιπροσωπεύει τις παραμέτρους εκτέλεσης της δοκιμής με τα πεδία httpMethod, path, pathVariable και requestPathVariableValue όπου httpMethod είναι η HTTP μέθοδος που θα χρησιμοποιεί η δοκιμή και path, pathVariable και requestPathVariableValue η μορφή του path (εάν υπάρχει) και της μεταβλητής που υπάρχει στο path καθώς και η τιμή της (εάν υπάρχουν), αντίστοιχα.

Η δοκιμή εκτελείται με την μέθοδο runTest και αποθηκεύει τα αποτελέσματα στα πεδία actualStatusCode και actualResponseBody που αντιπροσωπεύουν το HTTP Status Code που έλαβε ως απάντηση ο Ίκαρος με την εκτέλεση της δοκιμής καθώς και το σώμα που έλαβε από την ίδια απάντηση ο Ίκαρος.

[4] Terraform - https://www.terraform.io/
[5] JMeter - https://jmeter.apache.org/
[6] BiRT - https://projects.eclipse.org/projects/technology.birt

Το πεδίο pass παραμετροποιείται συγκρίνοντας τα πεδία actualStatusCode, actualResponseBody με τα πεδία expectedStatusCode και expectedResponseBody αντίστοιχα. Εάν οι άνωθεν τιμές είναι ίδιες, τότε το pass παίρνει την τιμή true (δηλ. η περίπτωση δοκιμής έχει “περαστεί” επιτυχώς και άρα η συμπεριφορά του συστήματος είναι σωστή), αλλιώς παίρνει την τιμή false.

- **LoadTest**



Εικόνα 14 - Διάγραμμα κλάσεων για την κλάση Load Test

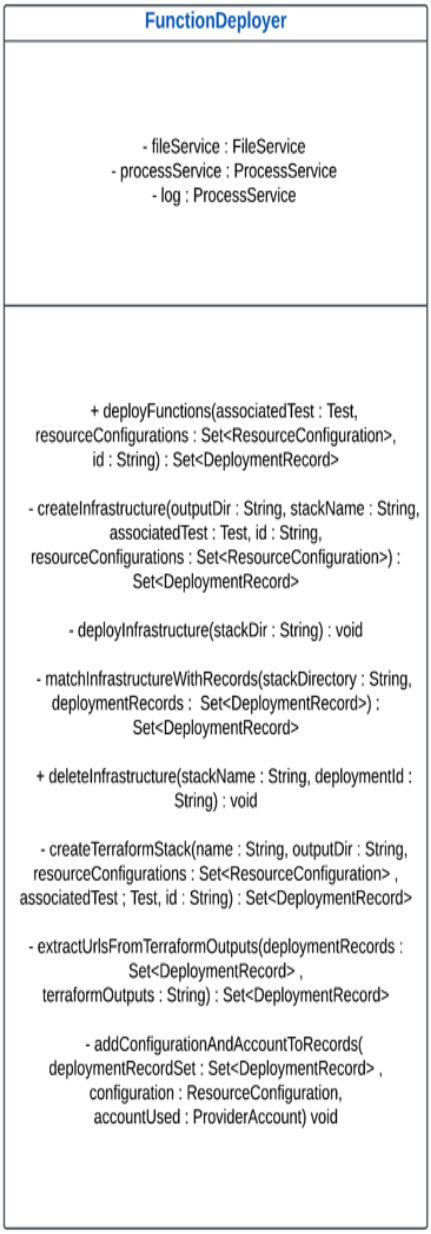
Η κλάση αυτή παραμετροποιεί μια μη-λειτουργική δοκιμή που θα εκτελεστεί με το JMeter.

Τα πεδία functionUrl και functionMethod αντιπροσωπεύουν το URL στο οποίο βρίσκεται η συνάρτηση και την μέθοδο HTTP που θα χρησιμοποιηθεί για την εκτέλεση

της δοκιμής, αντίστοιχα. Το πεδίο testPlanTree περιέχει όλα τα load profiles που έχει φτιάξει ο χρήστης για την συγκεκριμένη δοκιμή.

Για την προσθήκη ενός load profile καλείται η μέθοδος addLoadProfile με τις αντίστοιχες παραμέτρους. Δημιουργείται ένα load profile ανά παραμετροποίηση της συνάρτησης.

- **FunctionDeployer**



Εικόνα 15 - Διάγραμμα κλάσεων για την κλάση FunctionDeployer

Η κλάση αυτή είναι υπεύθυνη για την διάταξη των συναρτήσεων στους παρόχους. Χρησιμοποιεί ένα ProcessService (υπηρεσία διαδικασιών) για την

εκτέλεση εντολών Terraform στο σύστημα ή στο δοχείο που βρίσκεται η εφαρμογή και ένα FileService για διαγραφή των αρχείων που παράγει στο σύστημα/δοχείο.
Αρχικά καλείται η μέθοδος deployFunctions, που περιέχει την δοκιμή η οποία εκτελείται (οι συναρτήσεις διατάσσονται στους παρόχους ως μέρος της διαδικασίας εκτέλεσης μιας δοκιμής), τις παραμετροποιήσεις της συνάρτησης (Resource Configurations) και ένα id που έχει ανατεθεί στην εκτέλεση της δοκιμής αυτής.

Η μέθοδος deployFunctions() ενορχηστρώνει την υπόλοιπη διαδικασία διάταξης και εκτελεί σειριακά τα ακόλουθα βήματα

1. Δημιουργεί τους φακέλους (directories) που θα χρησιμοποιηθούν χρησιμοποιώντας το fileService (υπηρεσία αρχείων)

2. Δημιουργεί τα Terraform stacks (στοίβες) που θα χρησιμοποιηθούν για να διαταχθούν οι συναρτήσεις χρησιμοποιώντας την μέθοδο createInfrastructure()

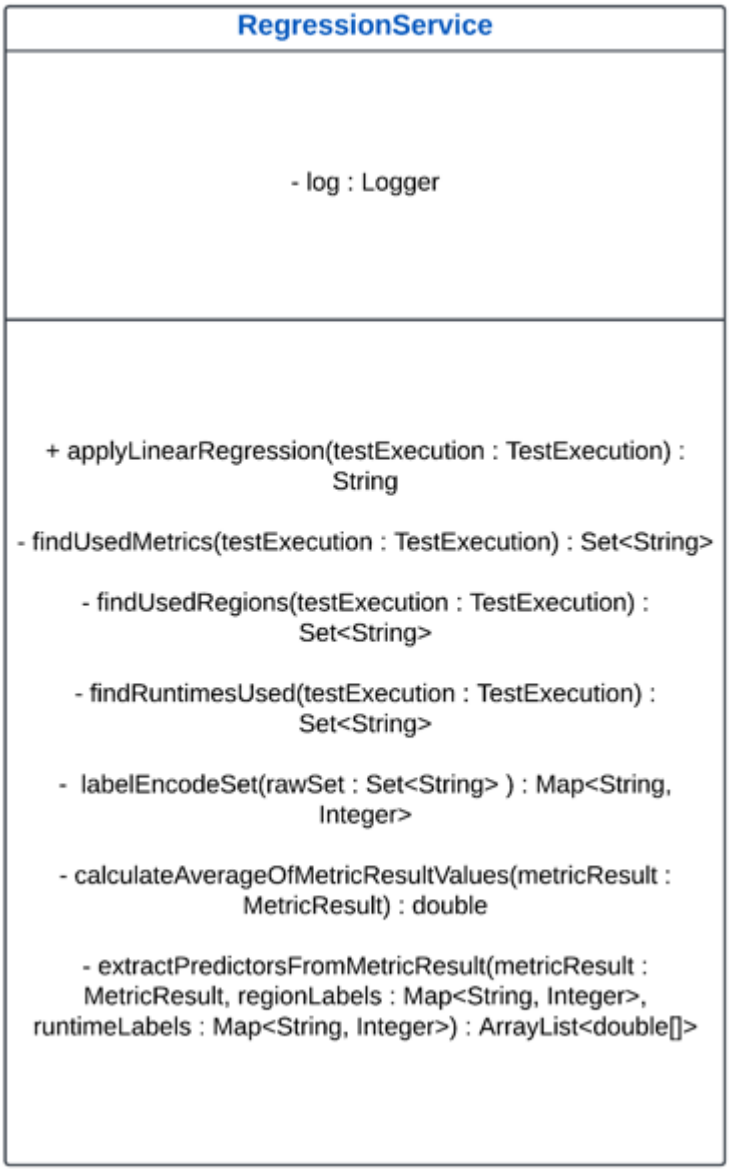
    a. Η createInfrastructure() χρησιμοποιεί τα ResourceConfigurations (διαμορφώσεις πόρων) και το Test για να εξάγει την απαραίτητη πληροφορία με την οποία θα παραμετροποιήσει τα stacks.

3. Διατάσει τις συναρτήσεις χρησιμοποιώντας τα άνωθεν stacks χρησιμοποιώντας την μέθοδο deployInfrastructure()

    a. Η μέθοδος deployInfrastructure() διατάσει τα stacks χρησιμοποιώντας ένα ProcessService στο ευρετήριο (directory) όπου βρίσκονται τα stacks, το οποίο εκτελεί μια εντολή Terraform που διατάσσει τα stacks: *terraform apply -auto-approve*

4. Παράγει τα απαραίτητα deployment records χρησιμοποιώντας την μέθοδο matchInfrastructureWithRecords()

- **RegressionService**



Εικόνα 16 - Διάγραμμα κλάσεων για την κλάση RegressionService

Η κλάση αυτή είναι υπεύθυνη για την εκτέλεση μιας γραμμικής παλινδρόμησης στα αποτελέσματα της εκτέλεσης μιας μη-λειτουργικής δοκιμής.

Αρχικά εκτελείται η μέθοδος applyLinearRegression με παράμετρο τα αποτελέσματα της εκτέλεσης μιας μη-λειτουργικής δοκιμής, η οποία ενορχηστρώνει την διαδικασία παραγωγής της αναφοράς.

Η μέθοδος applyLinearRegression() εκτελεί τις μεθόδους findUsedRegions(), findRuntimesUsed() και findUsedMetrics() έτσι ώστε να δημιουργήσει ένα σύνολο με τις μεταβλητές που υπάρχουν στο σύνολο των δεδομένων που χρησιμοποιείται.
Στην συνέχεια κωδικοποιεί τις μεταβλητές (η διαδικασία αυτή είναι απαραίτητη καθώς οι μεταβλητές βρίσκονται σε μορφή κειμένου) με την τεχνοτροπία label encoding (κωδικοποίηση ετικετών) χρησιμοποιώντας την μέθοδο labelEncodeSet() και χρησιμοποιεί την μέθοδο extractPredictorsFromMetricResult() για να παράγει τις μεταβλητές x καθώς και την μέθοδο calculateAverageOfMetricResultValues() για να υπολογίσει την μεταβλητή y.

Στο τέλος επιστρέφει μια εξίσωση της μορφής y = a*x1 + b*x2…, η οποία εκφράζει τον γραμμικό τρόπο “πρόβλεψης” ή υπολογισμού μιας μετρικής με βάση τα χαρακτηριστικά της διαμόρφωσης πόρων και την τοποθεσία. Η εξίσωση αυτή παράγεται ανά διαφορετικό πάροχο.

3.5 Ικανοποίηση απαιτήσεων

Πίνακας 2 - Βαθμός ικανοποίησης απαιτήσεων

Απαίτηση	**Βαθμός Ικανοποίησης**	**Δικαιολόγηση**
εγγραφή χρήστη	Πλήρως Ικανοποιημένη	
διαγραφή χρήστη	Πλήρως Ικανοποιημένη	
ενημέρωση στοιχείων χρήστη	Πλήρως Ικανοποιημένη	
επαναφορά/ενημέρωση κωδικού	Μη Ικανοποιημένη	Σε περίπτωση χρήσης Auth0 ικανοποιείται, σε περίπτωση χρήσης HTTP Basic δεν ικανοποιείται. Επειδή η χρήση Auth0 είναι η προτιμώμενη μέθοδος αυθεντικοποίησης και πρέπει να υπάρχει και ένα καλό επίπεδο ασφαλείας δεν δόθηκε έμφαση στην περαιτέρω ανάπτυξη υποστηρικτικών λειτουργιών

Απαίτηση	**Βαθμός Ικανοποίησης**	**Δικαιολόγηση**
		για την HTTP Basic αυθεντικοποίηση
ταυτοποίηση χρήστη	Πλήρως Ικανοποιημένη	
δημιουργία νέας δοκιμής	Πλήρως Ικανοποιημένη	
εμφάνιση δοκιμής	Πλήρως Ικανοποιημένη	
ανανέωση δοκιμής	Πλήρως Ικανοποιημένη	
διαγραφή δοκιμής	Πλήρως Ικανοποιημένη	
αναζήτηση δοκιμής	Πλήρως Ικανοποιημένη	
εκτέλεση δοκιμής	Πλήρως Ικανοποιημένη	
εμφάνιση εκτέλεσης δοκιμής	Πλήρως Ικανοποιημένη	
διαγραφή εκτέλεσης δοκιμής	Πλήρως Ικανοποιημένη	
αναζήτηση εκτέλεσης δοκιμής	Πλήρως Ικανοποιημένη	
Κάθε λειτουργία δεν μπορεί να παίρνει πάνω από 5 δευτερόλεπτα	Πλήρως Ικανοποιημένη	
Να υποστηρίζονται τουλάχιστον 50 ταυτόχρονοι χρήστες	Μερικώς Ικανοποιημένη	Εξαιτίας της χρήσης του εργαλείου Terraform, το οποίο αντιμετωπίζει ορισμένα ζητήματα με την εκτέλεση παράλληλων εργασιών, υπάρχει μια γεωμετρικά αυξανόμενη καθυστέρηση για την εξυπηρέτηση των χρηστών
Δοχειοποίηση της υπηρεσίας	Πλήρως Ικανοποιημένη	
Κατάλληλος χειρισμός λαθών/εξαιρέσεων	Πλήρως Ικανοποιημένη	
Κατάλληλη επικύρωση δεδομένων	Πλήρως Ικανοποιημένη	
Καλό επίπεδο ασφάλειας	Πλήρως Ικανοποιημένη	

4. ΕΠΙΔΕΙΞΗ

4.1 Οδηγίες Εγκατάστασης

Ο Ίκαρος βασίζεται στα εργαλεία JMeter και Terraform[7] για την σωστή λειτουργία του ενώ προϋποθέτει την χρήση μιας PostgreSQL[8] βάσης δεδομένων. Συνεπώς τα άνωθεν συστατικά πρέπει να παραμετροποιηθούν κατάλληλα ή να είναι ήδη εγκατεστημένα για την χρήση της εφαρμογής.

Ο Ίκαρος παραμετροποιείται με την χρήση των εξής μεταβλητών περιβάλλοντος:

> **Παραμετροποίηση βασικών λειτουργιών του Ίκαρου**

- FUNCTION_SOURCES_DIRECTORY
  Η τοποθεσία που ο Ίκαρος θα τοποθετήσει τον πηγαίο κώδικα των συναρτήσεων. Προεπιλεγμένη τιμή το μονοπάτι icarusData/users

> **Παραμετροποίηση του παρόχου OAuth2**

- AUTH0_AUDIENCES
  Τα audiences που πρέπει να περιέχει το oauth2 token. Τα audiences αντιπροσωπεύουν τους παραλήπτες του token και μέσα σε αυτούς πρέπει να περιλαμβάνονται οι τιμές αυτής της μεταβλητής

- AUTH0_DOMAIN
  Το domain του oauth2 παρόχου

- AUTH0_PROVIDER_JWKS
  Το Key Set του oauth2 παρόχου. Σε αυτό το URL ο Ίκαρος θα αναζητήσει τα δημόσια κλειδιά που χρησιμοποίησε ο OAuth2 πάροχος για να κρυπτογραφήσει το token

- AUTH0_PROVIDER_URI
  Το provider URI του Oauth2 παρόχου

[7] Εγκατάσταση Terraform - https://developer.hashicorp.com/terraform/tutorials/aws-get-started/install-cli
[8] Εγκατάσταση PostgreSQL - https://www.postgresql.org/docs/current/tutorial-install.html

> **Παραμετροποίηση του Auth0**

- AUTH0_SYNCHRONIZE_DATABASE
  Ενεργοποίηση της λειτουργίας συγχρονισμού της βάσης δεδομένων με εκείνη του παρόχου Auth0. Boolean μεταβλητή με προεπιλεγμένη τιμή το false

- AUTH0_MANAGEMENT_API_ID
  Το ID του Auth0 Management API στο οποίο έχει πρόσβαση ο χρήστης

- AUTH0_CLIENT_ID
  Το ID του Auth0 Client

- AUTH0_CLIENT_SECRET
  Το Secret του Auth0 Client

> Παραμετροποίηση της σύνδεσης με την Βάση Δεδομένων PostgreSQL

- DATASOURCE_URL
  Το URL στο οποίο είναι προσβάσιμη η Βάση Δεδομένων

- DB_PASSWORD
  Ο κωδικός τον οποίο θα χρησιμοποιήσει η εφαρμογή για να συνδεθεί

- DB_USERNAME
  Το όνομα χρήστη που θα χρησιμοποιήσει η εφαρμογή για να συνδεθεί

> Παραμετροποίηση του JMeter

- JMETER_HOME
  Η τοποθεσία όπου βρίσκεται εγκατεστημένο το JMeter

- JMETER_LOGS_DIR
  Η τοποθεσία που θα αποθηκεύει η εφαρμογή τα αρχεία logs μετά την εκτέλεση μη λειτουργικών δοκιμών

- JMETER_PROPERTIES
  Η τοποθεσία που βρίσκεται το config file του JMeter

> Παραμετροποίηση του Terraform

- STACK_OUTPUT_DIRECTORY
  Η τοποθεσία που ο Ίκαρος θα αποθηκεύει προσωρινά τα stacks. Ένα Terraform Stack αντιπροσωπεύει ένα σύνολο από υποδομές νέφους τις οποίες διαχειρίζεται το Terraform. Προεπιλεγμένη τιμή το μονοπάτι icarusData/terraform/stacks

- USE_LOCAL_PROVIDERS
  Ενεργοποίηση της χρήσης τοπικών Terraform Providers. Ένας Terraform Provider είναι ένα plugin (πρόσθετο) του ίδιου του Terraform που επιτρέπει την επικοινωνία με διάφορα APIs και υπηρεσίες παρόχων νέφους. Η ενεργοποίηση χρήσης τοπικών παρόχων Terraform βελτιώνει σημαντικά την απόδοση της εφαρμογής σε περιπτώσεις κακής σύνδεσης δικτύου. Αυτή η μεταβλητή περιβάλλοντος είναι Boolean με προεπιλεγμένη τιμή το false.

- LOCAL_PROVIDERS_DIRECTORY
  Η τοποθεσία που βρίσκονται εγκατεστημένοι οι τοπικοί Terraform Providers

> Παραμετροποίηση του BiRT

- TEST_CASE_RESULT_REPORT_DIRECTORY
  Η τοποθεσία που βρίσκεται το πρότυπο της αναφοράς των λειτουργικών δοκιμών

- METRIC_RESULT_REPORT_DIRECTORY
  Η τοποθεσία που βρίσκεται το πρότυπο της αναφοράς των μη λειτουργικών δοκιμών

> Παραμετροποίηση HTTP Basic

- ENABLE_HTTP_BASIC
  Ενεργοποίηση της χρήσης HTTP Basic. Boolean μεταβλητή με προεπιλεγμένη τιμή το false

- ENABLE_TEST_USER
  Χρήση ενός δοκιμαστικού λογαριασμού χρήστη. Boolean μεταβλητή με προεπιλεγμένη τιμή το false. Ο δοκιμαστικός χρήστης είναι μια λειτουργία που στοχεύει καθαρά στην δοκιμή του Ίκαρου και χρησιμοποιείται μόνο σε περίπτωση όπου επιλέγεται η HTTP Basic μέθοδος αυθεντικοποίησης. Δεν πρόκειται για λειτουργία που πρέπει να χρησιμοποιείται σε περιβάλλοντα παραγωγής (production)

- TEST_USER_EMAIL
  Το email του δοκιμαστικού λογαριασμού

- TEST_USER_PASSWORD
  Ο κωδικός του δοκιμαστικού λογαριασμού

- TEST_USER_USERNAME
  Το όνομα χρήστη του δοκιμαστικού λογαριασμού

Ο Ίκαρος μπορεί να παραμετροποιηθεί με την χρήση μεταβλητών περιβάλλοντος σε όλα τα σενάρια εγκατάστασης που περιγράφονται παρακάτω. Για την εγκατάσταση της εφαρμογής χρησιμοποιώντας το jar αρχείο ή τον maven wrapper προαπαιτείται ο πηγαίος κώδικας της εφαρμογής, που μπορεί να βρεθεί στο αποθετήριο https://github.com/ttomtsis/icarus.

4.1.1 Docker

Η εγκατάσταση με την χρήση της εικόνας δοχείου είναι ο προτιμότερος τρόπος εγκατάστασης της εφαρμογής. Ο χρήστης χρειάζεται μονάχα το εργαλείο docker[9] εγκατεστημένο, έπειτα απλώς χρησιμοποιεί το αρχείο docker compose, το οποίο παρέχεται ήδη, εκτελώντας την εντολή *docker compose up* ώστε να καταστήσει την εφαρμογή έτοιμη προς χρήση.

4.1.2 Jar

Η εφαρμογή μπορεί επίσης να εγκατασταθεί με την χρήση ενός jarfile το οποίο μπορεί να βρεθεί στην σελίδα του αποθετηρίου της εφαρμογής στο GitHub. Για την εγκατάσταση της με την χρήση jar απαιτείται η εγκατάσταση και σωστή παραμετροποίηση της Java 21[10], καθώς και μια βάση PostgreSQL που να είναι προσβάσιμη από την εφαρμογή. Το εργαλείο Terraform[11] πρέπει να βρίσκεται στο PATH ενώ απαιτείται και το nodeJS[12] εγκατεστημένο (από την έκδοση 18 και μεταγενέστερα) έτσι ώστε να λειτουργεί ομαλά το Terraform CDK. Επίσης, πρέπει να παραμετροποιηθεί η μεταβλητή περιβάλλοντος JMETER_HOME και να δείχνει στον φάκελο εγκατάστασης του JMeter.

Έπειτα, ο χρήστης μπορεί να χρησιμοποιήσει την εφαρμογή όπως κάθε άλλη Java εφαρμογή, εκτελώντας το αρχείο jar με την χρήση της εντολής java (πχ *java -jar icarus.jar*)

[9] Εγκατάσταση docker - https://docs.docker.com/engine/install/
[10] Εγκατάσταση Java 21 - https://docs.oracle.com/en/java/javase/21/install/overview-jdk-installation.html#GUID-8677A77F-231A-40F7-98B9-1FD0B48C346A
[11] Εγκατάσταση Terraform - https://developer.hashicorp.com/terraform/tutorials/aws-get-started/install-cli
[12] Εγκατάσταση NodeJs - https://nodejs.org/en/learn/getting-started/how-to-install-nodejs

4.1.3 Maven/Maven wrapper

Ο χρήστης μπορεί να χρησιμοποιήσει είτε το εργαλείο maven είτε τον maven wrapper για να παράγει ένα jar και να εγκαταστήσει την εφαρμογή, όπως εξηγήθηκε προηγουμένως. Για την χρήση του maven απαιτείται η εγκατάσταση ενός JDK στο περιβάλλον του χρήστη (εφόσον δεν υπάρχει ήδη - δείτε ενότητα 4.1.2). Αρχικά, ο χρήστης μπορεί να χρησιμοποιήσει το εργαλείο git[13] για να κάνει clone το αποθετήριο από GitHub και να χρησιμοποιήσει το maven για να παράγει το jar αρχείο χρησιμοποιώντας την εντολή *mvn clean package*, συνεχίζοντας όπως περιγράφηκε παραπάνω.

4.2 Επίδειξη Εφαρμογής

4.2.1 Εγγραφή χρήστη και παροχή διαπιστευτηρίων παρόχου νέφους

- **ΒΗΜΑ 1: Εγγραφή χρήστη**
    - **Εγγραφή με την χρήση HTTP Basic**
      Ο χρήστης στέλνει ένα αίτημα εγγραφής χρήστη στο endpoint /api/v0/users/register χρησιμοποιώντας την HTTP μέθοδο POST.
      Το αίτημα πρέπει να περιέχει στο σώμα του, ένα αντικείμενο JSON που να περιλαμβάνει τα πεδία username, password και email.

      πχ
      ```
      {
        "username": "demoAcc",
        "password": "demoAcc",
        "email": "demoAcc@demoEmail.com"
      }
      ```

    - **Εγγραφή με την χρήση Auth0**
      Ο χρήστης χρησιμοποιεί την λειτουργία register του Auth0 στέλνοντας ένα αίτημα POST στο OAuth authorization URL της Auth0 εφαρμογής του. Αφού προμηθευτεί από το Auth0 ένα JWT token, τότε ο χρήστης είναι ελεύθερος να χρησιμοποιήσει το εν λόγω token για να αποδείξει την ταυτότητα του στον Ίκαρο. Το εν λόγω token πρέπει να χρησιμοποιηθεί και στις επόμενες λειτουργίες.

- **ΒΗΜΑ 2: Παροχή διαπιστευτηρίων παρόχου νέφους**
    - **Προσθήκη λογαριασμού Amazon Web Services - AWS**

[13] Εγκατάσταση Git - https://git-scm.com/book/en/v2/Getting-Started-Installing-Git

Ο χρήστης αποστέλλει ένα αίτημα προσθήκης AWS λογαριασμού στο endpoint /api/v0/users/{username}/accounts/aws χρησιμοποιώντας την HTTP μέθοδο POST όπου {username} είναι το όνομα χρήστη του. Το αίτημα πρέπει να περιέχει στο σώμα του ένα αντικείμενο JSON που να περιλαμβάνει τα πεδία name, awsAccessKey και awsSecretKey. Τα πεδία awsAccessKey και awsSecretKey είναι το ζεύγος κλειδιών που προμηθεύεται ο χρήστης μέσω της γραφικής διεπαφής του AWS για την προγραμματιστική πρόσβαση στις υπηρεσίες του AWS.

- **Προσθήκη λογαριασμού Google Cloud Platform – GCP**
  Ο χρήστης αποστέλλει ένα αίτημα προσθήκης GCP λογαριασμού στο endpoint /api/v0/users/{username}/accounts/gcp χρησιμοποιώντας την HTTP μέθοδο POST.
  Το αίτημα πρέπει να περιέχει στο σώμα του ένα αντικείμενο JSON που να περιλαμβάνει τα πεδία name, gcpProjectId και gcpKeyfile. Το gcpProjectId αντιπροσωπεύει το project (έργο) που θα χρησιμοποιήσει η εφαρμογή (το εν λόγω έργο πρέπει να υπάρχει ήδη και δεν δημιουργείται αυτόματα από την εφαρμογή) ενώ το gcpKeyfile είναι ένα εμφωλευμένο αντικείμενο json που περιέχει τα πεδία type, project_id, private_key_id, private_key, client_email, client_id, auth_uri, token_uri, auth_provider_x509_cert_url, client_x509_cert_url και universe_domain. Τα πεδία του gcpKeyfile συμπληρώνονται σύμφωνα με το keyfile που προμηθεύεται ο χρήστης από το GCP.

4.2.2 Διενέργεια λειτουργικής δοκιμής

Για την διενέργεια όλων των βασικών λειτουργιών της προτεινόμενης υπηρεσίας, συμπεριλαμβανομένου αυτής της λειτουργικής δοκιμής, απαιτείται η ολοκλήρωση της προηγούμενης περίπτωσης χρήσης: ‘Εγγραφή χρήστη και παροχή διαπιστευτηρίων παρόχου νέφους.

- **ΒΗΜΑ 1: Δημιουργία λειτουργικής δοκιμής**
  Ο χρήστης αποστέλλει ένα αίτημα δημιουργίας λειτουργικής δοκιμής στο endpoint api/v0/tests/functional χρησιμοποιώντας την HTTP μέθοδο POST.
  Το αίτημα πρέπει να περιλαμβάνει στο σώμα του τα πεδία name και httpMethod.
  Το πεδίο httpMethod αναπαριστά την μέθοδο HTTP που θα χρησιμοποιηθεί στο αίτημα που θα σταλεί από τον Ίκαρο για την διενέργεια της λειτουργικής δοκιμής.

  πχ
  ```
  {
     "name": "DemoTest",
     "httpMethod": "GET"
  ```

}

- **ΒΗΜΑ 2: Δημιουργία Test Case για την λειτουργική δοκιμή (Μπορεί να επαναληφθεί, μια λειτουργική δοκιμή περιέχει πολλά Test Cases)**
  Ο χρήστης αποστέλλει ένα αίτημα δημιουργίας Test Case στο endpoint api/v0/tests/functional/{functionalTestID}/test-cases χρησιμοποιώντας την HTTP μέθοδο POST, όπου {functionalTestID} είναι το αναγνωριστικό της λειτουργικής δοκιμής που μόλις δημιουργήθηκε προηγουμένως.
  Το αίτημα πρέπει να περιλαμβάνει στο σώμα του τα πεδία name και description.

- **ΒΗΜΑ 3: Δημιουργία Test Case Member για το Test Case (Μπορεί να επαναληφθεί, ένα Test Case μπορεί να έχει πολλά Test Case Members)**
  Ο χρήστης αποστέλλει ένα αίτημα δημιουργίας Test Case Member στο endpoint api/v0/tests/functional/{functionalTestID}/test-cases/{testCaseID}/test-case-members χρησιμοποιώντας την HTTP μέθοδο POST, όπου {testCaseID} είναι το αναγνωριστικό της περίπτωσης δοκιμής που μόλις δημιουργήθηκε προηγουμένως.
  Το αίτημα πρέπει να περιλαμβάνει στο σώμα του τα πεδία expectedResponseCode και expectedResponseBody, τα οποία αναπαριστούν το Http Status Code και το Body (σώμα) που να αναμένεται να επιστραφεί σε περίπτωση σωστής λειτουργίας της συνάρτησης, αντίστοιχα.

  πχ
  ```
  {
    "expectedResponseCode": 200,
    "expectedResponseBody": ""
  }
  ```

- **ΒΗΜΑ 4: Δημιουργία Resource Configuration για την λειτουργική δοκιμή**
  Ο χρήστης αποστέλλει ένα αίτημα δημιουργίας Resource Configuration στο endpoint api/v0/tests/{functionalTestID}/resource-configurations χρησιμοποιώντας την HTTP μέθοδο POST.
  Το αίτημα πρέπει να περιλαμβάνει στο σώμα του τα πεδία regions, memoryConfigurations, platform και functionRuntime οπου regions είναι μια λίστα με τις τοποθεσίες διάταξης των συναρτήσεων, memoryConfigurations μια λίστα με το μέγεθος της μνήμης που θα έχουν οι συναρτήσεις και functionRuntime το περιβάλλον εκτέλεσης των συναρτήσεων.

  πχ
  ```
  {
    "regions": ["US_EAST_1"],
    "memoryConfigurations": [256],
    "platform": "AWS",
  ```

```
"functionRuntime": "NODEJS16"
    }
```

- **ΒΗΜΑ 5: Δημιουργία Συνάρτησης**
  Ο χρήστης αποστέλλει ένα αίτημα δημιουργίας Συνάρτησης στο endpoint api/v0/functions χρησιμοποιώντας την HTTP μέθοδο POST.
  Το αίτημα πρέπει να περιλαμβάνει δεδομένα φόρμας (form data) με δύο πεδία. Το πεδίο functionMetadata περιέχει δεδομένα αναφορικά με την συνάρτηση και πρέπει να περιλαμβάνει τα υπο-πεδία name, description και functionHandler ενώ το πεδίο functionSource περιέχει τον πηγαίο κώδικα της συνάρτησης σε μορφή zip ως Multipart Form Data. Το πεδίο functionHandler αναπαριστά το σημείο εισόδου της συνάρτησης, δηλαδή την μέθοδο/κλάση που θα κληθεί σε περίπτωση ενεργοποίησης της συνάρτησης.

  πχ
  ```
  {
    "name": "demoFunction",
    "description": "Deployment Test",
    "functionHandler": "main.handler"
  }
  ```

- **ΒΗΜΑ 6: Ανανέωση των στοιχείων της λειτουργικής δοκιμής**
  Ο χρήστης αποστέλλει ένα αίτημα ανανέωσης στοιχείων (λειτουργικής) δοκιμής στο endpoint api/v0/tests/functional/{functionalTestID} χρησιμοποιώντας την HTTP μέθοδο PUT.
  Το αίτημα πρέπει να περιλαμβάνει στο σώμα του τα πεδία accountsList και targetFunction, στα οποία αντιστοιχούν στο ID του λογαριασμού που θα χρησιμοποιήσει ο χρήστης για την εκτέλεση της δοκιμής και στο ID της συνάρτησης, η οποία θα συμμετέχει στην δοκιμή, αντίστοιχα. Σε περίπτωση που ο χρήστης δεν επιθυμεί διάταξη της συνάρτησης, μπορεί να παρέχει και το πεδίο functionUrl. Το πεδίο functionUrl αναπαριστά το URL στο οποίο έχει ήδη διαταχθεί η συνάρτηση (από τον χρήστη).

  πχ
  ```
  {
    "accountsList": [1],
    "targetFunction": 1
  }
  ```

- **ΒΗΜΑ 7: Εκτέλεση της δοκιμής**
  Ο χρήστης αποστέλλει ένα αίτημα εκτέλεσης δοκιμής στο endpoint api/v0/tests/functional/{functionalTestID}/execute χρησιμοποιώντας την HTTP μέθοδο POST.
  Σε αυτό το σημείο ο χρήστης ξεκίνησε την ασύγχρονη εκτέλεση της λειτουργικής δοκιμής. Στην απάντηση του Ίκαρου υπάρχει η κεφαλίδα Location, που δείχνει το URL που μπορεί να χρησιμοποιήσει ο χρήστης για να ελέγξει την πορεία της εκτέλεσης (της δοκιμής).

- **ΒΗΜΑ 8: Έλεγχος της κατάστασης εκτέλεσης της δοκιμής**
  Ο χρήστης αποστέλλει ένα αίτημα ελέγχου κατάστασης εκτέλεσης δοκιμής στο endpoint
  api/v0/tests/functional/{functionalTestID}/executions/{deploymentID}/status
  χρησιμοποιώντας την HTTP μέθοδο GET, όπου {deploymentId} είναι το αναγνωριστικό που αναθέτει ο Ίκαρος στην συγκεκριμένη διαδικασία εκτέλεσης της δοκιμής (Μία δοκιμή μπορεί να εκτελείται πολλές φορές ταυτόχρονα και σε κάθε εκτέλεση της ο Ίκαρος αναθέτει νέο αναγνωριστικό). Το εν λόγω URL παρέχεται από τον Ίκαρο στην κεφαλίδα location του προηγούμενου βήματος.

- **ΒΗΜΑ 9: Λήψη αναφοράς αποτελεσμάτων της λειτουργικής δοκιμής**
  Εφόσον η λειτουργική δοκιμή έχει ολοκληρωθεί, ο χρήστης μπορεί να αποστέλλει ένα αίτημα λήψης της σχετικής αναφοράς στο endpoint api/v0/tests/{functionalTestID}/executions/reports?deploymentID={deployment ID} χρησιμοποιώντας την HTTP μέθοδο GET.

  Στην αναφορά που παράγει ο Ίκαρος διακρίνονται καθαρά οι λεπτομέρειες που αφορούν την δοκιμή, η παραμετροποίηση της συνάρτησης και σε ποια πλατφόρμα διατάχθηκε καθώς και μια λίστα των Test Cases που εξετάστηκαν μαζί με τα αποτελέσματα τους. Επίσης, διακρίνεται καθαρά ο συνολικό αριθμός επιτυχημένων και αποτυχημένων περιπτώσεων δοκιμής. Για κάθε περίπτωση δοκιμής αναγράφονται ξεκάθαρα οι είσοδοι και οι έξοδοι καθώς και τα αποτελέσματα της.

  Το πρότυπο αναφοράς για λειτουργικές δοκιμές που παρέχεται μαζί με τον Ίκαρο δημιουργεί αναφορές στην παρακάτω μορφή:

ICARUS REPORT

Creation date: Feb 16 2024 5:54:57 PM

Report details:

Test Execution:		**Status:**	
Author:	testuser	**Start date:**	
Test:	DemoTest3	**End date:**	

Test details:

Test ID:	1
Test Name:	DemoTest3
Test Description:	
Http Method:	GET
Exposed Path:	
Exposed Path Variable:	

Resource configuration details:

ID	Runtime	Platform	Memory	CPU	Region
1	NODEJS16	AWS	256		US_EAST_1

Function details:

Function ID:	1
Name:	demoFunction101
Description:	Deployment Test
Entrypoint:	main.handler

Test Case Results:

Test Case ID	Test Case Member ID	Request Body	Request Path Variable	Expected Response Code	Expected Response Body	Actual Response Code	Actual Response Body	PASS ?
1	1			200		200	Hello, World!	FAIL

Test Case Result Count:

Test Cases Passed:	0
Test Cases Failed:	1

Εικόνα 17 - Παράδειγμα αναφοράς για λειτουργικές δοκιμές

4.2.3 Διενέργεια μη-λειτουργικής δοκιμής

Για την διενέργεια της μη-λειτουργικής δοκιμής, απαιτείται, όπως προαναφέρθηκε, η ολοκλήρωση της προηγούμενης περίπτωσης χρήσης: ‘Εγγραφή χρήστη και παροχή διαπιστευτηρίων παρόχου νέφους.

- **ΒΗΜΑ 1: Δημιουργία μη-λειτουργικής δοκιμής**
  Ο χρήστης αποστέλλει ένα αίτημα δημιουργίας μη-λειτουργικής δοκιμής στο endpoint api/v0/tests/performance χρησιμοποιώντας την HTTP μέθοδο POST. Το αίτημα πρέπει να περιλαμβάνει στο σώμα του τα πεδία name, httpMethod και chosenMetrics, όπου name το όνομα που επιλέγει ο χρήστης για την δοκιμή, httpMethod η μέθοδος HTTP που θα χρησιμοποιηθεί για την διενέργεια της δοκιμής και chosenMetrics η λίστα με τις μετρικές που θα συλλεχθούν με το πέρας της δοκιμής.

  πχ
  ```
  {
     "name": "DemoPerformanceTest",
     "targetFunction": 1,
     "httpMethod": "GET",

     "chosenMetrics": ["INVOCATIONS"]
          }
  ```

- **ΒΗΜΑ 2: Δημιουργία Load Profile για την μη-λειτουργική δοκιμή**
  Ο χρήστης αποστέλλει ένα αίτημα δημιουργίας Load Profile στο endpoint api/v0/tests/performance/{performanceTestID}/load-profiles χρησιμοποιώντας την HTTP μέθοδο POST, όπου {performanceTestID} είναι το αναγνωριστικό της μη λειτουργικής δοκιμής που δημιουργήθηκε προηγουμένως.
  Το αίτημα πρέπει να περιλαμβάνει στο σώμα του τα πεδία loadTime, rampUp, concurrentUsers, startDelay και thinkTime. Το πεδίο loadTime αναπαριστά την διάρκεια που θα εφαρμοστεί φόρτος στο σύστημα, το πεδίο rampUp αναπαριστά την χρονική στιγμή που όλοι οι χρήστες θα έχουν εισέλθει στο σύστημα και θα δημιουργούν φόρτο, το πεδίο concurrentUsers αναπαριστά τον αριθμό των ταυτόχρονων χρηστών που παράγουν τον φόρτο ενώ τα πεδία startDelay και thinkTime αναπαριστούν την αρχική καθυστέρηση πριν την εφαρμογή του φόρτου και την καθυστέρηση κάθε ταυτόχρονου χρήστη πριν την εφαρμογή του φόρτου που του αναλογεί, αντίστοιχα. Ο χρήστης της εφαρμογής μπορεί να εκτελέσει αυτό το βήμα πολλές φορές για μία μη λειτουργική δοκιμή.

πχ

```
{
   "loadTime": 60,
   "rampUp": 0,
   "concurrentUsers": 10,
   "startDelay": 0,
   "thinkTime": 0
}
```

- **ΒΗΜΑ 3: Δημιουργία Resource Configuration για την μη-λειτουργική δοκιμή**
  Ο χρήστης αποστέλλει ένα αίτημα δημιουργίας Resource Configuration στο endpoint api/v0/tests/{performanceTestID}/resource-configurations χρησιμοποιώντας την HTTP μέθοδο POST.
  Το αίτημα πρέπει να περιλαμβάνει στο σώμα του τα πεδία regions, memoryConfigurations, platform και functionRuntime. Αυτό το βήμα και τα πεδία του είναι τα ίδια με την περίπτωση στην δημιουργία της λειτουργικής δοκιμής

- **ΒΗΜΑ 4: Δημιουργία Συνάρτησης**
  Ο χρήστης αποστέλλει ένα αίτημα δημιουργίας Συνάρτησης στο endpoint api/v0/functions χρησιμοποιώντας την HTTP μέθοδο POST (εφόσον η συνάρτηση έχει ήδη δημιουργηθεί από τον Ίκαρο, τότε απλώς κρατάμε το ID της συνάρτησης και το επαναχρησιμοποιούμε αργότερα).
  Το αίτημα πρέπει να περιλαμβάνει δεδομένα φόρμας (form data) με δύο πεδία. Το πεδίο functionMetadata περιέχει δεδομένα αναφορικά με την συνάρτηση και πρέπει να περιλαμβάνει τα υπο-πεδία name, description και functionHandler ενώ το πεδίο functionSource περιέχει τον πηγαίο κώδικα της συνάρτησης σε μορφή zip ως Multipart Form Data. Αυτό το βήμα και τα πεδία του είναι τα ίδια με την περίπτωση στην δημιουργία της λειτουργικής δοκιμής

- **ΒΗΜΑ 5: Ανανέωση των στοιχείων της μη-λειτουργικής δοκιμής**
  Ο χρήστης αποστέλλει ένα αίτημα ανανέωσης στοιχείων δοκιμής στο endpoint api/v0/tests/functional/{performanceTestID} χρησιμοποιώντας την HTTP μέθοδο PUT.
  Το αίτημα πρέπει να περιλαμβάνει στο σώμα του τα πεδία accountsList και targetFunction, τα οποία αντιστοιχούν στο ID του λογαριασμού που θα χρησιμοποιήσει ο χρήστης για την εκτέλεση της δοκιμής και στο ID της συνάρτησης, η οποία θα συμμετέχει στην δοκιμή, αντίστοιχα.

πχ

```
{
   "accountsList": [1],
   "targetFunction": 1
}
```

- **ΒΗΜΑ 6: Εκτέλεση της δοκιμής**
  Ο χρήστης αποστέλλει ένα αίτημα εκτέλεσης δοκιμής στο endpoint api/v0/tests/functional/{performanceTestID}/execute χρησιμοποιώντας την HTTP μέθοδο POST.
  Σε αυτό το σημείο ο χρήστης ξεκίνησε την ασύγχρονη εκτέλεση της μη-λειτουργικής δοκιμής. Στην απάντηση του Ίκαρου υπάρχει η κεφαλίδα Location, που δείχνει το URL που μπορεί να χρησιμοποιήσει ο χρήστης για να ελέγξει την πορεία της εκτέλεσης.
  Αυτό το βήμα είναι ίδιο με την περίπτωση της λειτουργικής δοκιμής.

- **ΒΗΜΑ 7: Έλεγχος της κατάστασης εκτέλεσης της δοκιμής**
  Ο χρήστης αποστέλλει ένα αίτημα ελέγχου κατάστασης εκτέλεσης δοκιμής στο endpoint
  api/v0/tests/functional/{performanceTestID}/executions/{deploymentID}/status χρησιμοποιώντας την HTTP μέθοδο GET, όπου {deploymentId} είναι το αναγνωριστικό που αναθέτει ο Ίκαρος στην συγκεκριμένη διαδικασία εκτέλεσης της δοκιμής (Μία δοκιμή μπορεί να εκτελείται πολλές φορές ταυτόχρονα και σε κάθε εκτέλεση της ο Ίκαρος αναθέτει νέο αναγνωριστικό). Το εν λόγω URL παρέχεται από τον Ίκαρο στην κεφαλίδα location του προηγούμενου βήματος.

- **ΒΗΜΑ 8: Λήψη αναφοράς αποτελεσμάτων της λειτουργικής δοκιμής**
  Ο χρήστης αποστέλλει ένα αίτημα λήψης αναφοράς στο endpoint api/v0/tests/{performanceTestID}/executions/reports?deploymentID={deploymentID} χρησιμοποιώντας την HTTP μέθοδο GET. Αυτό το βήμα είναι ίδιο με την περίπτωση της λειτουργικής δοκιμής

  Το πρότυπο αναφοράς για μη-λειτουργικές δοκιμές που παρέχεται μαζί με τον Ίκαρο δημιουργεί αναφορές στην παρακάτω μορφή. Περιέχει αναλυτικά λεπτομέρειες που αφορούν την δοκιμή, τις διατάξεις της συνάρτησης καθώς και τα προφίλ φόρτου που χρησιμοποιήθηκαν ενώ παρουσιάζει στον χρήστη αναλυτικά τις μετρικές που συλλέχθηκαν σε συνδυασμό με γραφικές παραστάσεις που έφτιαξε ο ίδιος ο Ίκαρος.

ICARUS REPORT

Creation date: Feb 16 2024 6:16:26 PM

Report details:

Test Execution:	2	**Status:**	FINISHED
Author:	testuser	**Start date:**	Feb 16, 2024 6:11 PM
Test:	DemoPerformanceTest	**End date:**	Feb 16, 2024 6:15 PM

Test details:

Test ID:	2
Test Name:	DemoPerformanceTest
Test Description:	
Http Method:	GET
Exposed Path:	
Exposed Path Variable:	
Path Variable's value	
Request Body used:	

Metrics
INVOCATIONS

Resource configuration details:

ID	Runtime	Platform	Memory	CPU	Region
2	NODEJS16	AWS	256		US_EAST_1

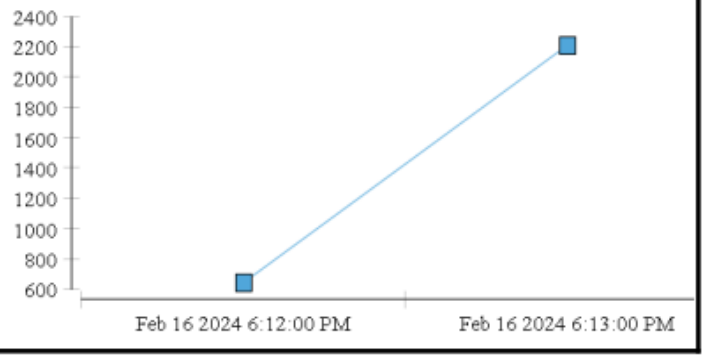
Load Profile details:

ID	Concurrent Users	Load Time	Ramp Up	Start Delay	Think Time
1	10	60	0	0	0

Function details:

Function ID:	1
Name:	demoFunction101
Description:	Deployment Test
Entrypoint:	main.handler

Metric Results:

ID	Configuration	Metric Name	Metric Results Graph
1	2	INVOCATIONS	

Linear Regression:

Linear regressions for metrics:
Metric: INVOCATIONS
Unable to perform regression: not enough data (1 rows) for this many predictors (3 predictors)

Εικόνα 18 - Παράδειγμα αναφοράς για δοκιμές απόδοσης

5. ΣΥΜΠΕΡΑΣΜΑΤΑ & ΜΕΛΛΟΝΤΙΚΕΣ ΚΑΤΕΥΘΥΝΣΕΙΣ

5.1 Συνεισφορά

Ο Ίκαρος αποτελεί ένα χρήσιμο ερευνητικό εργαλείο που μπορεί να χρησιμοποιηθεί για την σύγκριση διαφορετικών παρόχων υπηρεσιών νέφους. Προσφέρει έναν πολύ απλό τρόπο συγγραφής δοκιμών και διάταξης συναρτήσεων σε διαφορετικούς παρόχους, επιτρέποντας την μελέτη της λειτουργικής και μη λειτουργικής συμπεριφοράς μιας συνάρτησης αλλά και την ανάπτυξη μοντέλων απόδοσης. Η εφαρμογή υποστηρίζει τις πλατφόρμες AWS Lambda και Google Cloud Functions, έτσι ώστε να μην εστιάσει μόνο στην πλατφόρμα AWS Lambda που έχει μελετηθεί υπερβολικά. Ο Ίκαρος συνεισφέρει στην διέγερση της έρευνας στον τομέα του serverless computing καθώς αφαιρεί την πολυπλοκότητα της υποκείμενης διαδικασίας διάταξης και επιτρέπει την διενέργεια αυτοματοποιημένων δοκιμών που έχουν συγγραφεί από τους χρήστες ενώ αυτοματοποιεί την διαδικασία παραγωγής ευρημάτων από κάθε δοκιμή.

Ταυτόχρονα ο Ίκαρος μπορεί να ενσωματωθεί στις επιχειρηματικές ανάγκες οποιουδήποτε οργανισμού, είτε με τροποποιήσεις στον κώδικα του, είτε στην αρχική μορφή του καθώς προσφέρεται με την άδεια MPL 2.0.

5.2 Συμπεράσματα

Κατά την διαδικασία συγγραφής της εφαρμογής συμπεράναμε τους λόγους που η πλατφόρμα AWS Lambda έχει περίοπτη θέση, ανάμεσα στις πλατφόρμες FaaS. Η συντριπτική πλειοψηφία των serverless εργαλείων υποστηρίζουν την πλατφόρμα AWS Lambda, ενώ η ίδια η πλατφόρμα διαθέτει ένα μεγάλο εύρος υπηρεσιών. Πιθανολογείται πως αυτό συμβαίνει εξαιτίας του μεγαλύτερου χρόνου ύπαρξης της στην αγορά των serverless υπηρεσιών αλλά και εξαιτίας της μαζικής αποδοχής της από την βιομηχανία. Το εύρος των εργαλείων που υποστηρίζουν την πλατφόρμα AWS Lambda κυμαίνεται από τοπικούς προσομοιωτές των υποδομών της Amazon (Emulators όπως LocalStack [14]) μέχρι επεκτάσεις για την κύρια λειτουργικότητα της πλατφόρμας από τους επίσημους συνεργάτες της Amazon.

Ταυτόχρονα, στην περίπτωση των εργαλείων ή των πλαισίων που αφορούν πολλαπλούς παρόχους (πχ. Serverless Framework[15], Terraform, Pulumi[16]), οι λειτουργίες που στοχεύουν στην χρήση των υποδομών της Amazon λειτουργούν πολύ καλύτερα συγκριτικά με εκείνες άλλων παρόχων. Επίσης, η Amazon προσφέρει

[14] LocalStack Emulator - https://www.localstack.cloud/
[15] Serverless framework - https://www.serverless.com/
[16] Pulumi - https://www.pulumi.com/

πλήθος SDK's και API's για την επέκταση της κύριας λειτουργικότητας της πλατφόρμας αλλά και την υλοποίηση εργαλείων, με πολύ γνωστό παράδειγμα το εργαλείο Terraform (που χρησιμοποιεί ο Ίκαρος), το οποίο χρησιμοποιεί το Jsii kernel[17] που προσφέρει η εταιρεία. Η διαδικασία διάταξης συναρτήσεων στην Lambda είναι περίτεχνη και ιδιαίτερα λεπτομερής. Ο χρήστης καλείται να παραμετροποιήσει κάθε πτυχή της διάταξης ενώ ταυτόχρονα η διαδικασία παρακολούθησης των συναρτήσεων και συλλογής μετρικών είναι αρκετά λεπτομερής.

Εν αντιθέση, στην περίπτωση της πλατφόρμας Google Cloud Functions (GCF), τα εργαλεία που προσφέρονται από την ίδια την πλατφόρμα είναι λιγότερα, ενώ η ενσωμάτωση της λειτουργικότητας της πλατφόρμας σε εργαλεία τρίτων συνήθως είναι σε πειραματικό στάδιο (εάν υπάρχει), όπως στην περίπτωση τόσο του Serverless Framework όσο και του Terraform (αξίζει να σημειωθεί πως δεν υπάρχει αυτήν την στιγμή προσομοιωτής των υποδομών της Google που να λειτουργεί αξιόπιστα). Η αρχιτεκτονική των συναρτήσεων είναι εγγενώς απλούστερη συγκριτικά με την πλατφόρμα AWS Lambda, κάτι που μπορούμε να δούμε με το πολύ απλό παράδειγμα της διάταξης μιας σύνθετης RESTful υπηρεσίας και στις δύο πλατφόρμες. Στην περίπτωση της AWS Lambda, για κάθε endpoint του API πρέπει να δημιουργηθεί και να παραμετροποιηθεί στο ApiGateway ξεχωριστό Route (το ApiGateway είναι η υπηρεσία της Amazon που είναι υπεύθυνη για την διαχείριση των routes και ένα route αντιπροσωπεύει ένα endpoint του API), ενώ στην περίπτωση της GCF ο χρήστης απλώς διατάσει την συνάρτηση. Επίσης, συγκριτικά με την πλατφόρμα AWS Lambda, οι μετρικές που προσφέρει η GCF δεν είναι το ίδιο λεπτομερείς και ο χρήστης καλείται συχνά να δημιουργήσει τις δικές του. Συνολικά, η χρήση της GCF αποτελεί καλό σημείο εκκίνησης για την κατανόηση βασικών εννοιών του FaaS μοντέλου εξαιτίας της πολύ απλούστερης προσέγγισης που ακολουθεί συγκριτικά με την AWS Lambda.

5.3 Εμπειρία που αποκομίσθηκε

Κατά την ανάπτυξη της εφαρμογής, υπήρξε η ανάγκη τροποποίησης του αρχικού διαγράμματος οντοτήτων συσχετίσεων. Κατά την διάρκεια αυτών των τροποποιήσεων έγινε εμφανές το κόστος που επιφέρουν αυτές οι αλλαγές στην περίπτωση ενός μεσαίου/μεγάλου μεγέθους έργου λογισμικού.

Αντίστοιχα, έγινε εμφανής η ανάγκη της σωστής επιλογής των εργαλείων που θα ενσωματωθούν στην εφαρμογή. Προτού υλοποιηθεί ο Ίκαρος, αναπτύχθηκε πληθώρα μικρότερων πρωτοτύπων με στόχο την εξοικείωση, την καλύτερη κατανόηση και τον έλεγχο των εξωτερικών εργαλείων που θα μπορούσαν να χρησιμοποιηθούν για τις ανάγκες της εφαρμογής. Μέσω αυτής της διαδικασίας απορρίφθηκε η χρήση του serverless framework, καθώς εστίαζε στην ανάπτυξη συναρτήσεων στην πλατφόρμα Lambda και όχι συνολικά στην διαχείριση συναρτήσεων σε διαφορετικούς παρόχους, ενώ προτιμήθηκε η χρήση του Terraform.

[17] Amazon Jsii - https://aws.github.io/jsii/

Η διαδικασία ανάπτυξης των πρωτοτύπων υπήρξε χρονοβόρα αλλά ήταν καθοριστική για την ομαλή υλοποίηση της κύριας λειτουργικότητας.

Ταυτόχρονα, έγινε σαφές πως το πιο χρονοβόρο και σύνθετο σκέλος της υλοποίησης
είναι η ανάπτυξη της επιχειρηματικής λογικής που διέπει τις διαδικασίες και όχι η ενσωμάτωση εξωτερικών εργαλείων. Η ανάπτυξη των επιχειρηματικών διαδικασιών καθώς και η υλοποίηση των οντοτήτων και των μεταξύ τους συσχετίσεων υπήρξε ιδιαίτερα σύνθετη, ειδικά εξαιτίας της χρήσης του Hibernate ORM, η εξοικείωση με το οποίο αποδείχθηκε ιδιαίτερα χρονοβόρα και σύνθετη καθώς έπρεπε να επιλυθούν ζητήματα όπως απομόνωση συναλλαγών (transaction isolation) και συνθήκες αγώνα (race conditions) που αφορούσαν την βάση δεδομένων και τις ασύγχρονες εκτελέσεις των αιτημάτων του χρήστη με τρόπο που δεν θα επηρεάζουν σημαντικά την απόδοση της εφαρμογής.

Επίσης, πρέπει να σημειωθεί πως ενώ η ενσωμάτωση των εξωτερικών συστημάτων γίνεται απλούστερη με την ανάπτυξη μικρότερων πρωτοτύπων, εξακολουθεί να είναι μια σύνθετη διαδικασία και ιδιαίτερα χρονοβόρα. Κάθε εργαλείο/σύστημα εκθέτει διαφορετικό API, αναπτύσσεται με διαφορετική άδεια λογισμικού και δεν είναι εφικτός ο διεξοδικός έλεγχος της λειτουργικότητας του καθώς η διαδικασία αυτή είναι χρονοβόρα, ειδικά στην περίπτωση εργαλείων/συστημάτων με ελλιπή ή δυσνόητη βιβλιογραφία. Ταυτόχρονα, η σωστή επιλογή του κατάλληλου εργαλείου είναι κρίσιμη καθώς η μετανάστευση από την χρήση ενός εργαλείου σε ένα άλλο είναι μια ιδιαίτερα χρονοβόρα και σύνθετη διαδικασία.

Μια ακόμα πτυχή της υλοποίησης που έγινε εμφανής ήταν η χρήση των διαφορετικών API’s που παρέχουν τόσο η Amazon, όσο και η Google. Οι διαφορές στον τρόπο λειτουργίας των δύο παρόχων, παρουσιάζουν σημαντικές δυσκολίες στην προσπάθεια ενοποίησης της λειτουργικότητας τους με τέτοιο τρόπο έτσι ώστε ο τελικός χρήστης να τις αντιμετωπίζει σαν ενιαία πλατφόρμα. Το πιο απλό παράδειγμα του γεγονότος αυτού είναι η διαφορά στην διαδικασία αυθεντικοποίησης ανάμεσα στους δύο παρόχους, με την Google να χρησιμοποιεί ένα json keyfile ενώ η Amazon χρησιμοποιεί ένα ζεύγος κλειδιών.

Παράλληλα, παρατηρήθηκε η ευκολία υλοποίησης δοκιμών με τα εργαλεία Rest Assured και JMeter καθώς και η φορητότητα των δοκιμών ανάμεσα σε παρόχους και συναρτήσεις.

5.4 Μελλοντικές κατευθύνσεις

Η εφαρμογή μπορεί να τροποποιηθεί έτσι ώστε να καλύψει περισσότερους παρόχους αλλά και πιο σύνθετες serverless εφαρμογές. Παράλληλα, ο Ίκαρος θα μπορούσε να υποστηρίζει στρατηγικές βελτιστοποίησης των συναρτήσεων της εκάστοτε πλατφόρμας (πχ. Performance Optimizer στην περίπτωση της Lambda), την δημιουργία εξατομικευμένων μετρικών από τον κάθε χρήστη αλλά και την σύγκριση ανάμεσα σε ιδιόκτητα εναύσματα ανάμεσα σε παρόχους (πχ. σύγκριση ιδιόκτητων Message Queues και database triggers). Ταυτόχρονα, η εφαρμογή θα μπορούσε να κάνει χρήση της υπηρεσίας ElasticSearch έτσι ώστε να είναι ευκολότερη η δημιουργία αναφορών και η παραγωγή μοντέλων απόδοσης. Με την ενσωμάτωση του ElasticSearch, επίσης, θα δοθεί μεγαλύτερη σημασία στην χρήση της εφαρμογής για ανάπτυξη μοντέλων απόδοσης και όχι μόνο για την συγγραφή δοκιμών.

Επίσης, είναι εφικτή η μετατροπή της εφαρμογής από N-Tier σε serverless και εν συνεχεία διάταξη της στο AWS Serverless Application Repository αλλά και η δοχειοποίηση της εφαρμογής ως native image. Η μετατροπή της εφαρμογής σε μια serverless αρχιτεκτονική προσφέρει μεγαλύτερη απόδοση αλλά και ευκολία διάταξης σε υπάρχων πλατφόρμες, ιδίως της Lambda, ενώ ταυτόχρονα η δοχειοποίηση ως native image μειώνει τους πόρους που χρησιμοποιεί η εφαρμογή αλλά και τον χρόνο εκκίνησης, με αποτέλεσμα μικρότερα startup latencies και cold starts.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] H. B. Hassan, S. A. Barakat, and Q. I. Sarhan. "Survey on serverless computing." *Journal of Cloud Computing* 10.1 (2021): 1-29.

https://journalofcloudcomputing.springeropen.com/articles/10.1186/s13677-021-00253-7

[2] P. Mell, and T. Grance. "The nist definition of cloud computing." *National Institute of Science and Technology, Special Publication* 800.2011 (2011): 145.

https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-145.pdf

[3] T. Erl and E. Monroy. “Fundamental Concepts and Models” *Cloud Computing: Concepts, Technology, Security, and Architecture*. 2nd ed., Pearson, 2023. Ch. 4. Print.

https://www.pearson.com/en-us/subject-catalog/p/cloud-computing-concepts-technology-and-architecture-second-edition/P200000009788/9780138052188

[4] F. Liu, J. Tong, J. Mao, R. B. Bohn, J. V. Messina, M. L. Badger and D. M. Leaf. "NIST cloud computing reference architecture." *NIST special publication* 500.2011 (2011): 1-28.

https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication500-292.pdf

[5] S. Kounev, N.Herbst, C. L. Abad, A. Iosup, I. Foster, P. Shenoy, O. Rana and A. A. Chien. "Serverless computing: What it is, and what it is not?." *Communications of the ACM* 66.9 (2023): 80-92.

https://dl.acm.org/doi/abs/10.1145/3587249

[6] S. Eismann. “Fundamentals”. *Performance Engineering of Serverless Applications and Platforms*. Diss. Universität Würzburg, 2023 Ch. 2..

https://nbn-resolving.org/urn:nbn:de:bvb:20-opus-303134

[7] V. Lenarduzzi and A. Panichella. (2020). Serverless Testing: Tool Vendors' and Experts' Point of View. IEEE Software. PP. 10.1109/MS.2020.3030803.

https://ieeexplore.ieee.org/document/9222011

[8] D. Dickerson, C. Gardner, C. Marcin and K.Hartig. *The Forrester Wave™: Functions-As-A-Service Platforms, Q2 2023*. Forrester Research, 6 June 2023

https://reprints2.forrester.com/#/assets/2/374/RES178501/report?refid=1d6398e7-13f6-4a1f-b63d-46ad9a9f04d1

[9] S. Eismann, J. Scheuner, E. Eyk and M. Schwinger. "The state of serverless applications: Collection, characterization, and community consensus." *IEEE Transactions on Software Engineering* 48.10 (2021): 4152-4166.

https://ieeexplore.ieee.org/abstract/document/9543531

[10] D. Raham, R. Black, and E. van Veenendaal. *Foundations of Software Testing: ISTQB Certification*. 4th ed., Cengage, 2019

https://www.cengage.uk/c/co/9781473764798/

[11] K. Yorkston. *Performance Testing*. Apress, 2021.

https://link.springer.com/content/pdf/10.1007/978-1-4842-7255-8.pdf