ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΙΓΑΙΟΥ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΑΚΩΝ ΚΑΙ ΕΠΙΚΟΙΝΩΝΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Υλοποίηση Ροών Εργασίας μέσω της Χρήσης Συναρτήσεων χωρίς Εξυπηρετητές

(Realizing Workflows Through the Use of Serverless Functions)

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

Γαλιατσάτου Παντελεήμων-Ιάσων (icsd13033)

Επιβλέπων: Κρητικός Κυριάκος

Μέλη εξεταστικής επιτροπής: Χαραλαμπίδης Γιάννης, Σκούτας Δημήτριος

Σάμος 2026

Πίνακας Περιεχομένων

1. Εισαγωγή 8

1.1. Περιγραφή του προβλήματος 8

1.2. Προτεινόμενη Λύση 8

1.3. Δομή της αναφοράς 10

2. Υπόβαθρο 11

2.1. Επιχειρησιακή Διαδικασία (Business Process) και Ροές Εργασίας 11

2.2. Συστήματα Διαχείρισης Ροών Εργασίας 13

2.3. Υπολογισμός Χωρίς Εξυπηρετητές 15

3. Συγκριτική Ανάλυση Συναφών Εργασιών 21

4. Ανάπτυξη Λογισμικού 23

4.1. Ανάλυση Απαιτήσεων 23

4.2. Σχεδίαση 24

4.2.1. Διάγραμμα Περιβάλλοντος 25

4.2.2. Διάγραμμα Περιπτώσεων Χρήσης 26

4.2.3. Διάγραμμα Συστατικών 27

4.2.4. Διάγραμμα Ακολουθίας 29

4.2.5. Διάγραμμα Οντοτήτων Συσχετίσεων 32

4.3. Υλοποίηση 35

4.3.1. Υλοποίηση Αρχιτεκτονικής Συστήματος 35

4.3.2. Περιγραφή Επαναχρησιμοποιούμενων Εργαλείων & Συστημάτων 35

4.3.3. Κύριες Κλάσεις Υλοποίησης 39

4.4. Δοκιμή 47

4.5. Βαθμός Ικανοποίησης Απαιτήσεων 48

5. Εγκατάσταση και Επίδειξη Εφαρμογής 51

5.1. Εγκατάσταση Συστήματος 51

5.2. Επίδειξη Συστήματος 51

5.2.1. Χρήση του Serverless Framework 51

5.2.2. Δημιουργία Ροής Εργασίας με Serverless Script Tasks 54

5.2.3. Εκτέλεση Ροής Εργασίας με Serverless Script Tasks 71

6. Συμπεράσματα & Μελλοντική Εργασία 73

6.1. Συμπεράσματα 73

6.2. Μελλοντική Εργασία 73

Βιβλιογραφία 75

Λίστα Εικόνων

Εικόνα 1 - Παράδειγμα μιας ιδιωτικής επιχειρησιακής διαδικασίας 12
Εικόνα 2 - Αρχιτεκτονική Serverless Πλατφόρμας Apache OpenWhisk 20
Εικόνα 3 - Διάγραμμα περιβάλλοντος 26
Εικόνα 4 - Διάγραμμα Περιπτώσεων Χρήσης 27
Εικόνα 5 - Διάγραμμα Συστατικών (Component Diagram) 29
Εικόνα 6 - Διάγραμμα Ακολουθίας (Create/Update Workflow, Create/Update Serverless Functions) 30
Εικόνα 7 - Διάγραμμα Ακολουθίας (Execute Workflow, Execute Serverless Functions) 32
Εικόνα 8 - Διάγραμμα Οντοτήτων-Συσχετίσεων 34
Εικόνα 9 - Κλάση SslScriptExecutionListener (Μέρος Πρώτο) 39
Εικόνα 10 - Κλάση SLSScriptExecutionListener (Μέρος Δεύτερο) 40
Εικόνα 11 - Κλάση SLSPlatformService 41
Εικόνα 12 - Φάκελος με το αρχεία που χρειάζεται το Serverless Framework και σχετίζονται με τον ορισμό εφαρμογής με αναγνωριστικό test-app 42
Εικόνα 13 - Κλάση SLSConfigService (μέρος πρώτο) 43
Εικόνα 14 - Κλάση SLSConfigService (μέρος δεύτερο) 44
Εικόνα 15 - Κλάση SLSConfigService (μέρος τρίτο) 44
Εικόνα 16 - Μέθοδοι που περιέχει η κλάση ScriptTaskParseService. 45
Εικόνα 17 - Μέθοδος validateAndProcessScriptTask της κλάσης ScriptTaskParseService. 46
Εικόνα 18 - Template αρχείο διαμόρφωσης για την πλατφόρμα AWS Lambda 53
Εικόνα 19 - Template αρχείο διαμόρφωσης για τον cloud πάροχο Google. 53
Εικόνα 20 - Template αρχείο διαμόρφωσης για τον cloud πάροχο Openwhisk. 54
Εικόνα 21 - Πληροφορίες που απαρτίζουν ένα script task στο Flowable 55
Εικόνα 22 - Αναδυόμενο παράθυρο για εισαγωγή/τροποποίηση των Execution Listeners καθώς και των μεταβλητών τους. 57
Εικόνα 23 – Μενού ορισμενων εφαρμογων (App Definitions) 58
Εικόνα 24 - Αναδυόμενο παράθυρο ορισμού εφαρμογής 58
Εικόνα 25 - Μενού ορισμένων ροών εργασίασ 58
Εικόνα 26 - Αναδυόμενο παράθυρο δημιουργίας ροής εργασίας 59
Εικόνα 27 - Δημιουργημένη ροή εργασίας «Test Model» 60
Εικόνα 28 - Visual editor ροής εργασίας 60
Εικόνα 29 - Δημιουργία γεγονότος εκκινησης (start event signal) 60
Εικόνα 30 - Κώδικας 1oy Script Task 61
Εικόνα 31 - Γραφική διεπαφή για την δημιουργία ροής εργασίας 62
Εικόνα 32 - Execution Listener για την serverless συνάρτηση του 1ου script task 62
Εικόνα 33 - Κώδικας 2ου script task 63
Εικόνα 34 - Execution Listener για την serverless συνάρτηση του 2ου script task 64
Εικόνα 35 - Κώδικας 3ου script task 65
Εικόνα 36 - Execution Listener για την serverless συνάρτηση του 3ου script task 66
Εικόνα 37 - Δημιουργία τελικού γεγονότος (end event signal) 66
Εικόνα 38 - Αποθήκευση ροής εργασίας 67
Εικόνα 39 - Αποδοχή αποθηκευσης της ροής εργασίας 67
Εικόνα 40 - Τροποποίηση ορισμού εφαρμογής 68
Εικόνα 41 - Αντιστοίχιση ροών εργασίας σε ορισμό εφαρμογής (1) 68

Εικόνα 42 - Αντιστοίχιση ροών εργασίας σε ορισμό εφαρμογής (2) 69
Εικόνα 43 - Μενού διαχείρισης ορισμού εφαρμογής 69
Εικόνα 44 - Παραγμένα αρχεία κώδικα & διαμόρφωσης 70
Εικόνα 45 - Απόσπασμα επιτυχημένου πακεταρίσματος και διάταξης (μέρος 1^{ο}) 70
Εικόνα 46 - Απόσπασμα επιτυχημένου πακεταρίσματος και διάταξης (μέρος 2^{ο}) 70
Εικόνα 47 - Απόσπασμα serverless συνάρτησης του task1 σε aws lambda 71
Εικόνα 48 - Απόσπασμα serverless συνάρτησης του task2 στο google cloud 71
Εικόνα 49 - Εκτέλεση της serverless συνάρτησης (task1) 71
Εικόνα 50 - Εκτέλεση της serverless συνάρτησης (task2) 72
Εικόνα 51 - Εκτέλεση της serverless συνάρτησης (task3) 72

Λίστα Πινάκων

Πίνακας 1 - Συγκριτική Παρουσίαση Δημοφιλών Συστημάτων Διαχείρισης Ροών Εργασίας 15
Πίνακας 2 - Σύγκριση Βασικών Χαρακτηριστικών των Τεσσάρων Serverless Πλατφορμών 18
Πίνακας 3 - Βαθμός Ικανοποίησης Απαιτήσεων 48

Περίληψη

Στόχος της διπλωματικής είναι να αντιμετωπίσει κάποιους από τους περιορισμούς που εμφανίζονται κατά την υλοποίηση ροών εργασίας (workflows) με τη χρήση αυτοματοποιημένων scripts στις λειτουργίες (tasks) τους. Αυτό το επιτυγχάνει χρησιμοποιώντας συναρτήσεις χωρίς εξυπηρετητές (serverless functions). Η παραδοσιακή υλοποίηση των ροών εργασίας όσον αφορά τις λειτουργίες (tasks) τους γίνεται γράφοντας scripts μέσα στον ορισμό τους σε κάποια scripting γλώσσα προγραμματισμού (δηλαδή γλώσσα σεναρίου) που υποστηρίζει ένα σύστημα διαχείρισης ροών εργασίας. Εξ ορισμού οι δυνατότητες που ανέρχονται από την χρήση αυτοματοποιημένων scripts στις λειτουργίες εξαρτώνται πλήρως από το ίδιο το σύστημα (πχ. μόνο χρήση συγκεκριμένης γλώσσας scripting) ενώ η επαναχρησιμοποίησή τους γίνεται χειρωνακτικά με κίνδυνο εμφάνισης λαθών κατά την αντιγραφή ή και αντιγραφής λανθασμένου/αναξιόπιστου κώδικα. Η χρήση των serverless συναρτήσεων εξαλείφει αυτούς τους περιορισμούς και κινδύνους, η λειτουργικότητά τους μπορεί να επαναχρησιμοποιείται ενώ προσφέρονται επιπλέον δυνατότητες, όπως υποστήριξη ποικιλίας γλωσσών προγραμματισμού, οικονομική αποδοτικότητα, αυτόματη κλιμάκωση, μεταφορά υπολογιστικού φόρτου σε παρόχους νέφους, επαναχρησιμοποίηση σε διαφορετικές ροές και ευκολία ανάπτυξης και ενημέρωσης.

Η συνεισφορά της εργασίας έγκειται στην ανάπτυξη μιας επέκτασης για ένα συγκεκριμένο σύστημα διαχείρισης ροών εργασίας, το Flowable, που επιτρέπει την αυτοματοποιημένη ενσωμάτωση, πακετάρισμα και εκτέλεση serverless συναρτήσεων μέσω του Serverless Framework, με πλεονεκτήματα, εκτός από τα προαναφερόμενα γενικής φύσεως, την υποστήριξη πολλαπλών πλατφορμών (AWS, GCP, OpenWhisk), τη δυνατότητα τοπικής/απομακρυσμένης εκτέλεσης (των συναρτήσεων) και την αποφυγή επανάληψης κώδικα (σε πολλαπλούς ορισμούς ροών εργασίας) μέσω κεντρικού ορισμού (της εκάστοτε συνάρτησης που τον υλοποιεί).

Λέξεις Κλειδιά: συναρτήσεις χωρίς εξυπηρετητές, πλαίσιο Serverless, πολυπλατφορμική υποστήριξη, συστήματα διαχείρισης ροών εργασίας, Flowable, BPMN, επαναχρησιμοποίηση συναρτήσεων

Summary

The goal of this thesis is to address some of the limitations that arise during the implementation of workflows when automated scripts are used within their tasks. This is achieved by utilizing serverless functions. Traditionally, the implementation of workflow tasks involves writing scripts within their definitions using a scripting language supported by the workflow management system. By definition, the capabilities offered by such automated scripts depend entirely on the system itself (e.g., limited to a specific scripting language), while their reuse must be performed manually introducing the risk of errors during copying or even the reuse of faulty or unreliable code. The use of serverless functions eliminates these limitations and risks, allowing their functionality to be reused while offering additional benefits, such as support for multiple programming languages, cost efficiency, automatic scaling, offloading computational workloads to cloud providers, reuse across different workflows, and ease of deployment and maintenance.

The contribution of this thesis lies in the development of an extension for a workflow management system called Flowable that enables the automated integration, packaging, and execution of serverless functions through the Serverless Framework. Beyond the general advantages mentioned above, this approach features multi-platform support (AWS, GCP, OpenWhisk), allows both local and remote function execution, and avoids code duplication across multiple workflow definitions by introducing a centralized definition of each function implementation.

Keywords: serverless functions, Serverless framework, multi-platform support, workflow management systems, Flowable, BPMN, function re-use

1. Εισαγωγή

1.1. Περιγραφή του προβλήματος

Η επιχειρησιακή διαδικασία (business process) είναι μια συλλογή επιχειρησιακών καθηκόντων και δραστηριοτήτων (activities) που όταν εκτελούνται από άτομα ή συστήματα με ένα δομημένο τρόπο παράγουν ένα αποτέλεσμα που συμβάλλει στους επιχειρησιακούς στόχους μιας εταιρείας ή οργανισμού. Οι επιχειρησιακές διαδικασίες υλοποιούνται από ροές εργασίας (workflows) με ένα πιο λεπτομερή τρόπο, όπου μια δραστηριότητα της διαδικασίας μπορεί να αντιστοιχίζεται σε μια σύνθεση λειτουργιών (tasks) στην αντίστοιχη ροή εργασίας.

Συνήθως, το άτομο που είναι υπεύθυνο για τη μοντελοποίηση ροών εργασίας (workflows) προσπαθεί να την αυτοματοποιήσει όσο το δυνατόν περισσότερο ώστε να αυξήσει την απόδοση στην εκτέλεσή της και να μειώσει το αντίστοιχο κόστος. Προς αυτό τον σκοπό, ενδέχεται να υλοποιήσει ή να επαναχρησιμοποιήσει υπηρεσίες ιστού (web services) και προγράμματα σεναρίου (scripts) για την υλοποίηση ορισμένων από τις λειτουργίες της ροής εργασίας. Η ανάπτυξη και των δύο αυτών τύπων συστατικών δεσμεύεται στην χρήση μιας ή περισσοτέρων γλωσσών προγραμματισμού (πχ. αντικειμενοστρεφούς για την υλοποίηση των υπηρεσιών ιστού και σεναρίου για την υλοποίηση προγραμμάτων σεναρίου). Αυτό δημιουργεί υψηλές απαιτήσεις ως προς τις γνώσεις και ικανότητες προγραμματισμού του εν λόγω ατόμου. Επιπλέον, εστιάζοντας στο δεύτερο τύπο συστατικών, τα προγράμματα σεναρίου, δημιουργούνται επιπρόσθετα προβλήματα διότι: (α) ο κώδικας του προγράμματος μπορεί να επαναλαμβάνεται σε διάφορα μέρη της ίδιας ή και διαφορετικών ροών εργασίας, αυξάνοντας τον φόρτο για τον μοντελοποιητή και (β) είναι δύσκολος ο εντοπισμός και η επαναχρησιμοποίηση των προγραμμάτων σεναρίων διότι αυτά “κρύβονται” μέσα στους ορισμούς των ροών εργασίας και είναι δύσκολο να ανακαλυφθούν. Επιπλέον, η εκτέλεση των προγραμμάτων σεναρίων αυξάνει το φόρτο του συστήματος διαχείρισης ροών εργασίας σε αντίθεση με τη χρήση των υπηρεσιών ιστού, οι οποίες αποτελούν λογισμικό που εκτελείται απομακρυσμένα.

1.2. Προτεινόμενη Λύση

Για την καταπολέμηση των προβλημάτων που ανέρχονται από την εκτέλεση προγραμμάτων σεναρίων μπορεί να χρησιμοποιηθεί ο υπολογισμός χωρίς διακομιστές (serverless computing), ένα μοντέλο του υπολογιστικού νέφους όπου ο πάροχος αναλαμβάνει τη διαχείριση των διακομιστών και την εκτέλεση του κώδικα κατ΄απαίτηση, χωρίς ο προγραμματιστής να χρειάζεται να ασχοληθεί με την υποδομή. Σε αυτό το μοντέλο, οι προγραμματιστές μπορούν να αναπτύσσουν και να εκτελούν εφαρμογές (συνήθως ονομάζονται συναρτήσεις - functions) χωρίς να διαχειρίζονται διακομιστές, πληρώνοντας μόνο για τους (πραγματικούς) πόρους που χρησιμοποιούνται, ενώ ο πάροχος υπηρεσιών νέφους (cloud) αναλαμβάνει την προμήθεια, διαχείριση και κλιμάκωση της υποδομής αλλά και την εκτέλεση και κλιμάκωση των αντίστοιχων εφαρμογών/συναρτήσεων πάνω από αυτή την υποδομή. Τα βασικά συστατικά του περιλαμβάνουν το Function-as-a-Service (FaaS) για την εκτέλεση μεμονωμένων συναρτήσεων ως απόκριση σε συμβάντα (events) και το Backend-as-a-Service (BaaS), το οποίο παρέχει έτοιμες συνοδευτικές υπηρεσίες, όπως αυτόματα κλιμακούμενες βάσεις δεδομένων, συστήματα μηνυματοδότησης, υπηρεσίες αυθεντικοποίησης και API Gateways (πύλες προγραμματιστικών διεπαφών). **[1]**

Με την χρήση του υπολογισμού χωρίς εξυπηρετητές και ειδικότερα των συναρτήσεων χωρίς εξυπηρετητές (serverless functions) αντιμετωπίζονται τα προαναφερόμενα προβλήματα. Κι αυτό διότι οι serverless συναρτήσεις αποτελούν κι αυτές απομακρυσμένο λογισμικό προς εκτέλεση, το οποίο έχει την ικανότητα να κλιμακώνεται πιο γρήγορα σε σχέση με τις υπηρεσίες ιστού. Επιπλέον, το λογισμικό αυτό μπορεί να συγγραφεί σε οποιαδήποτε γλώσσα προγραμματισμού - αυτό δίνει την ευχέρεια στην ομάδα έργου να χρησιμοποιήσει την γλώσσα προγραμματισμού της αρεσκείας της, η οποία ενδέχεται να χρησιμοποιείται και για την ανάπτυξη ιδίων υπηρεσιών ιστού. Επιπλέον, υπάρχει

αρκετή ευελιξία διότι η συγγραφή κάθε serverless συνάρτησης μπορεί να γίνεται σε διαφορετική γλώσσα, πχ. ανάλογα με τις προγραμματιστικές ικανότητες του μέλους της ομάδας έργου που ανατίθεται να την υλοποιήσει. Ακόμη, το λογισμικό αυτό είναι πιο εύκολο στον εντοπισμό εφόσον συγγράφεται ξεχωριστά από την ροή εργασίας ενώ μπορεί να χρησιμοποιείται πολλαπλές φορές στο πλαίσιο μιας ροής εργασίας ή και διαφορετικών για την υλοποίηση συγκεκριμένης λειτουργικότητας. Επιπρόσθετα, το λογισμικό αυτό συγγράφεται ανεξάρτητα από τη ροή εργασίας, επιτρέποντας την ανεξάρτητη διατήρησή του από αυτήν. Τέλος, η χρήση των serverless συναρτήσεων επιτρέπει την περαιτέρω αποφόρτιση του συστήματος διαχείρισης ροών εργασίας, πράγμα που μπορεί να συντελέσει στη καλύτερη απόδοση και κλιμακωσιμότητά του.

Για τη διάταξη και το πακετάρισμα των συναρτήσεων χωρίς εξυπηρετητές, μπορούν να χρησιμοποιηθούν δύο προσεγγίσεις. Η πρώτη βασίζεται σε πλαίσια (frameworks) ειδικά σχεδιασμένα για έναν μόνο πάροχο υποδομής, όπως το AWS SAM[1] για την AWS ή το Google Cloud Functions[2] για την Google, οπότε προσφέρει βαθύτερη ενσωμάτωση με τη συγκεκριμένη serverless πλατφόρμα που προσφέρεται (από τον αντίστοιχο πάροχο νέφους). Ωστόσο, αυτή η προσέγγιση οδηγεί σε δέσμευση με έναν και μόνο πάροχο (vendor lock-in), γεγονός που περιορίζει την ευελιξία ενός οργανισμού. Αντίθετα, η δεύτερη προσέγγιση αξιοποιεί γενικά πλαίσια, όπως το **Serverless Framework[3],** τα οποία υποστηρίζουν πολλαπλές πλατφόρμες και επιτρέπουν την ανεξαρτησία από συγκεκριμένους παρόχους. Η χρήση ενός γενικού πλαισίου είναι ενδεδειγμένη ειδικά όταν ένας οργανισμός δεν επιθυμεί να κλειδωθεί σε έναν μόνο πάροχο, αλλά θέλει να έχει την ευχέρεια να χρησιμοποιεί διαφορετικές πλατφόρμες ακόμη και ταυτόχρονα και να μεταβαίνει από τη μία στην άλλη ανάλογα με τις ανάγκες ή τις συνθήκες της εκάστοτε χρονικής στιγμής.

Το Serverless Framework, το οποίο υιοθετείται στην παρούσα διπλωματική εργασία, προσφέρει μια κοινή διεπαφή για την διαχείριση serverless συναρτήσεων ανεξαρτήτως παρόχου. Παρέχει δυνατότητα διάταξης συναρτήσεων τόσο σε απομακρυσμένους παρόχους νέφους (AWS Lambda, Google Cloud Functions, OpenWhisk κ.λπ.) όσο σε τοπικά περιβάλλοντα ανάπτυξης, επιτρέποντας στους προγραμματιστές να ελέγχουν και να δοκιμάζουν τη συμπεριφορά των συναρτήσεων τους τοπικά, πριν τις αναπτύξουν σε παραγωγικό περιβάλλον (cloud provider) με υποστήριξη FaaS (Function as a Service).

Η παρούσα διπλωματική εργασία στηρίζεται ουσιαστικά σε αυτό το πλαίσιο ώστε να προτείνει και να υλοποιήσει μια επέκτασή του πάνω σε σύστημα διαχείρισης ροών εργασίας (WFMS), και συγκεκριμένα στο Flowable[4]. Η επέκταση αυτή επιτρέπει την ενσωμάτωση serverless συναρτήσεων σε λειτουργίες τύπου script task, με δυνατότητα συγγραφής του κώδικα σε πολλές γλώσσες προγραμματισμού, επικύρωσης της συνάρτησης (πχ. μέσω τοπικής της εκτέλεσης), αποθήκευσης και επαναχρησιμοποίησης της σε διαφορετικά σημεία της ίδιας ή και διαφορετικών ροών. Η συνάρτηση μπορεί να εκτελείται τοπικά ή απομακρυσμένα, ανάλογα με τις ρυθμίσεις και ο προγραμματιστής μπορεί ευέλικτα να ορίσει αυτές τις επιλογές είτε τοπικά ανα task είτε καθολικά για ολόκληρη την ροή.

[1] AWS SAM: Πλαίσιο ανοικτού κώδικα της Amazon Web Services για την ανάπτυξη εφαρμογών χωρίς εξυπηρετητές (serverless) (https://aws.amazon.com/serverless/sam/)

[2] Google Cloud Functions: Υπηρεσία υπολογισμού χωρίς εξυπηρετητές της Google Cloud Platform, που επιτρέπει την ανάπτυξη και εκτέλεση μεμονωμένων συναρτήσεων ως απόκριση σε συμβάντα (events) ή κλήσεις HTTP, χωρίς διαχείριση υποδομής (https://cloud.google.com/functions?hl=en)

[3] Serverless Framework**:** Πλαίσιο ανοικτού κώδικα για την ανάπτυξη και διαχείριση εφαρμογών χωρίς εξυπηρετητές (serverless) σε διάφορες πλατφόρμες υπολογιστικού νέφους (https://www.serverless.com).

[4] Flowable: Μηχανή και πλατφόρμα ανοικτού κώδικα για την μοντελοποίηση και εκτέλεση επιχειρησιακών διαδικασιών (BPM) με υποστήριξη των προτύπων BPMN, CMMN και DMN (**https://www.flowable.com**).

1.3. Δομή της αναφοράς

Το υπόλοιπο μέρος της τρέχουσας αναφοράς οργανώνεται ως εξής: Στο Κεφάλαιο 2 παρουσιάζεται το θεωρητικό υπόβαθρο και οι βασικές τεχνολογίες που χρησιμοποιούνται από την εργασία μας, όπως οι ροές εργασίας, τα συστήματα WFMS και ο υπολογισμός χωρίς διακομιστές. Στο Κεφάλαιο 3 πραγματοποιείται συγκριτική ανάλυση συναφών εργασιών που σχετίζονται με το αντικείμενο της παρούσας εργασίας.

Το Κεφάλαιο 4 παρουσιάζεται η ανάπτυξη του συστήματος, η οποία πραγματοποιήθηκε με βάση το μοντέλο καταρράκτη, καλύπτοντας την ανάλυση απαιτήσεων, τη σχεδίαση, την υλοποίηση, τη δοκιμή του συστήματος καθώς και τον βαθμό ικανοποίησης των αρχικών απαιτήσεων. Στο Κεφάλαιο 5 παρέχονται οδηγίες εγκατάστασης του συστήματος και επιδεικνύεται η χρήση του μέσω ενός παραδείγματος που αφορά την εκτέλεση ροής εργασίας με serverless συναρτήσεις, αποδεικνύοντας τη λειτουργικότητα και επεκτασιμότητα του συστήματος. Τέλος, το Κεφάλαιο 6 περιλαμβάνει τα βασικά συμπεράσματα της εργασίας και προτείνει μελλοντικές κατευθύνσεις για περαιτέρω ανάπτυξη και βελτίωση του συστήματος που αναπτύχθηκε στο πλαίσιό της.

2. Υπόβαθρο

Σκοπός αυτού του κεφαλαίου είναι να παρουσιάσει κάποιες βασικές πληροφορίες για τις βασικές έννοιες και τεχνολογίες που σχετίζονται με το αντικείμενο της διπλωματικής εργασίας ώστε να γίνει κατανοητό τόσο το περιεχόμενο των επόμενων κεφαλαίων αλλά και η συνεισφορά και σημαντικότητα της εργασίας αυτής. Συγκεκριμένα θα αναλυθούν η έννοια της ροής εργασίας (workflows), τα συστήματα διαχείρισης ροών εργασίας (workflow management systems) και ο υπολογισμός χωρίς εξυπηρετητές (serverless computing).

2.1. Επιχειρησιακή Διαδικασία (Business Process) και Ροές Εργασίας

Μια επιχειρησιακή διαδικασία (business process) αποτελεί ένα σύνολο δομημένων και συσχετιζόμενων δραστηριοτήτων ή εργασιών που εκτελούνται από ανθρώπους ή συστήματα, με στόχο την επίτευξη ενός συγκεκριμένου επιχειρηματικού αποτελέσματος. Η επιχειρησιακή διαδικασία είναι μια σειρά βημάτων σχεδιασμένων για την παραγωγή ενός προϊόντος ή υπηρεσίας. Επίσης, περιγράφει το τι πρέπει να γίνει για τη επίτευξη ενός οργανωσιακού στόχου, και διακρίνεται από χαρακτηριστικά, όπως σαφή όρια, εισροές και εκροές, καθορισμένη σειρά και προσθήκη αξίας για τον πελάτη. [3]

Από την άλλη πλευρά, μια ροή εργασίας (workflow) αποτελεί την αλληλουχία των βημάτων που απαιτούνται για την εκτέλεση μιας συγκεκριμένης εργασίας. Ενώ η επιχειρησιακή διαδικασία καθορίζει το “τι” και το “γιατί” πίσω από τις επιχειρηματικές δραστηριότητες, η ροή εργασίας επικεντρώνεται στο “πώς”, συνεπώς είναι η λεπτομερής αλληλουχία των δραστηριοτήτων που πρέπει να εκτελεστούν για την ολοκλήρωση μιας μεμονωμένης διαδικασίας. Ο πιο σημαντικός διαχωρισμός μεταξύ των δύο είναι ο σκοπός τους: ο σκοπός μιας ροής εργασίας είναι η ολοκλήρωση μιας εργασίας, ενώ ο σκοπός μιας διαδικασίας είναι η επίτευξη ενός οργανωσιακού στόχου. [4] [5]

Αυτή η διάκριση είναι σημαντική για την κατανόηση του τρόπου με τον οποίο οι οργανισμοί λειτουργούν, βελτιστοποιούν τις λειτουργίες τους και επιτυγχάνουν τους στόχους τους. Οι επιχειρησιακές διαδικασίες μπορεί να περιλαμβάνουν πολλαπλές ροές εργασίας για την επίτευξη του επιθυμητού αποτελέσματος, ενώ οι ροές εργασίας εστιάζουν στα λεπτομερή βήματα που απαιτούνται για την ολοκλήρωση συγκεκριμένων εργασιών εντός μιας διαδικασίας.

Οι ροές εργασίας μοντελοποιούνται, εκτελούνται, διατηρούνται και εν γένει διαχειρίζονται από **συστήματα διαχείρισης ροών εργασίας (Workflow Management Systems)**. Επιπλέον οι ροές εργασίας μοντελοποιούνται με την χρήση κάποιων προτύπων. Αυτά τα πρότυπα παρέχουν στις επιχειρήσεις τη δυνατότητα να κατανοούν τις εσωτερικές επιχειρηματικές διαδικασίες μέσω διαγραμμάτων. Κάποια από αυτά τα πρότυπα είναι το **BPMN (Business Process Model and Notation)**, το **CMMN (Case Management Model and Notation)**, και το **DMN (Decision Model and Notation)**, τα οποία αναλύονται συντόμως παρακάτω. [6] [7] [8]

Κάθε διαδικασία περιγράφει τη ροή εργασιών, καθορίζοντας tasks, αποφάσεις, γεγονότα (events) και μεταβάσεις (transitions).

Το BPMN αποτελεί το διεθνές πρότυπο για την μοντελοποίηση επιχειρησιακών διαδικασιών, το οποίο έχει αναπτυχθεί από τον οργανισμό Object Management Group (OMG). Επιπλέον, δύο νέα πρότυπα με παρόμοια προσέγγιση με το BPMN έχουν αναπτυχθεί, που αφορούν τη μοντελοποίηση διαχείρισης περιπτώσεων (CMMN) και την μοντελοποίηση αποφάσεων (DMN). Η διαχείριση περιπτώσεων (Case Management) αφορά διαδικασίες που δεν ακολουθούν αυστηρά προκαθορισμένη ροή, αλλά προσαρμόζονται δυναμικά ανάλογα με τα δεδομένα και τις συνθήκες κάθε περίπτωσης, με το πρότυπο CMMN να επιτρέπει τη μοντελοποίηση τους. Αντίστοιχα, το DMN

επικεντρώνεται στη μοντελοποίηση των κανόνων και λογικών αποφάσεων που υποστηρίζουν τις επιχειρησιακές ροές. Τα πρότυπα BPMN, CMMN, DMN λειτουργούν συμπληρωματικά, καθώς περιγράφουν αντίστοιχα το πώς εκτελείται μια διαδικασία, πότε εκτελούνται ενέργειες ανάλογα με το πλαίσιο και τι αποφάσεις λαμβάνονται σε κάθε στάδιο.

Στην Εικόνα 1 απεικονίζεται μια ιδιωτική επιχειρησιακή διαδικασία ασφαλιστικής εταιρείας μοντελοποιημένη με το πρότυπο BMPN. Η διαδικασία αυτή αφορά την έγκριση μιας ασφαλιστικής πολιτικής, που περιλαμβάνει πέντε βασικά βήματα:

1. Προσδιορισμός εάν η παραγγελία είναι πλήρης (Determine Order is Complete)
2. Έλεγχος των στοιχείων του αιτούντος (Check Record of Applicant)
3. Καθορισμός του ασφαλίστρου (Determine Premium of Policy)
4. Έγκριση ή απόρριψη της πολιτικής (Approve or Reject Policy)
5. Ενημέρωση του αιτούντος για την έγκριση ή απόρριψη (Notify Applicant of Approval or Rejection)

Αυτή η διαδικασία αποτελεί ένα τυπικό παράδειγμα ροής που μπορεί να μοντελοποιηθεί με το πρότυπο BPMN. Το BPMN χρησιμοποιεί τυποποιημένα σύμβολα, όπως δραστηριότητες (απεικονίζονται ως παραλληλόγραμμα με στρογγυλεμένες γωνίες), γεγονότα (απεικονίζονται ως κύκλοι), και πύλες αποφάσεων (απεικονίζονται ως ρόμβοι).

Με την χρήση τέτοιων προτύπων οι επιχειρήσεις μπορούν να απεικονίσουν με σαφήνεια τις διαδικασίες τους, να εντοπίσουν σημεία βελτίωσης, και να επικοινωνήσουν αποτελεσματικά τις ροές εργασίας μεταξύ διαφορετικών τμημάτων και εμπλεκόμενων μερών (stakeholders). Αυτό διευκολύνει την αυτοματοποίηση των διαδικασιών και την αποτελεσματικότερη διαχείριση των επιχειρησιακών λειτουργιών

ΕΙΚΟΝΑ 1 - ΠΑΡΑΔΕΙΓΜΑ ΜΙΑΣ ΙΔΙΩΤΙΚΗΣ ΕΠΙΧΕΙΡΗΣΙΑΚΗΣ ΔΙΑΔΙΚΑΣΙΑΣ

Μετά την ανάλυση του μοντέλου στην Εικόνα 1 παρατηρείται ότι σε μια ροή εργασίας μπορούν να ενσωματωθούν διάφορα είδη βημάτων, όπως τα βήματα χρήστη (user tasks) (πχ. το πρώτο, δεύτερο και τέταρτο), τα αυτοματοποιημένα βήματα υπηρεσίας (service tasks) και τα βήματα σεναρίου (script tasks) (όπως φαίνεται στο 3ο, το οποίο είναι service task). Τα δύο τελευταία είδη σχετίζονται με την αυτοματοποιημένη εκτέλεση λειτουργιών. Τα service tasks χρησιμοποιούνται συνήθως για την κλήση εξωτερικών υπηρεσιών μέσω πρωτοκόλλων, όπως REST και SOAP. Από την άλλη μεριά, τα script tasks αποτελούν έναν τύπο αυτοματοποιημένου βήματος στο BPMN, στο οποίο το σύστημα διαχείρισης ροών εργασίας εκτελεί απευθείας ενσωματωμένο κώδικα χωρίς την ανάγκη κλήσης εξωτερικής υπηρεσίας. Υποστηρίζουν γλώσσες σεναρίου, όπως Javascript και Groovy, επιτρέποντας την υλοποίηση λογικής εντός της ίδιας της ροής εργασίας.

Σε αντίθεση με τα script tasks, όπου ο κώδικας εκτελείται απευθείας από το ίδιο το σύστημα διαχείρισης ροών εργασίας (WFMS), οι υπηρεσίες ιστού που υλοποιούν μέρη μιας ροής εργασίας εκτελούνται σε ένα εξωτερικό, απομακρυσμένο σύστημα και καλούνται μέσω πρωτοκόλλων όπως REST ή SOAP. Το WFMS δεν έχει ορατότητα στην εσωτερική εκτέλεση της υπηρεσίας αλλά παρακολουθεί και διαχειρίζεται μόνο το αποτέλεσμα της κλήσης (επιτυχία, σφάλμα, timeout) στο

όριο της επικοινωνίας, σε αντίθεση με τα script tasks όπου η ίδια μηχανή γνωρίζει και ελέγχει πλήρως την εκτέλεση του κώδικα.

2.2. Συστήματα Διαχείρισης Ροών Εργασίας

Ένα σύστημα διαχείρισης ροών εργασίας (WFMS) είναι ένα πληροφοριακό σύστημα που χρησιμοποιείται για την ανάπτυξη/μοντελοποίηση, την υλοποίηση, την εκτέλεση και την παρακολούθηση μιας καθορισμένης ακολουθίας εργασιών που είναι διατεταγμένες ως μια ροή εργασίας. Το WFMS επιτρέπει στον χρήστη να ορίζει διαφορετικές ροές εργασίας (workflows) για διαφορετικούς τύπους εργασιών ή διεργασιών. Αξίζει να αναφέρουμε ότι αυτές οι ροές εργασίας (workflows) που μπορούν να δημιουργηθούν μπορεί να είναι πλήρως αυτοματοποιημένες, δηλαδή να εκτελούνται αυτόματα δηλαδή ή να απαιτούν την αλληλεπίδραση με τον χρήστη οπότε να είναι χειρωνακτικές ή ενδέχεται να υποστηρίζεται ένας συνδυασμός αυτών των δύο (οπότε η ροή είναι εν μέρει αυτοματοποιημένη). Αυτά τα συστήματα δίνουν την δυνατότητα για την μοντελοποίηση ροών εργασιών με την χρήση ενός (ή και παραπάνω) από τα πρότυπα που προαναφέρθηκαν. [7]

Τα βασικά λειτουργικά χαρακτηριστικά που προσφέρονται από τα συστήματα διαχείρισης ροών εργασίας περιλαμβάνουν:

1. **Μοντελοποίηση και Σχεδιασμός ροών εργασίας:** Παρέχουν γραφικά εργαλεία σχεδιασμού που επιτρέπουν στους χρήστες να μοντελοποιούν ροές εργασίας με τη χρήση προτύπων, όπως το BPMN, δημιουργώντας οπτικές αναπαραστάσεις των διαδικασιών με δραστηριότητες, γεγονότα, πύλες αποφάσεων και άλλα στοιχεία.
2. **Δημιουργία και διαχείριση περιπτώσεων ροής εργασίας (workflow instances):** Μια περίπτωση ροής εργασίας είναι “κάθε στιγμιότυπο εκτέλεσης μιας διαδικασίας/ροής εργασίας που διαχειρίζεται το WFMS”. Κάθε φορά που εκκινείται μια ροή εργασίας, δημιουργείται μια περίπτωση ροής εργασίας που παρακολουθείται και διαχειρίζεται ξεχωριστά από το σύστημα. Για παράδειγμα, σε μια διαδικασία έγκρισης άδειας, κάθε αίτηση άδειας αποτελεί μια ξεχωριστή περίπτωση της ίδιας ροής εργασίας.
3. **Εκτέλεση ροών εργασίας:** Η μηχανή ροής εργασίας (workflow engine) δημιουργεί ένα στιγμιότυπο εκτέλεσης για κάθε ροή εργασίας και ενορχηστρώνει την εκτέλεση των διαφορετικών εργασιών και δραστηριοτήτων (που ενσωματώνονται σε αυτήν) στη σωστή σειρά. Αυτό περιλαμβάνει τη δρομολόγηση εργασιών στους κατάλληλους χρήστες ή συστήματα, τη διαχείριση των εξαρτήσεων μεταξύ των εργασιών και τη διασφάλιση ότι κάθε βήμα εκτελείται με τη σωστή σειρά.
4. **Παρακολούθηση και ανάλυση:** Τα συστήματα παρέχουν αναλυτικές πληροφορίες για την κατάσταση των εργασιών, τον χρόνο εκτέλεσης, τους εμπλεκόμενους πόρους και άλλες μετρήσεις απόδοσης, επιτρέποντας στους διαχειριστές να εντοπίζουν σημεία συμφόρησης ή καθυστερήσεις.
5. **Διαχείριση δεδομένων και εγγραφών:** Τα WFMS συχνά περιλαμβάνουν λειτουργίες διαχείρισης εγγράφων και δεδομένων, επιτρέποντας στους χρήστες να έχουν πρόσβαση σε σχετικά έγγραφα, φόρμες και δεδομένα που απαιτούνται για την ολοκλήρωση των εργασιών τους.
6. **Ειδοποιήσεις και υπενθυμίσεις:** Αυτοματοποιημένες ειδοποιήσεις ενημερώνουν τους χρήστες για εκκρεμείς εργασίες, προθεσμίες ή αλλαγές στις περιπτώσεις ροής εργασίας, διασφαλίζοντας ότι οι εργασίες ολοκληρώνονται εγκαίρως.
7. **Ιστορικό και καταγραφή:** Τήρηση λεπτομερούς ιστορικού για κάθε περίπτωση ροής εργασίας, συμπεριλαμβανομένων των ενεργειών που έγιναν, των χρηστών που τις εκτέλεσαν και του χρόνου ολοκλήρωσης. Παρέχεται επίσης και δυνατότητα ανάκαμψης σε προηγούμενες εκδόσεις.

Μερικά δημοφιλή παραδείγματα εμπορικών συστημάτων διαχείρισης ροών εργασίας που χρησιμοποιούνται στον επιχειρησιακό κόσμο είναι:

1. **Camunda -**Ένα ανοιχτού κώδικα σύστημα BPM που υποστηρίζει BPMN, CMMN και DMN. [8]
    - **Άδεια:** Apache License 2.0 (core)
    - **Χαρακτηριστικά:** Πλήρης υποστήριξη BPMN 2.0, CMMN, DMN, ενσωματωμένος modeler, REST API, αρχιτεκτονική οδηγούμενη από γεγονότα (event- driven).
    - **Γλώσσες/Πλατφόρμα:** Java, scripting (δηλ. υλοποίηση script tasks) σε Javascript ή Groovy
2. **BonitaSoft –** Μια πλατφόρμα ανάπτυξης επιχειρησιακών εφαρμογών με ενσωματωμένη διαχείριση ροών εργασίας. [9]
    - **Άδεια:** GPL v2 (Community Edition)
    - **Χαρακτηριστικά:** low-code development studio, drag & drop οπτική διεπαφή, ενσωματωμένες φόρμες (embedded forms)
    - **Γλώσσες/Πλατφόρμα:** Java, scripting σε Javascript
3. **ProcessMaker –** Μια πλατφόρμα διαχείρισης επιχειρησιακών διαδικασιών με χαμηλό κώδικα (low-code[5]). [10]
    - **Άδεια:** AGPL v3 (Community Edition)
    - **Χαρακτηριστικά:** low-code BPMN designer, ψηφιακές φόρμες (Dynaforms), REST API, analytics dashboard (διεπαφή ανάλυσης, multi-tenant (υποστήριξη πολλαπλών μισθωτών)
    - **Γλώσσες/Πλατφόρμα:** PHP, Javascript για client-side logic(λογική στην πλευρά του πελάτη)
4. **Flowable –** Ένα ευέλικτο σύστημα BPM που υποστηρίζει και Case Management. [11]
    - **Άδεια:** Apache License 2.0 (core)
    - **Χαρακτηριστικά:** Παρέχει υποστήριξη για BPMN, CMMN, DMN με java modules, ενσωματωμένο modeler, REST API, event registry (ευρετήριο γεγονότων), form engine (μηχανή φορμών), process instance migration (μετανάστευση περιπτώσεων διαδικασιών).
    - **Γλώσσες/Πλατφόρμα:** Java, scripting σε Javascript ή Groovy

Ο Πίνακας 1 προχωρά σε μια σύντομη συγκριτική παρουσίαση των παραπάνω προϊόντων με βάση συγκεκριμένα κριτήρια, όπως η άδεια χρήσης, τα υποστηριζόμενα πρότυπα (BPMN, CMMN, DMN), η ύπαρξη ενσωματωμένου εργαλείου μοντελοποίησης και διεπαφή REST API, οι δυνατότητες δημιουργίας φορμών, οι υποστηριζόμενες γλώσσες προγραμματισμού και συγγραφής σεναρίων (scripting), καθώς και χαρακτηριστικά, όπως η διεπαφή ανάλυσης δεδομένων (analytics dashboard), η υποστήριξη πολλαπλών οργανισμών (multi-tenancy) και η αρχιτεκτονική οδηγούμενη από γεγονότα (event driven).

Από την σύγκριση αυτή ξεχωρίζουν το Camunda και το Flowable, καθώς προσφέρουν την πληρέστερη κάλυψη προτύπων (BPMN, CMMN, DMN), ανοιχτή άδεια χρήσης (Apache License 2.0), ευρύ φάσμα γλωσσών συγγραφής σεναρίων, πλήρη υποστήριξη πολλαπλών οργανισμών και αρχιτεκτονική οδηγούμενη από γεγονότα, σε αντίθεση με το BonitaSoft και το ProcessMaker που περιορίζονται στο πρότυπο BPMN και παρέχουν μερικές μόνο από τις δυνατότητες.

[5] Χαμηλός κώδικας (low-code): Προσέγγιση ανάπτυξης λογισμικού που επιτρέπει τη δημιουργία εφαρμογών κυρίως μέσω γραφικών εργαλείων (drag-and-drop, οπτικά διαγράμματα) με ελάχιστη ή καθόλου χειρόγραφη γραφή κώδικα. Απευθύνεται τόσο σε προγραμματιστές όσο και σε επαγγελματίες χωρίς τεχνικό υπόβαθρο.

Στην δική μας περίπτωση επιλέχθηκε το Flowable ως βάση για τη serverless επέκταση, για τους λόγους που αναπτύσσονται αναλυτικά στην ενότητα 4.3.2.1.

ΠΙΝΑΚΑΣ 1 - ΣΥΓΚΡΙΤΙΚΗ ΠΑΡΟΥΣΙΑΣΗ ΔΗΜΟΦΙΛΩΝ ΣΥΣΤΗΜΑΤΩΝ ΔΙΑΧΕΙΡΙΣΗΣ ΡΟΩΝ ΕΡΓΑΣΙΑΣ

Χαρακτηριστικά	Camunda	BonitaSoft	ProcessMaker	Flowable
Άδεια	Apache License 2.0 (core)	GPL v2 (Community Edition)	AGPL v3 (Community Edition)	Apache License 2.0 (core)
Πρότυπα	BPMN, CMMN, DMN	BPMN	BPMN	BPMN, CMMN, DMN
Ενσωματωμένος modeler	Ναι	Ναι	Ναι	Ναι
REST API	Ναι	Ναι	Ναι	Ναι
Δημιουργία φορμών	Ναι	Ναι	Ναι	Ναι
Γλώσσες Προγραμματισμού	Java	Java, Groovy	PHP, Javascript	Java
Γλώσσες scripting	Javascript, Groovy, JUEL, Python (Jython), Ruby (JRuby)	Javascript, Groovy	PHP, Javascript	Javascript, Groovy, JUEL
Analytics dashboard	Ναι	Enterprise version	Ναι	Enterprise version
Multi-tenancy	Ναι	Όχι	Ναι	Ναι
Event-driven	Ναι	Μερικώς	Μερικώς	Ναι

2.3. Υπολογισμός Χωρίς Εξυπηρετητές

Το **cloud computing (υπολογιστικό νέφος ή νεφο-υπολογιστική)** είναι μια κεντρικοποιημένη υποδομή που επιτρέπει την διανομή υπολογιστικών πόρων (τόσο υλικού όσο και λογισμικού) σε πελάτες κατ' απαίτηση, μέσω του Διαδικτύου με τιμολόγηση με βάση τη χρήση. Περιλαμβάνει φυσικούς και εικονικούς διακομιστές, αποθηκευτικό χώρο και άλλους υπολογιστικούς πόρους που μπορούν να παρέχονται δυναμικά και να διαμορφώνονται ανάλογα με τις ανάγκες. Οι εφαρμογές νέφους (στο επίπεδο του λογισμικού) είναι επίσης προσβάσιμες μέσω του διαδικτύου χρησιμοποιώντας ένα τυπικό πρόγραμμα περιήγησης - συνεπώς παίρνουν τη μορφή μιας εφαρμογής ιστού (web application). Εκτός από πόρους υποδομής και λογισμικό, ένα υπολογιστικό νέφος μπορεί να προσφέρει και πλατφόρμες πάνω από τις οποίες αναπτύσσεται και εκτελείται λογισμικό, πλατφόρμες που φιλοξενούνται από υποκείμενους πόρους της υποδομής του νέφους. Οι υπηρεσίες υποδομής (Infrastucture-as-a-Service), πλατφόρμας (Platform-as-aservice) και λογισμικού (Software-as-a-Service) αποτελούν τα βασικά μοντέλα παράδοσης ή τύπους υπηρεσιών που προσφέρονται από το υπολογιστικό νέφος.

Ο **υπολογισμός χωρίς εξυπηρετητές** (**serverless computing)** είναι ένα σχετικά νέο μοντέλο παράδοσης του υπολογιστικού νέφους, στο οποίο ο πάροχος είναι υπεύθυνος για την διαχείριση των υποκείμενων διακομιστών και πλατφορμών εκτέλεσης ενώ ο χρήστης είναι υπεύθυνος μόνο για την ανάπτυξη του αντίστοιχου λογισμικού που τρέχει πάνω από αυτές.

Στο μοντέλο υπολογισμού χωρίς διακομιστή, ο χρήστης δεν χρειάζεται να παρέχει ή να διαχειρίζεται διακομιστές και πληρώνει μόνο για τον υπολογιστικό χρόνο που πραγματικά χρησιμοποιείται το λογισμικό του. Το λογισμικό αυτό εκτελείται συνήθως σε δοχεία χωρίς κατάσταση (stateless

containers), τα οποία δημιουργούνται και καταστρέφονται όπως απαιτείται από τον πάροχο νέφους. Συνεπώς, ο πάροχος του νέφους είναι υπεύθυνος για την τροφοδοσία του λογισμικού αυτού, την εκτέλεσή του κατ'απαίτηση αλλά και την κλιμάκωσή του εφόσον αυτό χρειαστεί. Το λογισμικό αυτό είναι συνήθως μικρό σε όγκο και παίρνει την μορφή μιας συνάρτησης χωρίς διακομιστή (serverless function).

Είναι σημαντικό να σημειωθεί σε αυτό το σημείο πως στο serverless computing δεν σημαίνει ότι δεν εμπλέκονται διακομιστές, αλλά ότι η διαχείριση και η κλιμάκωση (scaling) των διακομιστών γίνεται με αφαιρετικό τρόπο σε κάποιον πάροχο νέφους δίχως να χρειάζεται να επέμβει ο χρήστης.

Οι συναρτήσεις χωρίς εξυπηρετητές είναι μια συγκεκριμένη υλοποίηση λογισμικού στο υπολογιστικό νέφος που επιτρέπει στους προγραμματιστές να καταναλώνουν μικρές λειτουργίες ενός σκοπού. Αυτές οι λειτουργίες συνήθως ενεργοποιούνται από συγκεκριμένα συμβάντα, όπως αλλαγές σε δεδομένα σε μια βάση δεδομένων, μηνύματα που αποστέλλονται σε μια ουρά ή αιτήματα σε ένα συγκεκριμένο τελικό σημείο προγραμματιστικής διεπαφής (**API endpoint) (το οποίο περιτυλίγει την εκάστοτε λειτουργία)**.

Οι συναρτήσεις χωρίς εξυπηρετητές μπορούν να γραφτούν σε μια ποικιλία γλωσσών προγραμματισμού και συνήθως αναπτύσσονται και εκτελούνται σε stateless περιβάλλον. Αυτό σημαίνει ότι δεν διατηρούν πληροφορίες μεταξύ των εκτελέσεων και ότι οποιαδήποτε κατάσταση απαιτείται πρέπει να μεταβιβαστεί ως είσοδος (από τον χρήστη ή το λογισμικό/πράκτορα που τις εκτελεί).

Μερικά από τα πλεονεκτήματα του υπολογιστικού νέφους και κατ' επέκταση των συναρτήσεων χωρίς εξυπηρετητές (serverless function) είναι τα ακόλουθα:

1. **Κλιμάκωση (Scalability):** Το υπολογιστικό νέφος επιτρέπει την κλιμάκωση των πόρων ανάλογα με τις ανάγκες, είτε πρόκειται για τον χειρισμό μιας ξαφνικής αύξησης της κυκλοφορίας είτε για προσαρμογή σε περιόδους χαμηλής χρήσης. Αυτό μπορεί να βοηθήσει στη βελτίωση της συνολικής απόδοσης του συστήματος και στη μείωση του κόστους.
2. **Οικονομική Απόδοση (Cost-effectiveness):** Το υπολογιστικό νέφος μπορεί να είναι πιο αποδοτικό (ως προς το κόστος) από τα παραδοσιακά μοντέλα υπολογιστών εσωτερικής εγκατάστασης (ιδιωτικές ιδιόκτητες υποδομές), καθώς οι χρήστες πληρώνουν μόνο για τους πόρους που χρησιμοποιούν στην πραγματικότητα και αναθέτουν στον πάροχο την διάθεση και διαχείρισή τους. Επιπλέον, ο πάροχος νέφους είναι υπεύθυνος για τη συντήρηση και την αναβάθμιση της υποδομής, γεγονός που μπορεί να μειώσει περαιτέρω το κόστος διότι οι χρήστες δεν χρειάζεται να κάνουν κάποιο είδος συντήρησης/αναβάθμισης (υποδομής).
3. **Ευελιξία (Flexibility):** Το υπολογιστικό νέφος επιτρέπει στους χρήστες να έχουν πρόσβαση στους πόρους και τις εφαρμογές τους από οπουδήποτε και από οποιαδήποτε συσκευή, ανά πάσα στιγμή, αρκεί να έχουν σύνδεση στο διαδίκτυο. Αυτό μπορεί να είναι ιδιαίτερα χρήσιμο για ομάδες που συνεργάζονται με απομακρυσμένο τρόπο, π.χ. στην ανάπτυξη μιας εφαρμογής λογισμικού μέσω μιας παρεχόμενης πλατφόρμας νέφους.
4. **Ασφάλεια (Security):** Οι πάροχοι νέφους εφαρμόζουν συνήθως αυστηρά μέτρα και βέλτιστες πρακτικές ασφάλειας για την προστασία των δεδομένων και των άλλων πόρων των χρηστών που μεταφέρονται στην υποδομή τους.
5. **Υψηλή Διαθεσιμότητα (High-Avaiability):** Είναι μια σειρά από εφαρμοζόμενους μηχανισμούς και τεχνολογίες στο υπολογιστικό νέφος που διασφαλίζουν ότι οι εφαρμογές και οι υπηρεσίες παραμένουν λειτουργικές και προσβάσιμες με ελάχιστο χρόνο διακοπής ακόμη και όταν προκύψουν βλάβες ή προβλήματα. Σε αυτό το πλαίσιο, περιλαμβάνονται μηχανισμοί/τεχνολογίες, όπως εξισορρόπηση φορτίου (load balancing), αυτόματη

μετανάστευση πόρων (automatic resource migation), αυτόματη κλιμάκωση (auto-scaling), γεωγραφική κατανομή και αναπαραγωγή (geo distribution and replication), διαχείριση αποτυχιών (failover management), συνεχής παρακολούθηση και αυτόματη αποκατάσταση (continuous monitoring and self-healing). [12]

6. **Καινοτομία (Innovation):** Οι πάροχοι νέφους ενημερώνουν και βελτιώνουν συνεχώς τις υπηρεσίες τους, πράγμα το οποίο μπορεί να βοηθήσει τους χρήστες να επωφεληθούν από τις νέες τεχνολογίες και λειτουργίες πιο γρήγορα και εύκολα από ότι θα μπορούσαν απο μόνοι τους.
7. **Συνεργασία (Collaboration):** Το υπολογιστικό νέφος επιτρέπει την εύκολη συνεργασία μεταξύ των ομάδων, με την δυνατότητα κοινής χρήσης και πρόσβασης σε πόρους και εφαρμογές από οποιαδήποτε τοποθεσία.

Εκτός από τα προαναφερόμενα πλεονεκτήματα, υπάρχουν και εξειδικευμένα που αφορούν το serverless computing, τα οποία θα αναλυθούν στη συνέχεια. Οι συναρτήσεις χωρίς εξυπηρετητές εξαλείφουν την ανάγκη διαχείρισης υποδομών, καθώς ο πάροχος αναλαμβάνει εργασίες, όπως η τροφοδοσία (provisioning), ενημερώσεις ασφαλείας και αυτόματη κλιμάκωση μειώνοντας σημαντικά το λειτουργικό overhead. Κάθε serverless συνάρτηση δημιουργείται και καταστρέφεται δυναμικά κατά την εκτέλεσή της, εξασφαλίζοντας ενσωματωμένη αντοχή σε σφάλματα (fault tolerance) και υψηλή διαθεσιμότητα χωρίς χειροκίνητες ρυθμίσεις. Το μοντέλο χρέωσης ανά εκτέλεση επιτρέπει ακριβή υπολογισμό κόστους βάσει του πραγματικού χρόνου εκτέλεσης και των κλήσεων αποφεύγοντας τις χρεώσεις αδράνειας (που ισχύει στα server-based μοντέλα παράδοσης). Τέλος, το event-driven μοντέλο με υποστήριξη πολλαπλών triggers (π.χ. HTTP,queues, database events) επιταχύνει τον κύκλο ανάπτυξης και απλοποιεί την υλοποίηση μίκρο-υπηρεσιών (microservices), επιτρέποντας στους προγραμματιστές να εστιάσουν αποκλειστικά στην επιχειρησιακή λογική. [13] [14 [15]

Μερικές από τις περιπτώσεις χρήσης του serverless computing είναι οι εξής:

1. **Back-end για web και mobile εφαρμογές**
    - Φιλοξενία REST APIs και microservices χωρίς διαχείριση των servers.
    - Υλοποίηση authentication flows (π.χ. OAuth).
2. **Αποθήκευση δεδομένων & Backup (Big Data Processing)**
    - Αντικειμενοστραφείς υπηρεσίες (object storage) για την διαχείριση αρχείων και blobs.
    - Αυτόματα snapshots βάσεων δεδομένων (SQL/NoSQL).
3. **Επεξεργασία μεγάλων δεδομένων (Big Data Processing)**
    - Batch Jobs (π.χ. ETL pipelines).
    - Real-time streaming analytics (π.χ. Apache Kafka triggers).
4. **Event-driven workflows & task scheduling**
    - Cron jobs, message queues, database triggers.
    - Ορισμός απλών ροών εργασίας που εκκινούνται από γεγονότα.
5. **Multimedia processing**
    - Αυτόματη κωδικοποίηση εικόνων/βίντεο κατά το upload.
    - Δημιουργία thumbnails ή μετατροπή μορφότυπου αρχείων (format conversion).
6. **Internet of Things (IOT) integrations**
    - Επεξεργασία δεδομένων αισθητήρων σε πραγματικό χρόνο.
    - Αυτοματισμοί βάσει συμβάντων από έξυπνες συσκευές.
7. **AI/ML**
    - Φιλοξενία μοντέλων μηχανικής μάθησης για online προβλέψεις (inference).

- Συνδυασμός με GPU-powered instances όπου απαιτείται. [16]

Μερικές από τις πιο δημοφιλείς πλατφόρμες του serverless computing είναι τα εξής:

1. **AWS Lambda**[6]
2. **Azure Functions**[7]
3. **Google Cloud Functions**[8]
4. **OpenWhisk**[9]

Παρακάτω παρέχεται ένας πίνακας σύγκρισης των βασικών χαρακτηριστικών για τις τέσσερις προαναφερόμενες serverless πλατφόρμες:

ΠΙΝΑΚΑΣ 2 - ΣΥΓΚΡΙΣΗ ΒΑΣΙΚΩΝ ΧΑΡΑΚΤΗΡΙΣΤΙΚΩΝ ΤΩΝ ΤΕΣΣΑΡΩΝ SERVERLESS ΠΛΑΤΦΟΡΜΩΝ

Χαρακτηριστικό	AWS Lambda	Azure Functions	Google Cloud Functions	Apache OpenWhisk
Πάροχος πλατφόρμας	AWS	Microsoft Azure	Google Cloud Platform	Open-source (αναπτύσσεται από Apache)
Ενεργοποιητές συμβάντων	HTTP (API Gateway), S3, DynamoDB, SQS, SNS, EventBridge – υποστηρίζονται επίσης ενεργοποιητές για υπηρεσίες streaming, IoT και παρακολούθησης εντός του AWS οικοσυστήματος	HTTP, Blob Storage, Service Bus, Event Grid, Queue Storage, Cosmos DB, Timer	HTTP, Pub/Sub, Cloud Storage, Firestore, Cloud Scheduler	HTTP, Cloudant, Kafka, CVS, κ.ά.
Γλώσσες προγραμματισμού	Node.js, Python, Java, Go, Ruby, .NET	Node.js, Python, Java, Go, Ruby, .NET	Node.js, Python, Java, Go, Ruby, .NET,	Node.js, Python, Java, Go, Ruby, .NET
Κλιμάκωση (Scaling)	Αυτόματη, concurrency limits[10]	Αυτόματη, plan-based (Consumption/App Service)	Αυτόματη, concurrency limits	Αυτόματη, configurable invoker pools[11]

[6] **AWS Lambda**: https://aws.amazon.com/lambda/
[7] **Azure Functions:** https://azure.microsoft.com/en-us/products/functions
[8] **Google Cloud Functions:** https://cloud.google.com/functions
[9] **OpenWhisk:** https://openwhisk.apache.org
[10] *Concurrency limits*: Το ανώτατο πλήθος στιγμιότυπων μιας συνάρτησης που εκτελούνται ταυτόχρονα. Όταν οι παράλληλες κλήσεις ξεπεράσουν το όριο, οι επιπλέον αιτήσεις τίθενται σε αναμονή ή απορρίπτονται (throttling).
[11] *Configurable invoker pools*: Στο Apache OpenWhisk οι invokers είναι τα συστατικά που εκτελούν τις συναρτήσεις μέσα σε containers. Επειδή το OpenWhisk είναι αυτό-φιλοξενούμενο, ο διαχειριστής μπορεί να ρυθμίσει το πλήθος και τους πόρους των invokers, ελέγχοντας τη χωρητικότητα εκτέλεσης και τον βαθμό παραλληλισμού.

Παρακολούθηση & Εντοπισμός σφαλμάτων	CloudWatch, X-Ray	Application Insights, Monitor	Stackdriver Logging & Trace	Προαιρετικά εργαλεία (Prometheus,..)
Μοντέλο Χρέωσης (Pricing)	Pay-per-execution + κατανάλωση πόρων (χρόνος εκτέλεσης * μνήμη/CPU)	Pay-per-execution + κατανάλωση πόρων (χρόνος εκτέλεσης * μνήμη/CPU)	Pay-per-execution + κατανάλωση πόρων (χρόνος εκτέλεσης * μνήμη/CPU)	Δωρεάν & on-premise (χωρίς πάγια κόστη)
Ενσωματώσεις (Integrations)	Σχεδόν σε όλο το AWS ecosystem	Στενή με Azure services	Στενή με GCP services	Plugins & OpenWhisk Catalog
Άδεια & Ανοικτός Κώδικας	Ιδιόκτητο core, CLI open-source	Ιδιόκτητο	Ιδιόκτητο	Apache License 2.0

Από τη σύγκριση προκύπτει ότι οι τρεις εμπορικές πλατφόρμες (AWS Lambda, Azure Function, Google Cloud Functions) παρουσιάζουν αρκετές ομοιότητες ως προς το μοντέλο χρέωσης, τις υποστηριζόμενες γλώσσες και την αυτόματη κλιμάκωση. Κύρια διαφοροποίηση τους αποτελεί ο βαθμός ενσωμάτωσης με το οικοσύστημα τους (δηλ. του εκάστοτε παρόχου). Αντίθετα, η Apache OpenWhisk ξεχωρίζει ως η μόνη ανοικτού κώδικα επιλογή, επιτρέποντας on-premise ανάπτυξη χωρίς πάγια κόστη, γεγονός που τη καθιστά ιδιαίτερα ελκυστική για οργανισμούς που επιθυμούν πλήρη έλεγχο της υποδομής τους.

Όλες οι serverless πλατφόρμες εφαρμόζουν μια κοινή αρχιτεκτονική 4 στρωμάτων αλλά διαφοροποιούνται σε λεπτομέρειες. Τα 4 αυτά στρώματα είναι τα εξής:

- Στο 1ο στρώμα (Εξωτερικές Υπηρεσίες) βρίσκονται οι “πηγές” των γεγονότων που ενεργοποιούν τις συναρτήσεις (Http αιτήματα, μηνύματα σε ουρές κ.α.)
- Στο 2ο στρώμα (Control Plane/Orchestrator) η πλατφόρμα παρακολουθεί τα εισερχόμενα events και διαχειρίζεται τον κύκλο ζωής των execution environments (περιβάλλοντα όπου εκτελούνται οι συναρτήσεις).
- Στο 3ο στρώμα (Runtime Environment) ο κώδικας της συνάρτησης φορτώνεται μαζί με τις εξαρτήσεις του σε ένα ελαφρύ απομονωμένο περιβάλλον. Εκεί εκτελείται ο handler της συνάρτησης, ελέγχεται ο χρόνος εκτέλεσης (timeout) και τα όρια μνήμης.
- Στο 4ο στρώμα (Monitoring & Logging) η πλατφόρμα εστιάζει στην ορατότητα και διάγνωση των functions, συλλέγοντας μετρικές, όπως αιτήσεις ανά δευτερόλεπτο, τον χρόνο εκτέλεσης, το ποσοστό σφαλμάτων, και την χρήση μνήμης /CPU, και καταγράφοντας δομημένες καταχωρήσεις (logs).

Η Εικόνα 2 απεικονίζει την συγκεκριμένη υλοποίηση της παραπάνω γενικής αρχιτεκτονικής στην πλατφόρμα Apache OpenWhisk. Παρότι τα τέσσερα λογικά στρώματα δεν εμφανίζονται με την ίδια σειρά ούτε ως διακριτές ζώνες, αντιστοιχούν άμεσα στα δομικά μέρη του διαγράμματος.

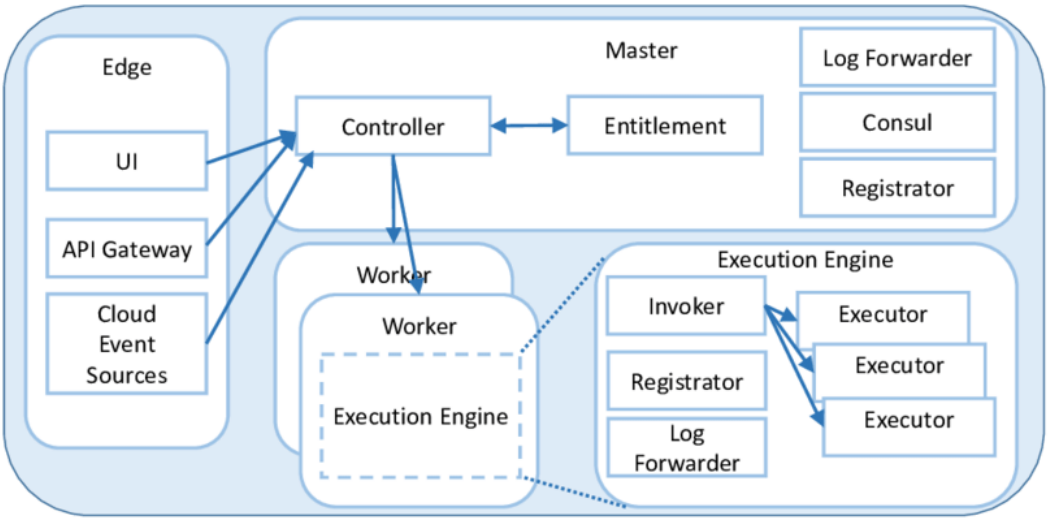
- Το 1ο στρώμα (Εξωτερικές Υπηρεσίες) αντιστοιχεί στη ζώνη **Edge**, όπου βρίσκονται οι πηγές των εισερχόμενων γεγονότων: το περιβάλλον χρήστη (UI), η πύλη API (API Gateway) και οι εξωτερικές πηγές γεγονότων cloud (Cloud Event Sources).
- Το 2ο στρώμα (Control Plane/Orchestrator) αντιστοιχεί στη ζώνη **Master**, με κεντρικό στοιχείο τον Controller, ο οποίος δέχεται τα αιτήματα, ελέγχει τα δικαιώματα πρόσβασης μέσω του

Entitlement και δρομολογεί την εκτέλεση προς τους Workers, συμπληρωματικά τα Consul και Registrator αναλαμβάνουν την ανακάλυψη και καταχώριση των υπηρεσιών.

- Το 3ο στρώμα (Runtime Environment) αντιστοιχεί στη ζώνη Worker και ειδικότερα στη μηχανή εκτέλεσης (Execution Engine): ο Invoker εκκινεί έναν ή περισσότερους Executors (απομονωμένα περιβάλλοντα σε containers) μέσα στους οποίους φορτώνεται και εκτελείται ο κώδικας της συνάρτησης.
- Το 4ο στρώμα (Monitoring & Logging) δεν αποτελεί εδώ ξεχωριστή ζώνη αλλά είναι ενσωματωμένο μέσω των στοιχείων Log Forwader που εμφανίζονται τόσο στη ζώνη Master όσο και εντός της μηχανής εκτέλεσης, συλλέγοντας και προωθώντας τις καταγραφές (logs) της εκτέλεσης.

Με τον τρόπο αυτό η αρχιτεκτονική της OpenWhisk υλοποιεί και τα τέσσερα λογικά στρώματα, με τα στρώματα εκτέλεσης (3ο) και καταγραφής (4ο) να συνυπάρχουν στην ίδια φυσική ζώνη (Worker).

Συνολικά, οι serverless πλατφόρμες παρέχουν το FaaS ως πλήρως διαχειριζόμενη υπηρεσία, επιτρέποντας στον χρήστη να εστιάσει μόνο στον κώδικα και στους κανόνες ενεργοποίησης, χωρίς να ανησυχεί για την (υποκείμενη) υποδομή. [17]



Εικονα 2 - Αρχιτεκτονικη Serverless Πλατφορμας Apache OpenWhisk

3. Συγκριτική Ανάλυση Συναφών Εργασιών

Η τεχνολογία του serverless computing έχει απασχολήσει έντονα την ερευνητική κοινότητα και τη βιομηχανία την τελευταία δεκαετία. Πολλές σχετικές εργασίες και πλατφόρμες έχουν μελετήσει το πως μπορούν να αναπτυχθούν και να διατεθούν συναρτήσεις χωρίς εξυπηρετητές, ωστόσο σε πολλές περιπτώσεις παρουσιάζονται σημαντικοί περιορισμοί, είτε στην πρακτική εφαρμογή είτε στη διασύνδεση με πραγματικά συστήματα.

Η εργασία των **Baldini et al. (2017)** [19] αναλύει τις βασικές αρχές και τις τεχνικές προκλήσεις του serverless μοντέλου, όπως η δυσκολία σύνθεσης λειτουργιών και η διαχείριση κατάστασης. **Οι Jonas et al. (2019)** [20] ορίζουν το υπολογιστικό αυτό μοντέλο από την οπτική του Berkley, συζητούν την εξέλιξη από παραδοσιακά μοντέλα του νέφους σε αυτό το μοντέλο, προσδιορίζουν κύριους περιορισμούς του μοντέλου και αναδεικνύουν ερευνητικές προκλήσεις και σχετικές κατευθύνσεις. Οι **Fox et al. (2017)** [21] εστιάζουν στην καταγραφή της πορείας του Function-as-a-Service (FaaS) τόσο στην έρευνα όσο και στη βιομηχανία, αναδεικνύοντας την ανάγκη για πιο ώριμα εργαλεία και καλύτερη ενσωμάτωση στις διαδικασίες ανάπτυξης. Οι **Kritikos & Skrzypek (2020)** [22] παρουσιάζουν συγκριτικά τα πιο γνωστά serverless πλαίσια, με έμφαση στο Serverless Framework, και ιδιαίτερα στη δυνατότητα υποστήριξης πολλών παρόχων και στις τεχνικές τους δυνατότητες.

Σχετική με το αντικείμενο της παρούσας εργασίας είναι και η μελέτη των **Yussupov et al. (2022**) [23], οι οποίοι προτείνουν μια μέθοδο μοντελοποίησης και ανάπτυξης ενορχηστρώσεων serverless συναρτήσεων βασισμένη σε καθιερωμένα πρότυπα. Η BPMN χρησιμοποιείται για τη μοντελοποίηση της λογικής ορχηστρωσης, ενώ η TOSCA[12] για τη μοντελοποίηση της διάταξής της, με στόχο την ανεξαρτησία από συγκεκριμένο πάροχο ή τεχνολογία τόσο σε επίπεδο μηχανισμών ενορχήστρωσης συναρτήσεων (π.χ. AWS Step Functions[13], Azure Durable Functions[14], IBM Composer[15]) όσο και σε επίπεδο εργαλείων ανάπτυξης. Στην προσέγγιση τους ωστόσο τα μοντέλα BPMN δεν εκτελούνται απευθείας αλλά μετασχηματίζονται αυτόματα σε ιδιόκτητες μορφές περιγραφής (π.χ. Amazon States Language) μέσω της εργαλειοθήκης που ανέπτυξαν, ενώ η τελική διάταξη πραγματοποιείται με ξεχωριστό εργαλείο αυτοματοποίησης διάταξης.

Οι περισσότερες από τις παραπάνω μελέτες προσφέρουν πολύτιμη θεωρητική βάση και συμβάλλουν στην κατανόηση των θεμελιωδών θεμάτων γύρω από το serverless μοντέλο. Η παρούσα εργασία έρχεται να συμπληρώσει αυτή την εικόνα, εστιάζοντας στην πρακτική πλευρά της ενσωμάτωσης serverless συναρτήσεων σε συστήματα διαχείρισης ροών εργασίας. Εκεί όπου οι σχετικές εργασίες εστιάζουν κυρίως στην αρχιτεκτονική και τη θεωρητική ανάλυση, εδώ προτείνεται ένας τρόπος αξιοποίησης των υπαρχόντων εργαλείων σε πραγματικά σενάρια χρήσης, δίνοντας έμφαση στην ευκολία υλοποίησης, την ανεξαρτησία από συγκεκριμένους παρόχους και την επαναχρησιμοποίηση των συναρτήσεων.

Η παρούσα εργασία διαφοροποιείται από τη μελέτη των Yussupov et al. (2022). Σε δύο σημεία. Πρώτον, η BPMN δεν αποτελεί ενδιάμεση αναπαράσταση που μετασχηματίζεται και εγκαταλείπεται, αλλά παραμένει η ίδια η γλώσσα εκτέλεσης μέσα σε ένα WFMS. Τα script tasks καλούν απευθείας τις serverless συναρτήσεις κατά την εκτέλεση χωρίς ενδιάμεσο στάδιο μετασχηματισμού. Δεύτερον, ως προς το ζήτημα της ετερογένειας των τεχνολογιών ανάπτυξης που επισημαίνουν και οι ίδιοι ως

[12] *Topology and Orchestration Specification for Cloud Applications (TOSCA)*: https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=tosca

[13] *AWS Step Functions*: https://docs.aws.amazon.com/step-functions/latest/dg/welcome.html

[14] *Azure Durable Functions*: https://learn.microsoft.com/el-gr/azure/durable-task/durable-functions/durable-functions-overview

[15] IBM Composer : https://github.com/ibm-functions/composer

εμπόδιο στην ανεξαρτησία παρόχου, η παρούσα υλοποίηση το αντιμετωπίζει αποκρύπτοντας τις διαφορές μεταξύ AWS, Google Cloud και OpenWhisk, ενώ το κατάλληλο αρχείο διαμόρφωσης (serverless.yml) παράγεται αυτόματα από το σύστημα με βάση κοινά εσωτερικά δεδομένα, χωρίς ο προγραμματιστής να χρειάζεται γνώση της εκάστοτε σύνταξης. Πρόκειται για διαφορετική προσέγγιση από τη χρήση επίσημου προτύπου όπως το TOSCA, βασισμένη σε αυτοματοποιημένη παραγωγή ρυθμίσεων αντί ανοιχτού προτύπου που ωστόσο επιτυγχάνει στην πράξη ανεξαρτησία παρόχου από την οπτική του προγραμματιστή.

Συνολικά η εργασία των Yussupov et al. (2022) προσφέρει μια πιο τυπική και θεωρητικά πληρέστερη λύση μέσω καθιερωμένων προτύπων (BPMN, TOSCA) και αντίστοιχης εργαλειοθήκης μετασχηματισμού, με κόστος όμως την εκμάθηση επιπλέον προτύπων και την εγκατάλειψη του αρχικού μοντέλου κατά την εκτέλεση. Η παρούσα εργασία επιτυγχάνει αντίστοιχη ανεξαρτησία παρόχου με σημαντικά μικρότερο κόστος πολυπλοκότητας, καθώς η BPMN παραμένει ενιαία γλώσσα μοντελοποίησης και εκτέλεσης σε όλο τον κύκλο ζωής της ροής εργασίας χωρίς ενδιάμεσα μοντέλα ή μετασχηματισμούς.

4. Ανάπτυξη Λογισμικού

Σε αυτό το κεφάλαιο ακολουθείται δομικά η βασική διαδικασία ανάπτυξης λογισμικού με βάση το μοντέλο του καταρράκτη που περιλαμβάνει τέσσερα διαδοχικά στάδια: πρώτα την ανάλυση απαιτήσεων, στη συνέχεια την σχεδίαση και έπειτα την υλοποίηση (με υποενότητες για την αρχιτεκτονική υλοποίησης, την περιγραφή των επαναχρησιμοποιούμενων εργαλείων και τον τρόπο διαμόρφωσης ενσωμάτωσης) και τέλος την επικύρωση του λογισμικού μέσω δοκιμών διαφορετικού είδους. Κάθε φάση εκτελείται σειριακά, διασφαλίζοντας μια ολοκληρωμένη και δομημένη προσέγγιση ανάπτυξης λογισμικού. Τονίζεται πως η τελευταία ενότητα του κεφαλαίου αποτελεί και αυτή ένα είδος (ποιοτικής) επικύρωσης (εκτός από την προτελευταία που καλύπτει τις δοκιμές), αξιολογώντας τον βαθμό ικανοποίησης των απαιτήσεων που έχουν τεθεί.

4.1. Ανάλυση Απαιτήσεων

Οι απαιτήσεις εντοπίστηκαν κυρίως με βάση το ίδιο το αντικείμενο της διπλωματικής (με λογικό τρόπο) και τη βιβλιογραφική ανασκόπηση σε έρευνες και εργασίες σχετικά με serverless frameworks, όπως οι [19] [20] [21].

1. **Λειτουργικές Απαιτήσεις**:
    - Α. Δυνατότητα συμπερίληψης του κώδικα συναρτήσεων χωρίς εξυπηρετητές (serverless functions) σε λειτουργίες αυτοματοποίησης (tasks) σε ροές εργασίας (workflows) BPMN, ειδικότερα σε **script tasks.** Σε αυτή την περίπτωση, ο κώδικας αποθηκεύεται για μετέπειτα χρήση στην βάση δεδομένων σε μια στήλη κειμένου ενός table αλλά και σε αρχείο το οποίο φιλοξενεί τον κώδικα και βρίσκετε σε ένα ορισμένο φάκελο της εφαρμογής για χρήση απο το Serverless Framework. Παρόλο που η επέκταση έγινε σε script task, η επέκταση και σε service task είναι παρόμοια οπότε σημειώνεται για μελλοντική ανάπτυξη/επέκταση της λύσης.
    - Β. Δυνατότητα αντιστοίχισης κάποιας ήδη υπάρχουσας **serverless συνάρτησης** σε μια αυτοματοποιημένη λειτουργία (script task) μέσα στην ροή εργασίας (workflow). Πιο συγκεκριμένα, αυτή η αντιστοίχιση ορίζεται κατά την διάρκεια της δημιουργίας ενός script task μέσα στην ροή εργασίας.
    - Γ. Δυνατότητα υποστήριξης πολλαπλών γλωσσών προγραμματισμού για την επικύρωσης του κώδικα της κάθε **s**erverless συνάρτησης με την χρήση του εργαλείου **ANtlr4**.
    - Δ. Δυνατότητα ρύθμισης για τοπική ή απομακρυσμένη εκτέλεση μιας **serverless συνάρτησης.** Η συγκεκριμένη απαίτηση θα μπορούσε να επιτευχθεί στο επίπεδο της αυτοματοποιημένης λειτουργίας (script task) με την χρήση μεταβλητών που ορίζονται στον **Execution Listener**. Κατά τον χρόνο εκτέλεσης (runtime), το σύστημα WFMS μπορεί να διαβάζει αυτήν την μεταβλητή από τον listener και με βάση την κατάσταση της μεταβλητής (αντιμετωπίζεται σαν true/false) να εκτελείται και διαφορετική λογική από το σύστημα.
    - Ε. Δυνατότητα πακεταρίσματος και διάταξης όλων των **serverless functions** που έχουν οριστεί μέσα σε μια ροή εργασίας. Οι serverless platform πάροχοι που θα πρέπει να υποστηρίζονται είναι οι AWS Lambda, Google Functions & IBM OpenWhisk**.**
    - ΣΤ. Το σύστημα πρέπει να εντοπίζει πότε ο κώδικας μιας serverless συνάρτησης έχει τροποποιηθεί και να αναδιατάσσει μόνο αυτή τη function. Συγκεκριμένα:
        - i. Σε κάθε script task θα πρέπει να αποθηκεύεται το SHA256 hash του κώδικα στη βάση δεδομένων.

ii. Κατά το πακετάρισμα θα πρέπει να υπολογίζεται εκ νέου το sha256 του τρέχοντος κώδικα από το script task και συγκρίνεται με το hash που υπάρχει στη βάση.
iii. Εάν τα δύο hashes διαφέρουν, θεωρείται ότι ο κώδικας έχει αλλάξει και θα πρέπει να γίνει αυτόματα η αναδιάταξη μόνο αυτής της function. Αν τα hashes ταυτίζονται, θα πρέπει να παραλείπεται το στάδιο της αναδιάταξης γιατί θεωρείται πως υπάρχει ήδη αντίστοιχη συνάρτηση διαθέσιμη στο σύστημα.

Ζ. Το σύστημα θα πρέπει να χειρίζεται πολλαπλές **serverless συναρτήσεις** βάσει διαφορετικών αρχείων διαμόρφωσης του **Serverless Framework** και να τις διατάσσει χωρίς χειροκίνητη παρέμβαση.

Η. Το σύστημα θα πρέπει να αποθηκεύει την πληροφορία της **serverless συνάρτησης** μαζί με τα απαραίτητα δεδομένα από τη ροή εργασίας (όπως το αναγνωριστικό του script task, το αναγνωριστικό ορισμού της εφαρμογής που φιλοξενεί ροής εργασίας, το όνομα της συνάρτησης, η γλώσσα προγραμματισμού, το SHA-256 hash του κώδικα και η ρύθμιση τοπικής ή απομακρυσμένης εκτέλεσης) για την αυτοματοποιημένη υποστήριξη των προαναφερόμενων λειτουργιών.

Θ. Το σύστημα θα πρέπει να είναι διαμορφωμένο έτσι ώστε να επιτρέπει την αλληλεπίδραση με το **Serverless Framework.** Αυτό θα πρέπει να επιτυγχάνεται με την δημιουργία μεταβλητών περιβάλλοντος συστήματος (system environment variables).

2. **Μη Λειτουργικές Απαιτήσεις**:

Α. Το σύστημα θα πρέπει να εκτελεί τις **serverless συναρτήσεις** μέσα σε ένα εύλογο χρονικό διάστημα.
i. Το 95% των ενάρξεων εκτέλεσης μιας function (warm start) πρέπει να ολοκληρώνονται σε ≤ 800 ms από τη στιγμή που ενεργοποιείται το script task μέχρι να επιστραφεί το αποτέλεσμα. Ουσιαστικά, μιλάμε για τον χρόνο που απαιτείται για να γίνει διαθέσιμη η συνάρτηση ώστε έπειτα να μπορεί να εκτελεστεί.
ii. Σε περίπτωση cold start, ο χρόνος έναρξης της εκτέλεσης πρέπει να είναι ≤ 2 δευτερόλεπτα.

Β. Το σύστημα θα πρέπει να πακετάρει και να διατάσσει τις **serverless συναρτήσεις** σε εύλογο χρονικό διάστημα
i. Το διάστημα αυτό εξαρτάται από το μέγεθος του κώδικα και δεν πρέπει να υπερβαίνει τα 10 λεπτά.

Γ. Το σύστημα θα πρέπει να παρέχει επαρκής πληροφόρηση για τυχόν σφάλματα κατά την διαδικασία υποβολής της ροής εργασίας αλλά και κατά την διαδικασία της εκτέλεσης όσον αφορά το κομμάτι των **serverless συναρτήσεων**.

Δ. Το σύστημα θα πρέπει να παρέχει κατάλληλο τρόπο για εύκολη εισαγωγή και τροποποίηση της πληροφορίας κάποιας serverless συνάρτησης σε μια ροή εργασίας, εύκολη τροποποίηση ρυθμίσεων εκτέλεσης και ασφαλή ρύθμιση/προσδιορισμό στοιχείων λογαριασμού για serverless πλατφόρμες. Αυτό θα πρέπει να επιτυγχάνεται με την βοήθεια του συστήματος διαχείρισης ροών προς επιλογή σε συνδυασμό με προσαρμοσμένη λογική με την μορφή κώδικα.

4.2. Σχεδίαση

Σε αυτήν την ενότητα παρουσιάζονται και αναλύονται διαγράμματα σχετικά με την αρχιτεκτονική του συστήματος, την ροή των δεδομένων μέσα σε αυτό, τις περιπτώσεις χρήσης του και τις οντότητες

δεδομένων που διαχειρίζεται. Τα διαγράμματα αυτά προέκυψαν κατά την φάση της σχεδίασης του συστήματος.

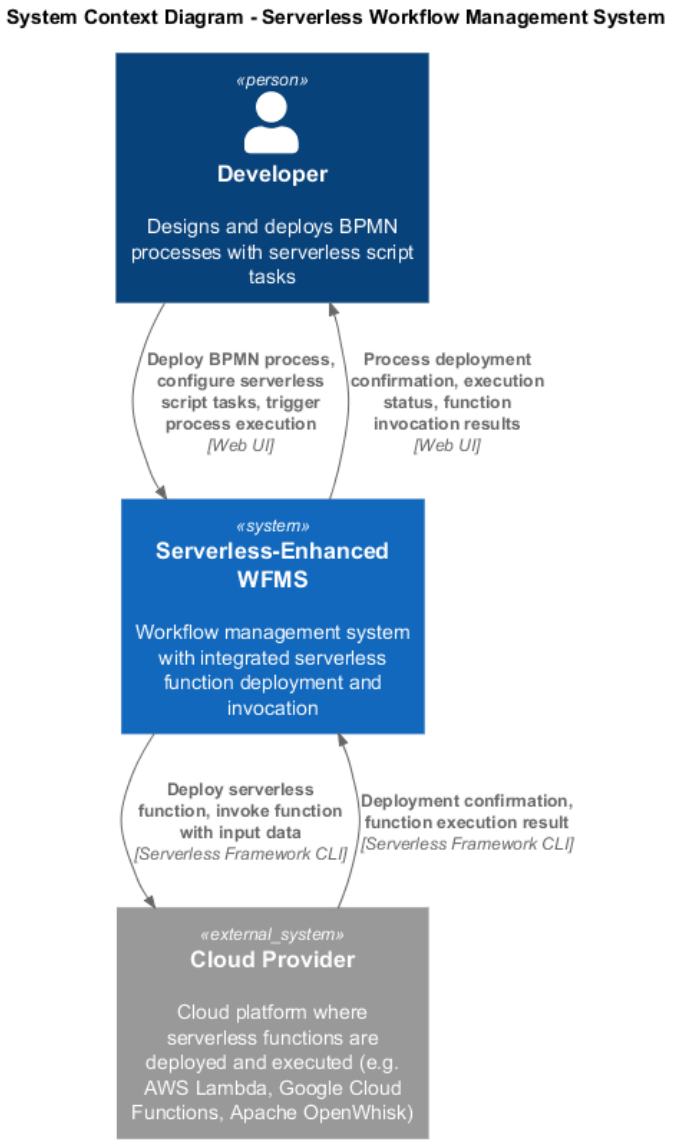
4.2.1. Διάγραμμα Περιβάλλοντος

Στην Εικόνα 3 απεικονίζεται το διάγραμμα περιβάλλοντος (**system context diagram**) του συστήματος.

Το κεντρικό σύστημα είναι το **Serverless-Enhanced WFMS,** το οποίο αποτελεί επέκταση του συστήματος διαχείρισης ροών εργασίας **Flowable** με ενσωματωμένη υποστήριξη για διάταξη και εκτέλεση serverless συναρτήσεων.

Η κύρια εξωτερική οντότητα είναι ο **Προγραμματιστής** (Developer), ο οποίος σχεδιάζει και αναπτύσσει BPMN διαδικασίες που περιλαμβάνουν serverless script tasks.

Το μοναδικό εξωτερικό σύστημα με το οποίο αλληλεπιδρά άμεσα η επέκταση είναι η serverless πλατφόρμα του εκάστοτε cloud παρόχου (AWS Lambda, Google Cloud Functions, Apache OpenWhisk). Για τη διάταξη (deployment) και την εκτέλεση (invocation) των serverless συναρτήσεων το σύστημα χρησιμοποιεί εσωτερικά το Serverless Framework, ένα εργαλείο γραμμής εντολών (CLI) βασισμένο σε Node.js, το οποίο είναι ενσωματωμένο στο στρώμα υποδομής της επέκτασης και το οποίο επικοινωνεί με το API/SDK της εκάστοτε serverless πλατφόρμας.



ΕΙΚΟΝΑ 3 - ΔΙΑΓΡΑΜΜΑ ΠΕΡΙΒΑΛΛΟΝΤΟΣ

4.2.2. Διάγραμμα Περιπτώσεων Χρήσης

Στην Εικόνα 4 απεικονίζεται το διάγραμμα περιπτώσεων χρήσης του συστήματός μας. Στο διάγραμμα αυτό βλέπουμε το σύστημα και την αλληλεπίδραση των εξωτερικών οντοτήτων με αυτό στο πλαίσιο των βασικών περιπτώσεων χρήσης, δηλαδή των βασικών του λειτουργιών. Κύρια εξωτερική οντότητα είναι ο προγραμματιστής (Developer) που δημιουργεί, τροποποιεί και εκτελεί τις ροές εργασίας μέσω του WFMS. Επιπλέον, η Serverless Platform (π.χ AWS Lambda, Google Cloud Functions, Apache OpenWhisk) αποτελεί την εξωτερική οντότητα με την οποία το σύστημα αλληλεπιδρά για την δημιουργία, τροποποίηση και εκτέλεση των serverless συναρτήσεων.

Περιγράφουμε παρακάτω συνοπτικά τις περιπτώσεις χρήσης και την αλληλεπίδραση που λαμβάνει χώρα μεταξύ του συστήματος και των οντοτήτων αυτών στο πλαίσιο των περιπτώσεων αυτών.

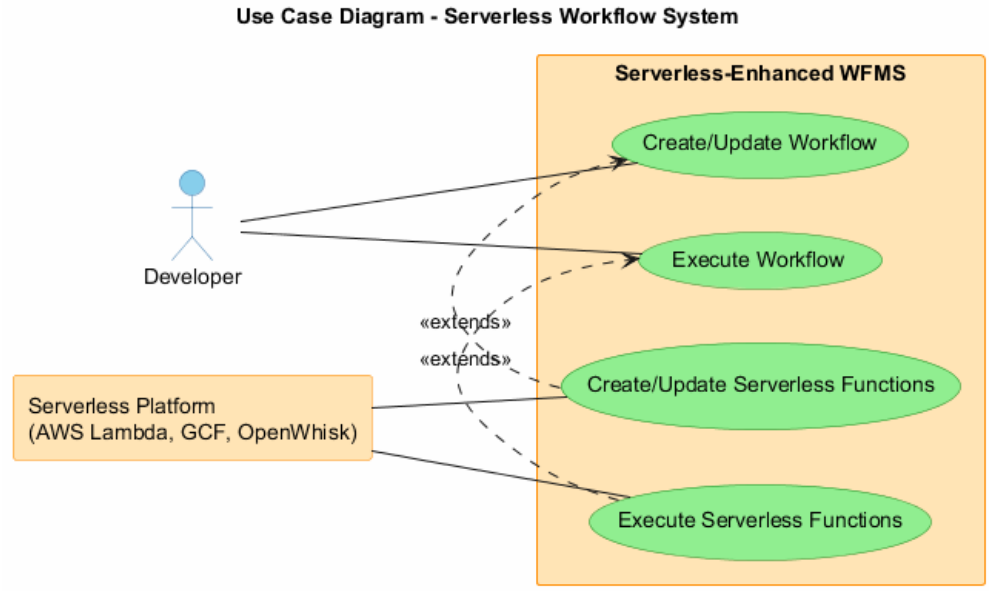
1. **Δρώντες (Actors):**
    a. Η οντότητα του προγραμματιστή (Developer) αλληλεπιδρά με το σύστημα σε όλες τις περιπτώσεις χρήσης που περιγράφονται παρακάτω.

b. Η εξωτερική οντότητα του Serverless Platform αλληλεπιδρά με το σύστημα κατά την δημιουργία, ενημέρωση και εκτέλεση των serverless συναρτήσεων, αναλαμβάνοντας το πακετάρισμα, τη διάταξη και την εκτέλεση είτε τοπικά είτε στο cloud.

2. **Περιπτώσεις Χρήσης (Use Cases):**
   a. **Create/Update Workflow:** Ο προγραμματιστής δημιουργεί ή τροποποιεί ροές εργασίας μέσα στο WFMS σύστημα.
      - Η περίπτωση αυτή επεκτείνεται από την περίπτωση χρήσης Create/Update Serverless Functions, η οποία αφορά τη δημιουργία ή τροποποίηση serverless συναρτήσεων που ενσωματώνονται στις ροές εργασίας. Η λειτουργία αυτή οδηγεί εν τέλει σε αλληλεπίδραση με την εξωτερική οντότητα Serverless Platform.

   b. Execute Workflow: Ο προγραμματιστής εκτελεί τις ροές εργασίας που έχουν οριστεί στο σύστημα.
      - Η περίπτωση αυτή επεκτείνεται από την περίπτωση χρήσης Execute Serverless Functions, η οποία αφορά την εκτέλεση των serverless συναρτήσεων που περιέχονται στις ροές εργασίας. Η εκτέλεση αυτή πραγματοποιείται μέσω της εξωτερικής οντότητας Serverless Platform.

Να σημειωθεί πως οι περιπτώσεις χρήσης που αφορούν τις serverless συναρτήσεις (Create/Update Serverless Functions και Execute Serverless Functions) αποτελούν επεκτάσεις βασικών περιπτώσεων χρήσης (διαχείρισης ροών εργασίας), πράγμα που σημαίνει ότι η κανονική λειτουργία του συστήματος δεν επηρεάζεται αν δεν χρησιμοποιηθεί η επέκταση serverless.



ΕΙΚΟΝΑ 4 - ΔΙΑΓΡΑΜΜΑ ΠΕΡΙΠΤΩΣΕΩΝ ΧΡΗΣΗΣ

4.2.3. Διάγραμμα Συστατικών

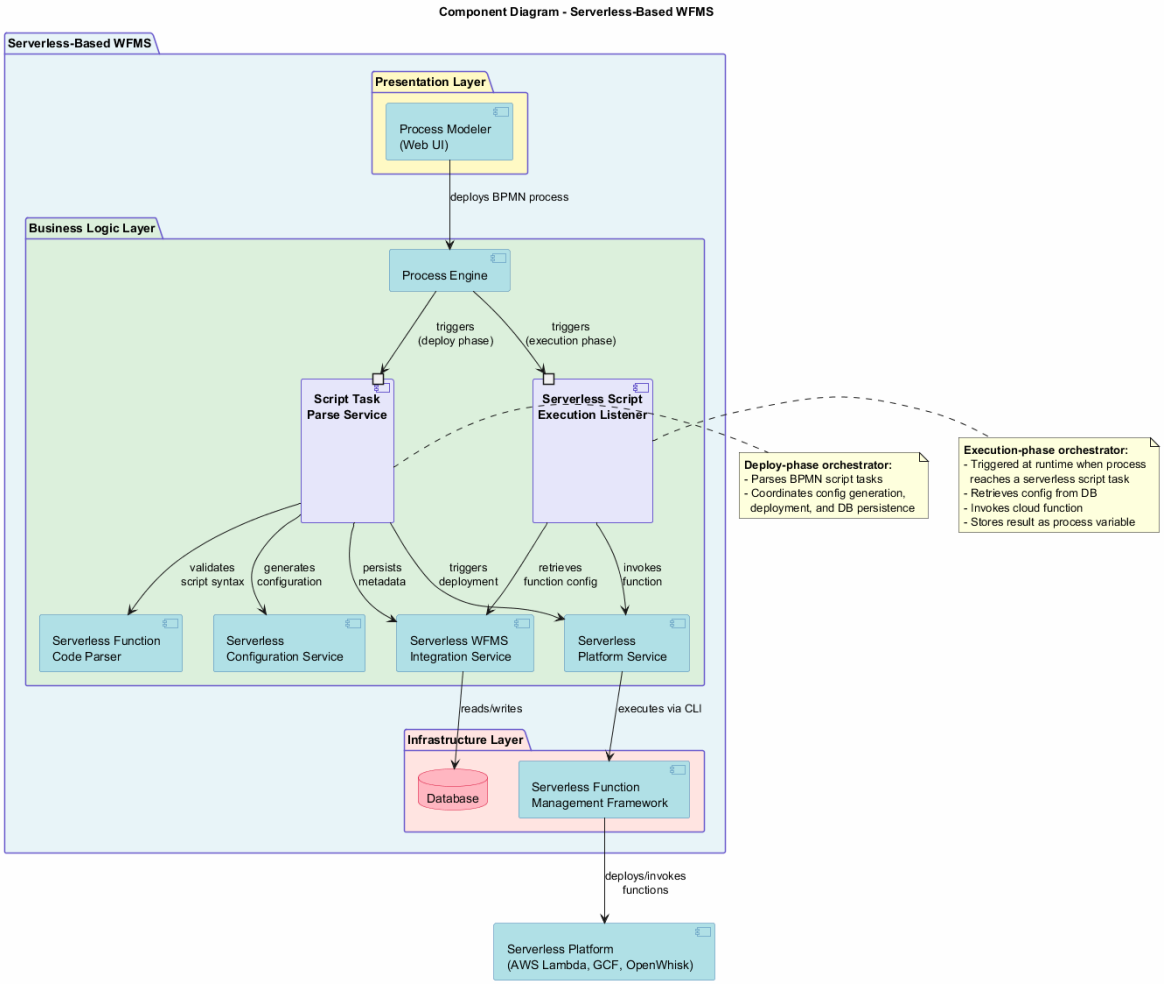
Ένα διάγραμμα συστατικών (Component Diagrams) χρησιμοποιείται για την οπτικοποίηση της αρχιτεκτονικής οργάνωσης των στοιχείων του συστήματος και των σχέσεων εξάρτησης μεταξύ τους. Επίσης, παρέχει μια άποψη υψηλού επιπέδου των στοιχείων ενός συστήματος. Στην Εικόνα 5 φαίνεται το διάγραμμα συστατικών, το οποίο αποτελείται από τα ακόλουθα συστατικά και τις εξαρτήσεις τους.

Τα συστατικά του λογισμικού που αναπτύχθηκε ακολουθούν τη διαστρωματική (layered) αρχιτεκτονική, με το επίπεδο υπηρεσιών (Service Layer) να περιλαμβάνει τα Script Task Parse Service, Serverless Script Execution Listener, Serverless Configuration Service, Serverless WFMS Integration Service και Serverless Platform Service και το επίπεδο υποδομής (Infrastructure Layer) που αποτελείται από τη Βάση Δεδομένων και το Serverless Function Management Framework, δύο συστατικά που εκτελούνται στην υποδομή του συστήματος και παρέχουν υπηρεσίες προς τα συστατικά του επιπέδου επιχειρησιακής λογικής.

Συστατικά:

- **Process Modeler:** Το γραφικό περιβάλλον μέσω του οποίου ο προγραμματιστής σχεδιάζει και δημοσιεύει ροές εργασίας BPMN. Αποστέλλει τον ορισμό της εφαρμογής στον Process Engine για ανάλυση και ανάπτυξη.
- **Process Engine (Flowable):** Η κύρια εφαρμογή με το σύστημα WFMS, η οποία ενεργοποιεί δυο διακριτές φάσεις για την διαχείριση των serverless συναρτήσεων: τη φάση διάταξης/ανάπτυξης (deploy phase), κατά την οποία οι συναρτήσεις παράγονται και αναπτύσσονται στην πλατφόρμα, και τη φάση εκτέλεσης (execution phase), κατά την οποία οι συναρτήσεις καλούνται κατα την εκτέλεση της ροής εργασίας.
- **Script Task Parse Service:** Αποτελεί τον κεντρικό συντονιστή της φάσης διάταξης των συναρτήσεων. Ενεργοποιείται από τη μηχανή WFMS κατά την ανάλυση (parsing) των BPMN αρχείων, επικυρώνει τον κώδικα των script tasks μέσω του Serverless Function Code Parser και συντονίζει τη δημιουργία αρχείων διαμόρφωσης, την διάταξη των serverless συναρτήσεων και την αποθήκευση μεταδεδομένων στην βάση δεδομένων.
- **Serverless Script Execution Listener:** Αποτελεί τον κεντρικό συντονιστή της φάσης εκτέλεσης. Ενεργοποιείται κατά τον χρόνο εκτέλεσης όταν μια διεργασία φτάσει σε ένα (serverless) script task, ανακτά τη διαμόρφωση της συνάρτησης από τη βάση δεδομένων μέσω του Serverless Flowable Integration Service, εκτελεί την κλήση της serverless συνάρτησης μέσω του Serverless Platform Service και αποθηκεύει το αποτέλεσμα ως μεταβλητή διεργασίας (process variable).
- **Serverless Configuration Service:** Διαχειρίζεται λειτουργίες που σχετίζονται με την δημιουργία ή την τροποποίηση των αρχείων διαμόρφωσης του Serverless Framework που ορίζουν τις serverless συναρτήσεις, τον πάροχο cloud και το περιβάλλον εκτέλεσής τους. Καλείται από τον Script Task Parse Service κατά τη φάση ανάπτυξης.
- **Serverless WFMS Integration Service:** Διαχειρίζεται λειτουργίες που σχετίζονται με την δημιουργία, την τροποποίηση και την ανάκτηση μεταδεδομένων των serverless συναρτήσεων στη βάση δεδομένων. Χρησιμοποιείται και από τις δύο φάσεις: ο Script Task Parse Service αποθηκεύει μεταδεδομένα κατά την ανάπτυξη, ενώ ο Serverless Script Execution Listener ανακτά τη διαμόρφωση κατά την εκτέλεση.
- **Serverless Platform Service:** Διαχειρίζεται λειτουργίες που σχετίζονται με την διάταξη και την εκτέλεση των **serverless συναρτήσεων** σε συνεργασία με το Serverless Function Management Service μέσω CLI. Χρησιμοποιείται και από τις δύο φάσεις: για την ανάπτυξη (deploy) κατά τη φάση ανάπτυξης και για την κλήση (invoke) κατά τη φάση εκτέλεσης.
- **Database:** Η βάση δεδομένων στην οποία αποθηκεύονται τα μεταδεδομένα των serverless συναρτήσεων (endpoints, script hashes, providers κ.λπ.) μέσω του Serverless WFMS Integration Service.
- **Serverless Function Management Service:** Εσωτερικό συστατικό υποδομής του συστήματος. Αποτελεί εργαλείο CLI για την ανάπτυξη και εκτέλεση serverless συναρτήσεων σε πολλαπλούς παρόχους cloud (AWS, GCP, OpenWhisk).

- **Serverless Function Code Parser**: Συστατικό υπεύθυνο για την επικύρωση του συντακτικού (syntax validation) του κώδικα των serverless συναρτήσεων. Χρησιμοποιεί την γραμματική ECMAScript μέσω του ANTLR4 για τον εντοπισμό συντακτικών σφαλμάτων.



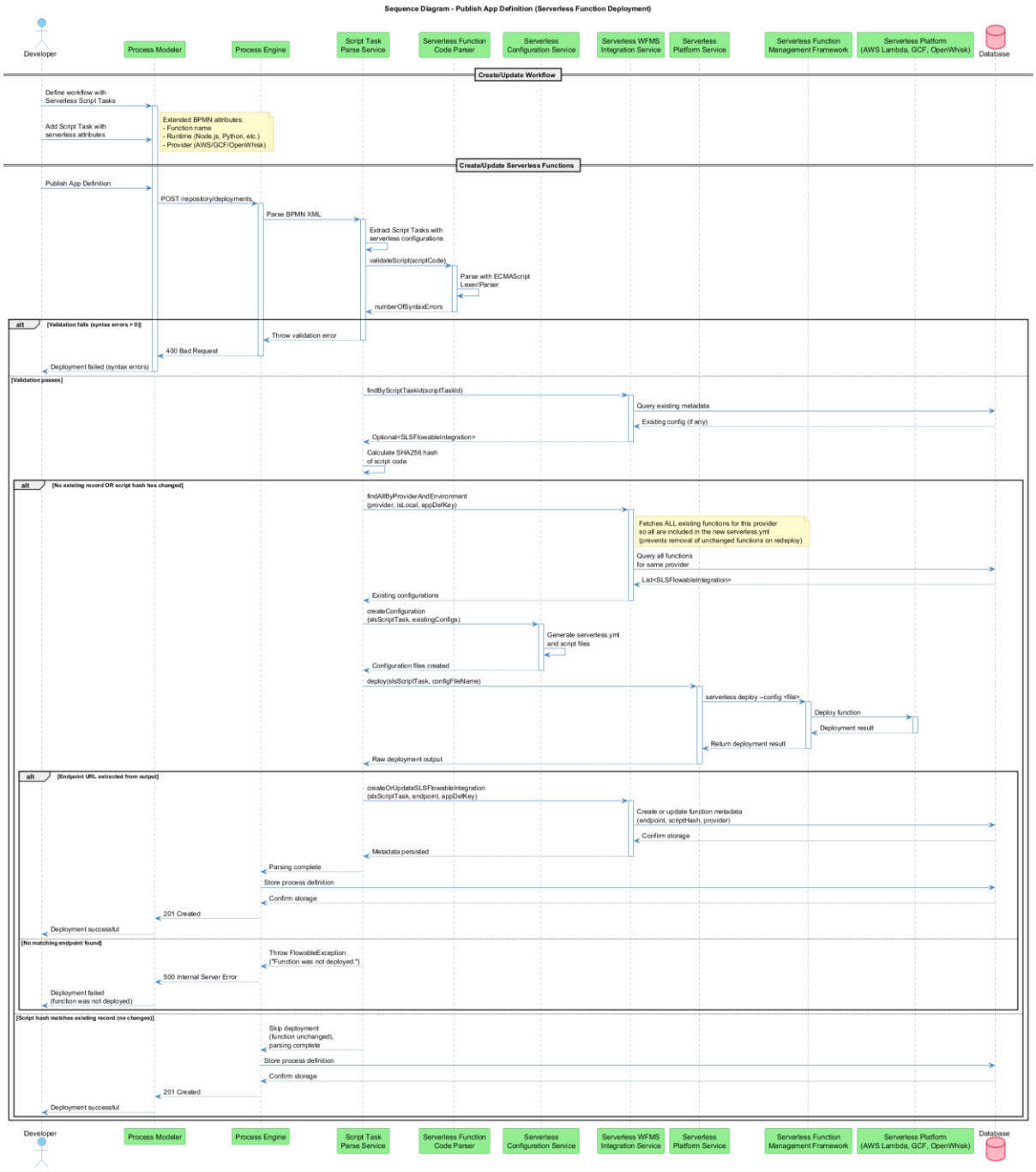
ΕΙΚΟΝΑ 5 - ΔΙΑΓΡΑΜΜΑ ΣΥΣΤΑΤΙΚΩΝ (COMPONENT DIAGRAM)

4.2.4. Διάγραμμα Ακολουθίας

Στην Εικόνα 6 παρουσιάζεται το διάγραμμα ακολουθίας για τις περίπτωσεις χρήσης Create/Update Workflow και Create/Update Serverless Functions. Συνεπώς, βλέπουμε την αλληλεπίδραση του προγραμματιστή με το **WFMS** κατά τη φάση διάταξης (deploy phase). Αρχικά, ο προγραμματιστής ορίζει τη ροή εργασίας μέσω του Process Modeler, προσθέτοντας Script Tasks με χαρακτηριστικά serverless (όνομα συνάρτησης, πάροχος cloud, runtime κ.λπ.). Κατά την υποβολή του ορισμού εφαρμογής (App Definition), η μηχανή WFMS ξεκινά την ανάλυση (parsing) του BPMN αρχείου μέσω του Script Task Parse Service. Η ενέργεια αυτή (Publish) πραγματοποιείται από τον προγραμματιστή μέσω του Process Modeler. Σημειώνεται πως ένα App Definition μπορεί να περιλαμβάνει πολλαπλές ροές εργασίας. Για κάθε Script Task που περιέχει δήλωση serverless συνάρτησης εκτελούνται τα εξής βήματα:

1. Επικύρωση του κώδικα της συνάρτησης μέσω του Serverless Function Code Parser για τον εντοπισμό συντακτικών σφαλμάτων. Αν εντοπιστούν συντακτικά σφάλματα, η διάταξη ακυρώνεται και ο Process Modeler επιστρέφει σφάλμα.
2. Αναζήτηση υπάρχουσας διαμόρφωσης στη βάση δεδομένων μέσω του Serverless Flowable Integration Service και υπολογισμός SHA256 hash του κώδικα.

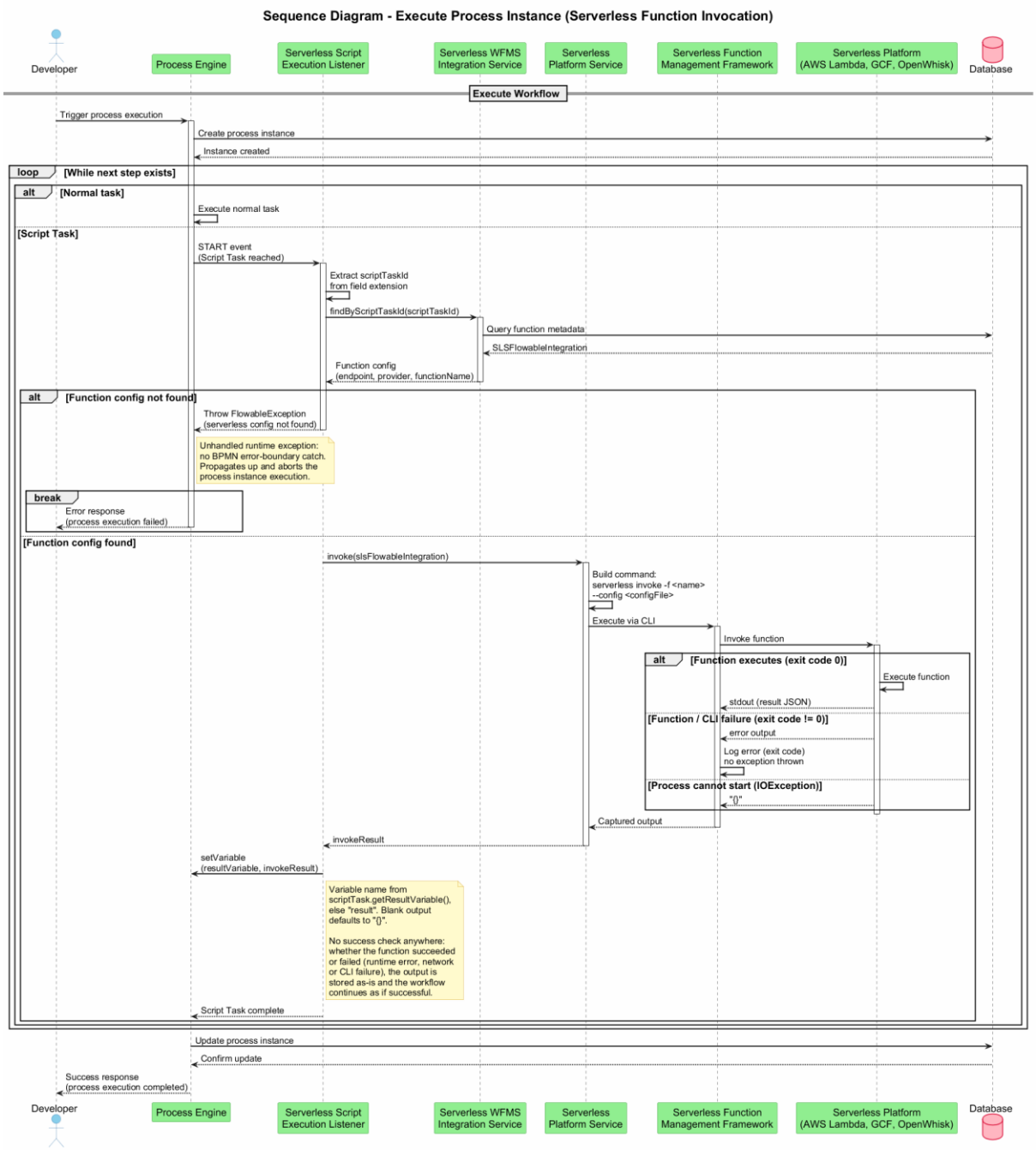
3. Αν το hash ταυτίζεται με τον υπάρχον (δηλαδή ο κώδικας δεν έχει αλλάξει), η διάταξη παραλείπεται για αποφυγή περιττών deployments.
4. Αν ο κώδικας είναι νέος ή τροποποιημένος, ανακτώνται όλες οι υπάρχουσες συναρτήσεις του ίδιου παρόχου ώστε να συμπεριληφθούν στην νέα διαμόρφωση και να αποφευχθεί η αφαίρεση τους κατά την επανυποβολή. Στην συνέχεια δημιουργούνται τα αρχεία διαμόρφωσης YAML μέσω του Serverless Configuration Service, εκτελείται η ανάπτυξη της συνάρτησης μέσω του Serverless Platform Service σε συνεργασία με το Serverless Function Management Framework. Αν εξαχθεί επιτυχώς το endpoint της συνάρτησης από το αποτέλεσμα της ανάπτυξης, αποθηκεύονται τα μεταδεδομένα στη βάση δεδομένων. Διαφορετικά, εγείρεται FlowableException και ο Process Modeler επιστρέφει σφάλμα.



ΕΙΚΟΝΑ 6 - ΔΙΑΓΡΑΜΜΑ ΑΚΟΛΟΥΘΙΑΣ (CREATE/UPDATE WORKFLOW, CREATE/UPDATE SERVERLESS FUNCTIONS)

Στην Εικόνα 7 παρουσιάζεται το διάγραμμα ακολουθίας για τις περιπτώσεις χρήσης Execute Workflow και Execute Serverless Functions. Βλέπουμε την αλληλεπίδραση του προγραμματιστή με το σύστημα κατά τη φάση εκτέλεσης (execution phase). Ο προγραμματιστής εκκινεί την εκτέλεση μιας ροής εργασίας μέσω κλήσης προς τη μηχανή (Process Engine) και αυτή με την σειρά της δημιουργεί ένα στιγμιότυπο διεργασίας (process instance) και η εκτέλεση προχωρά εντός της ίδιας κλήσης. Στη συνέχεια, η μηχανή επαναλαμβάνει τα βήματα της ροής εφόσον υπάρχει επόμενο βήμα. Για κάθε βήμα ελέγχεται ο τύπος του αν πρόκειται για κανονικό task εκτελείται από τη μηχανή απευθείας. Αν πρόκειται για Script Task που περιέχει serverless συνάρτηση, ενεργοποιείται ο Serverless Script Execution Listener, ο οποίος:

1. Εξάγει το scriptTaskId από τα χαρακτηριστικά του task.
2. Ανακτά τη διαμόρφωση της συνάρτησης από τη βάση δεδομένων μέσω του Serverless WFMS Integration Service. Αν δεν βρεθεί η διαμόρφωση, προκαλείται εξαίρεση τύπου FlowableException. Εφόσον δεν έχει οριστεί στο μοντέλο κάποιο BPMN error boundary event που να την πιάνει, η εξαίρεση διαδίδεται προς τα πάνω, η επανάληψη τερματίζεται μη κανονικά και το στιγμιότυπο διεργασίας ματαιώνεται και η κλήση εκκίνησης της ροής αποτυγχάνει με σφάλμα, το οποίο επιστρέφεται στον προγραμματιστή που την εκκίνησε.
3. Αν η διαμόρφωση βρεθεί, ο listener καλεί την serverless συνάρτηση μέσω του Serverless Platform Service και αποθηκεύει το αποτέλεσμα ως μεταβλητή διαδικασίας (process variable), πριν η ροή προχωρήσει στο επόμενο βήμα. Δεν πραγματοποιείται κανένας έλεγχος επιτυχίας: σε αντίθεση με την απούσα διαμόρφωση, η αποτυχία εκτέλεσης της συνάρτησης δεν διακόπτει τη ροή. Ανεξάρτητα από το αν η συνάρτηση πέτυχε ή απέτυχε (π.χ. runtime error, αποτυχία δικτύου, σφάλμα CLI), η έξοδος αποθηκεύεται ως έχει ή ως «{}» αν δεν επιστραφεί τίποτα και η ροή συνεχίζεται σαν να ήταν επιτυχής.
4. Με την ολοκλήρωση της επανάληψης χωρίς σφάλμα η μηχανή εκτέλεσης ενημερώνει το στιγμιότυπο διεργασίας στη βάση δεδομένων και επιστρέφει απάντηση επιτυχίας στον προγραμματιστή σηματοδοτώντας την ολοκλήρωση της εκτέλεσης.



ΕΙΚΟΝΑ 7 - ΔΙΑΓΡΑΜΜΑ ΑΚΟΛΟΥΘΙΑΣ (EXECUTE WORKFLOW, EXECUTE SERVERLESS FUNCTIONS)

4.2.5. Διάγραμμα Οντοτήτων Συσχετίσεων

Στην Εικόνα 8 παρουσιάζεται το διάγραμμα οντοτήτων-συσχετίσεων του συστήματος. Το διάγραμμα περιλαμβάνει πέντε οντότητες:

1. **AppDefinition**: αντιπροσωπεύει τον ορισμό εφαρμογής που αναπτύσσει ο προγραμματιστής μέσω του Flowable Modeler.
   Ιδιότητες:
   - **id:** μοναδικό αναγνωριστικό της διάταξης (deployment) στη μηχανή Flowable
   - **name:** το αναγνωρίσιμο όνομα του ορισμού εφαρμογής όπως δίνεται στο Flowable Modeler.

- **key:** το κλειδί του ορισμού εφαρμογής, σταθερό μεταξύ διαφορετικών εκδόσεων διάταξης.
- **deployTime:** χρονοσήμανση που δείχνει πότε ο ορισμός εφαρμογής αναπτύχθηκε στην μηχανή.
- **tenantId:** αναγνωριστικό μισθωτή (tenant), για απομόνωση δεδομένων σε περιβάλλον πολλαπλών μισθωτών (multi-tenancy).

2. **ProcessDefinition**: αντιστοιχεί σε μια διεργασία BPMN που περιέχεται στον ορισμό εφαρμογής.
   Ιδιότητες:
   - **id**: μοναδικό αναγνωριστικό της συγκεκριμένης έκδοσης της διαδικασίας.
   - **name**: το όνομα της διαδικασίας όπως ορίζεται στο BPMN μοντέλο.
   - **key**: το κλειδί της διαδικασίας, σταθερό ανά εκδόσεις.
   - **version**: ο αριθμός έκδοσης, ο οποίος αυξάνεται αυτόματα σε κάθε νέα ανάπτυξη της ίδιας διαδικασίας.
3. **ProcessInstance**: αναπαριστά ένα στιγμιότυπο εκτέλεσης μιας διαδικασίας.
   Ιδιότητες:
   - **id**: μοναδικό αναγνωριστικό του στιγμιότυπου εκτέλεσης.
   - **businessKey**: προαιρετικό εξωτερικό αναγνωριστικό που συνδέει το στιγμιότυπο με μια επιχειρησιακή οντότητα.
   - **isActive**: λογική τιμή που δείχνει αν το στιγμιότυπο βρίσκεται ακόμη σε εξέλιξη.
   - **tenantId**: αναγνωριστικό μισθωτή, κληρονομείται από τον ορισμό διεργασίας.
4. **ProcessVariable**: αποθηκεύει τις μεταβλητές (και τις τιμές τους) που παράγονται κατά την εκτέλεση μιας διαδικασίας (συμπεριλαμβανομένων των αποτελεσμάτων serverless συναρτήσεων).
   Ιδιότητες:
   - **id**: μοναδικό αναγνωριστικό της μεταβλητής.
   - **name**: το όνομα της μεταβλητής όπως αναφέρεται σε εκφράσεις του BPMN μοντέλου.
   - **type**: ο τύπος δεδομένων της τιμής.
   - **value**: η αποθηκευμένη τιμή, η οποία μπορεί να περιλαμβάνει και τα αποτελέσματα εκτέλεσης serverless συνάρτησης.
5. **ServerlessIntegration**: αποτελεί την προσαρμοσμένη οντότητα του συστήματος και αποθηκεύει τα μεταδεδομένα κάθε serverless συνάρτησης (endpoint, cloud πάροχος, hash του κώδικα).
   Ιδιότητες:
   - **id**: μοναδικό αναγνωριστικό της εγγραφής ολοκλήρωσης.
   - **scriptTaskId**: το αναγνωριστικό του Script Task στο BPMN μοντέλο στο οποίο αντιστοιχεί η serverless συνάρτηση.
   - **provider**: ο πάροχος cloud στον οποίο έχει διαταχθεί η συνάρτηση.
   - **functionName**: το όνομα της διεταγμένης συνάρτησης στον πάροχο cloud.
   - **scriptHash**: το SHA-256 hash του κώδικα της συνάρτησης, που χρησιμοποιείται για ανίχνευση αλλαγών και επανα-διάταξη.
   - **endpoint**: το σημείο πρόσβασης (URL/ARN) της διεταγμένης συνάρτησης.
   - **local**: λογική τιμή που δηλώνει αν η συνάρτηση εκτελείται τοπικά ή στο cloud.

Οι συσχετίσεις μεταξύ οντοτήτων είναι οι εξής: μια AppDefinition περιέχει πολλές ProcessDefinition, μια ProcessDefinition μπορεί να έχει πολλά ProcessInstance, κάθε ProcessInstance έχει πολλές

ProcessVariable και κάθε ServerlessIntegration αναφέρεται σε μια ProcessDefinition μέσω του πεδίου appDefKey.

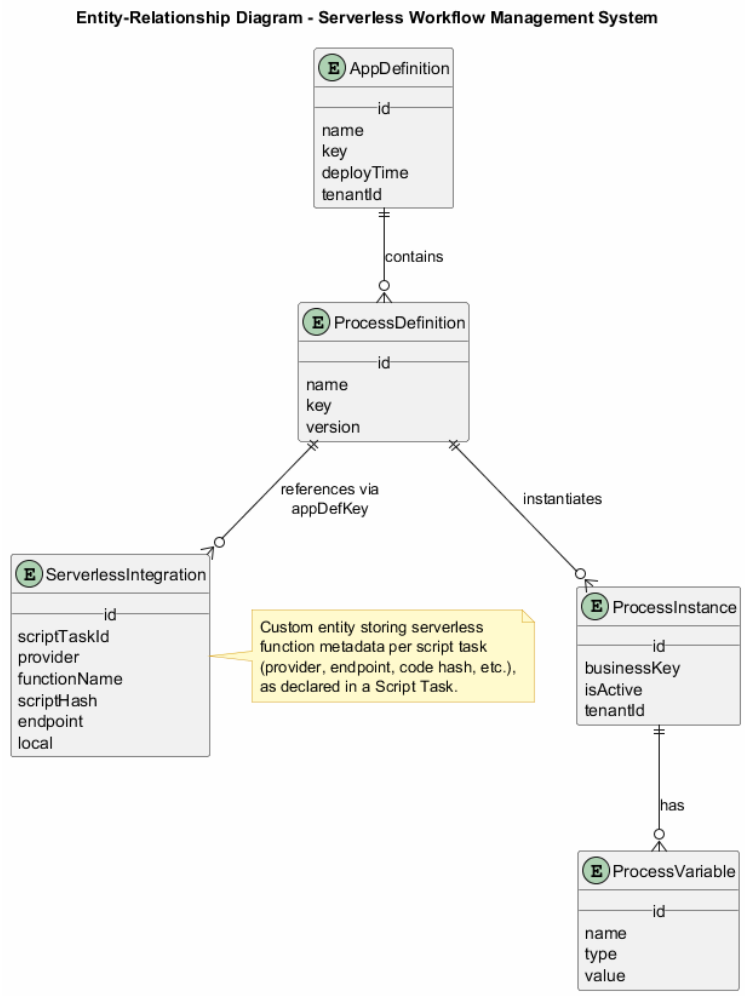
Οι πληθικότητες των συσχετίσεων του διαγράμματος αιτιολογούνται ως εξής:

AppDefinition – ProcessDefinition (1 προς πολλά): ένας ορισμός εφαρμογής στο Flowable Modeler μπορεί να περιέχει πολλές διαδικασίες BPMN, οι οποίες αναπτύσσονται μαζί ως ενιαία μονάδα.

ProcessDefinition – ProcessInstance (1 προς πολλά): μια διαδικασία μπορεί να εκκινηθεί πολλές φορές παράγοντας πολλά ανεξάρτητα στιγμιότυπα εκτέλεσης, καθένα με τη δική του κατάσταση και τις δικές του μεταβλητές.

ProcessInstance – ProcessVariable (1 προς πολλά): κατά τη διάρκεια εκτέλεσης ενός στιγμιοτύπου παράγονται και αποθηκεύονται πολλές μεταβλητές (είσοδοι/έξοδοι εργασιών, συμπεριλαμβανομένων των αποτελεσμάτων serverless συναρτλησεων).

ProcessDefinition – ServerlessIntegration (1 προς πολλά): μια διαδικασία μπορεί να περιέχει πολλά Script Tasks και κάθε ένα από αυτά μπορεί να υλοποιείται από διαφορετική serverless συνάρτηση. Επομένως, μια ProcessDefinition μπορεί να αντιστοιχεί σε πολλαπλές εγγραφές ServerlessIntegration, μία ανά Script Task με τη σύνδεση να γίνεται λογικά μέσω του πεδίου appDefKey που αντιστοιχεί στο key της διαδικασίας.



ΕΙΚΟΝΑ 8 - ΔΙΑΓΡΑΜΜΑ ΟΝΤΟΤΗΤΩΝ-ΣΥΣΧΕΤΙΣΕΩΝ

4.3. Υλοποίηση

4.3.1. Υλοποίηση Αρχιτεκτονικής Συστήματος

Για την υλοποίηση του συστήματος επιλέχθηκε ως WFMS το Flowable, ένα ανοιχτού κώδικα BPMN 2.0 engine βασισμένο σε Java/Spring Boot. Το Flowable WFMS επιλέχθηκε λόγω της ευελιξίας του ως embeddable engine, της ενσωμάτωσής του με το Spring Boot οικοσύστημα, της υποστήριξης επεκτασιμότητας μέσω custom parse handlers και execution listeners, καθώς και της ενεργής κοινότητας και της ωριμότητας της πλατφόρμας. Ως εργαλείο διαχείρισης και ανάπτυξης serverless συναρτήσεων επιλέχθηκε το Serverless Function Management Framework (Node.js CLI). Η επέκταση υλοποιήθηκε ως Spring Boot module που ενσωματώνεται άμεσα στον πυρήνα του Flowable.

Αναλυτικότερα πίσω από κάθε συστατικό του διαγράμματος συστατικών (Εικόνα 5) βρίσκεται:

- *Script Task Parse Service*: Επέκταση του ScriptTaskParseHandler του Flowable. Μετατρέπει το ScriptTask αντικείμενο σε εσωτερική αναπαράσταση SLSScriptTask, εκχωρεί no-operation (no-op) έκφραση στο Script Task ώστε η μηχανή Flowable να μην αποτύχει κατά την επικύρωση του κώδικα, και ελέγχει αν ο κώδικας έχει αλλάξει μέσω sha-256 hash σύγκρισης πριν εκκινήσει τη διάταξη.
- *Script Function Code Parser*: Spring υπηρεσία βασισμένη στο ANTLR ECMAScript, υπεύθυνη για τη συντακτική ανάλυση και επικύρωση των scripts που ορίζονται στα Script Tasks του BPMN διαγράμματος.
- *Serverless Script Execution Listener*: Υλοποίηση ExecutionListener του Flowable, ο οποίος ενεργοποιείται αυτόματα κατά την εκτέλεση Script Tasks και εκκινεί την κλήση της αντίστοιχης serverless συνάρτησης.
- *Serverless Configuration Service*: Spring υπηρεσία υπεύθυνη για την δημιουργία και διαχείριση του αρχείου ρυθμίσεων serverless.yml που χρησιμοποιείται από το Serverless Framework για την διάταξη των συναρτήσεων.
- *Serverless WFMS Integration Service*: Spring υπηρεσία, η οποία διαχειρίζεται την αντιστοίχιση μεταξύ Script Tasks και serverless συναρτήσεων. Για την επικοινωνία με τη βάση δεδομένων χρησιμοποιείται η τεχνολογία Spring Data JPA για την υλοποίηση ORM, ώστε να αποκρύπτονται οι low-level λεπτομέρειες διασύνδεσης με την υποκείμενη βάση δεδομένων.
- *Serverless Platform Service*: Spring υπηρεσία υπεύθυνη για την διάταξη (deploy) και εκτέλεση (invoke) serverless συναρτήσεων στις υποστηριζόμενες πλατφόρμες (AWS Lambda, Google Cloud Functions, Apache OpenWhisk) μέσω κλήσεων στο Serverless Framework CLI (που υλοποιεί το συστατικό Serverless Function Management Service).

4.3.2. Περιγραφή Επαναχρησιμοποιούμενων Εργαλείων & Συστημάτων

Σε αυτήν την ενότητα περιγράφονται αναλυτικά πληροφορίες σχετικά με τα επαναχρησιμοποιούμενα εργαλεία και συστήματα που επιλέχθηκαν για την υλοποίηση του συστήματος καθώς και πληροφορίες για τον τρόπο λειτουργίας, τους όπου κρίθηκε απαραίτητο, για να γίνει κατανοητή η ενσωμάτωση της λύσης.

4.3.2.1. Flowable WFMS

Για την διεκπεραίωση της διπλωματικής επιλέχθηκε το σύστημα διαχείρισης ροών εργασίας (WFMS) **Flowable**. Κάποιοι από τους λόγους που επιλέχθηκε το συγκεκριμένο εργαλείο είναι οι εξής:

1. Το **Flowable** είναι μια ελαφριά, ανοιχτού κώδικα μηχανή εκτέλεσης και διαχείρισης ροών εργασίας, η οποία παρέχει πλήρη υποστήριξη στο πρότυπο BPMN 2.0. Επιτρέπει την

ανάπτυξη ορισμών εργασίας, τη δημιουργία και εκτέλεση στιγμιότυπων τους, την εκτέλεση ερωτημάτων καθώς και την παρακολούθηση ιστορικού και σχετικών δεδομένων των διεργασιών. Η επιλογή του Flowable ως WFMS για την υλοποίηση της παρούσας εργασίας βασίστηκε ακριβώς στην πλήρη υποστήριξη του στο πρότυπο BPMN 2.0 με το οποίο ορίζονται και εκτελούνται οι ροές εργασίας που εξετάζονται στη διπλωματική.

2. Το **Flowable** είναι υλοποιημένο σε Java, η οποία είναι μια γλώσσα οικεία στο συγγραφέα της διπλωματικής και προγραμματιστή του επεκταμένου συστήματος WFMS.

3. Το **Flowable** είναι ένα ανοιχτού κώδικα λογισμικό, το οποίο μας επιτρέπει να το επεκτείνουμε ή ακόμα και να τροποποιήσουμε τις υπάρχουσες λειτουργίες του. Άρα, προσφέρει και ευελιξία όσον αφορά τον τρόπο χρήσης και ενσωμάτωσης του σε άλλα συστήματα.

4. Το **Flowable** παρέχει επίσης διάφορες εφαρμογές, οι οποίες παρέχουν ένα ολοκληρωμένο γραφικό περιβάλλον για την δημιουργία ροών εργασίας (**Flowable Modeler**). Αυτές οι εφαρμογές είναι διαθέσιμες και μπορούν να προστεθούν σε οποιαδήποτε εφαρμογή/υπηρεσία/αρχιτεκτονική.

5. Επιπροσθέτως, είναι ένα γνώριμο εργαλείο καθώς έχει χρησιμοποιηθεί στο παρελθόν και είναι αρκετά πλήρες για την εκπόνηση αυτής της διπλωματικής και τους βασικούς της στόχους.

Κάποιες από τις κεντρικές έννοιες στο Flowable που αξίζει να αναφερθούν είναι οι εξής:

1. **App Definition (Ορισμός Εφαρμογής):** Το Flowable ορίζει μια εφαρμογή (App) ως ένα πακέτο που περιλαμβάνει πολλαπλά Process Models (Μοντέλα Διαδικασίας), φόρμες, scripts και άλλα στοιχεία που απαιτούνται για την υλοποίηση μιας επιχειρησιακής διαδικασίας/ ροής εργασίας.
2. **Process Models (Μοντέλα Διαδικασίας):** Οι διαδικασίες στο Flowable σχεδιάζονται ως διαγράμματα BPMN (Business Process Model and Notation). Κάθε διαδικασία περιγράφει τη ροή εργασιών, καθορίζοντας tasks, αποφάσεις, γεγονότα (events) και μεταβάσεις (transitions).
3. **Script Tasks (Εργασίες Script):** Στα Process Models, τα Script Tasks επιτρέπουν την εκτέλεση προσαρμοσμένου κώδικα μέσα στη ροή εργασίας. Για παράδειγμα, μπορούν να χρησιμοποιηθούν για την κλήση εξωτερικών APIs ή την εκτέλεση σύνθετης λογικής.
4. **Execution Listeners**: Αυτά είναι ακροατές που ενεργοποιούνται σε καθορισμένα γεγονότα μέσα στη ροή εργασίας καθώς αυτή εκτελείται (π.χ. όταν ξεκινά ή ολοκληρώνεται μια εργασία/task). Μπορούν να χρησιμοποιηθούν για την εκτέλεση πρόσθετου κώδικα και την αποθήκευση του αποτελέσματος ως μεταβλητή διαδικασίας (process variable).

4.3.2.2. Γλώσσα Προγραμματισμού και Πλαίσιο Ανάπτυξης

Η γλώσσα προγραμματισμού που χρησιμοποιήθηκε είναι η **Java (v18.0.1.1)**. Για την υλοποίηση χρησιμοποιήθηκε το πλαίσιο ανάπτυξης εφαρμογών & υπηρεσιων ιστού **Spring Boot (v2.1.7),** το οποίο διαθέτει τα ακόλουθα χαρακτηριστικά:

1. Το Spring Boot παρέχει απρόσκοπτη ενοποίηση με το Flowable μέσω του Java API, καθιστώντας εύκολη τη διαμόρφωση και τη χρήση του Flowable σε μια εφαρμογή που βασίζεται στο Spring.

2. Το πλαίσιο διαχείρισης συναλλαγών (transaction management framework) του Spring παρέχει ισχυρή υποστήριξη συναλλαγών για Flowable διαδικασίες, διασφαλίζοντας ότι οι διαδικασίες εκτελούνται με συναλλακτικό (transactional) τρόπο. Στην περίπτωση αυτής της

διπλωματικής, το transaction management framework του Spring μπορεί να διασφαλίσει ότι οι εργασίες ροής εργασιών εκτελούνται ατομικά και με συνέπεια, με κατάλληλους μηχανισμούς επαναφοράς και ανάκτησης σε περίπτωση σφαλμάτων ή εξαιρέσεων (exceptions). Αυτό μπορεί να συμβάλει στη βελτίωση της αξιοπιστίας και της ακεραιότητας των δεδομένων του συστήματος ροής εργασιών, ενώ θα διευκολύνει τη διαχείριση και την κλιμάκωση καθώς μεγαλώνει η εφαρμογή (scaling).

3. Επιπροσθέτως, η χρήση του Spring Boot μπορεί να απλοποιήσει τη διαδικασία ανάπτυξης, μειώνοντας την ποσότητα του κώδικα boilerplate που πρέπει να γραφτεί και διευκολύνοντας τη σύνταξη καθαρού, συντηρήσιμου κώδικα.

4. Το Serverless Framework παρέχει τη δυνατότητα τοπικής εκτέλεσης των συναρτήσεων χωρίς να απαιτείται ανάπτυξη σε κάποιον πάροχο cloud. Αυτή η δυνατότητα αποτελεί βασική απαίτηση του συστήματος, καθώς επιτρέπει την εκτέλεση serverless συναρτήσεων σε τοπικό περιβάλλον (on-premise), χωρίς εξάρτηση από εξωτερικές υποδομές

Σε συνδυασμό με το Spring Boot χρησιμοποιήθηκε το Java Persistence API (JPA), το οποίο είναι μια προδιαγραφή Java για Σχεσιακή Χαρτογράφηση Αντικειμένων (ORM) που διευκολύνει την (αυτόματη) αντιστοίχιση αντικειμένων Java σε πίνακες σχεσιακών βάσεων δεδομένων, όπως η PostgreSQL, απλοποιώντας με αυτό τον τρόπο τη διασύνδεση με την βάση και απελευθερώνοντας τον προγραμματιστή από λεπτομέρειες διασύνδεσης χαμηλού επιπέδου. Το Spring Boot έχει ενσωματωμένη υποστήριξη για JPA μέσω της λειτουργικής μονάδας Spring Data JPA, η οποία παρέχει έναν ισχυρό και ευέλικτο τρόπο εργασίας με βάσεις δεδομένων σε μια εφαρμογή Spring.

Επίσης , το Spring Boot παρέχει πολλά οφέλη, όπως:

1. Βελτιωμένη δυνατότητα συντήρησης (**Improved maintainability**)
2. Αυξημένη παραγωγικότητα (**Increased productivity**)

4.3.2.3. Serverless Framework

Το **Serverless Framework** είναι ένα εργαλείο γραμμής εντολών (CLI) που μπορεί να χρησιμοποιηθεί στην ανάπτυξη και διαχείριση λειτουργιών χωρίς διακομιστή (serverless functions) σε πολλές serverless πλατφόρμες, συμπεριλαμβανομένου των AWS Lambda, Google Cloud Functions και OpenWhisk. Επίσης, προσφέρει πρόσθετα πλεονεκτήματα όπως:

1. *Υποστήριξη μεταξύ πλατφορμών (Cross-Platform support):* Το Serverless Framework υποστηρίζει πολλαπλές serverless πλατφόρμες, διευκολύνοντας την εναλλαγή μεταξύ παρόχων/πλατφορμών ή τη χρήση υβριδικής προσέγγισης δηλαδή πολλών serverless πλατφορμών μαζί (πχ. στο πλαίσιο υλοποίησης της ίδιας ή διαφορετικών συναρτήσεων & serverless εφαρμογών).
2. *Απλοποιημένη Ανάπτυξη*: Το Serverless Framework απλοποιεί την διαδικασία ανάπτυξης, αυτοματοποιώντας τη δημιουργία των απαραίτητων πόρων και παρέχοντας ένα ενοποιημένο αρχείο διαμόρφωσης (YAML configuration file), στο οποίο γράφεται όλη η πληροφορία για την συνάρτηση ή τις συναρτήσεις, όπως ποια serverless πλατφόρμα θα φιλοξενηθούν και διάφορες άλλες παραμέτρους που θα δούμε στην συνέχεια.
3. *Ορατότητα & έλεγχος συναρτήσεων*: Ακόμα παρέχει παρέχει μια διεπαφή ιστού, η οποία περιέχει στατιστικά σχετικά με τις συναρτήσεις, όπως πόσες φορές εκτελέστηκαν, αν υπάρχουν σφάλματα κατά την κλήση καθώς και ένα περιβάλλον για τη δοκιμή αυτών των συναρτήσεων.
4. Το Serverless Framework παρέχει τη δυνατότητα τοπικής εκτέλεσης των συναρτήσεων χωρίς να απαιτείται ανάπτυξη της σε κάποια (απομακρυσμένη) serverless πλατφόρμα. Αυτή η

δυνατότητα αποτελεί βασική απαίτηση του συστήματος, καθώς επιτρέπει την εκτέλεση serverless συναρτήσεων σε τοπικό περιβάλλον (on-premise), χωρίς εξάρτηση από εξωτερικές υποδομές. Ένα τέτοιο χαρακτηριστικό, όπως έχει ήδη ειπωθεί, επιτρέπει επίσης τη δοκιμή των συναρτήσεων, πριν αυτές μεταφερούν σε ένα παραγωγικό περιβάλλον (πχ. μιας serverless πλατφόρμας).

Αξίζει να αναφερθούν μερικές πληροφορίες σχετικά για τον τρόπο που λειτουργεί το **Serverless Framework** έτσι ώστε να γίνει κατανοητό πώς χρησιμοποιείται στην συνέχεια.

1. **Ορισμός Υπηρεσίας (Service Definition):** Οι προγραμματιστές δημιουργούν ένα αρχείο διαμόρφωσης, συνήθως με την ονομασία serverless.yml, που περιγράφει τις λειτουργίες, τα γεγονότα και τους πόρους της εφαρμογής χωρίς εξυπηρετητές. Στο αρχείο αυτό καθορίζονται οι υπηρεσίες, όπως οι **AWS Lambda functions**, καθώς και τα γεγονότα που τις ενεργοποιούν (π.χ. HTTP αιτήματα, γεγονότα από βάσεις δεδομένων).
2. **Διάταξη (Deployment):** Χρησιμοποιώντας την CLI εφαρνογή του Serverless, οι προγραμματιστές διατάσσουν τις υπηρεσίες με την εντολή serverless deploy. Αυτή η εντολή ανεβάζει τις συναρτήσεις στο νέφος (δηλ. σε μια serverless πλατφόρμα) και δημιουργεί την υποδομή (υποστήριξης των συναρτήσεων) σύμφωνα με την διαμόρφωση στο serverless.yml, περιλαμβάνοντας οποιεσδήποτε απαιτούμενες άδειες, μεταβλητές περιβάλλοντος ή τελικά σημεία API.
3. **Εκτέλεση (Execution):** Όταν προκύψουν τα καθορισμένα γεγονότα (π.χ. ένα HTTP αίτημα ή ένα νέο μήνυμα σε μια ουρά), οι λειτουργίες εκτελούνται αυτόματα από τον πάροχο της serverless πλατφόρμας στην οποία έχουν διαταχθεί. Το Serverless Framework διευκολύνει τη δημιουργία και ρύθμιση των απαραίτητων πόρων για την υποδοχή αυτών των γεγονότων, επιτρέποντας την ορθή λειτουργία, κλιμάκωση και βελτιστοποίηση του κόστους εκτέλεσης.
4. **Παρακολούθηση και Διαχείριση (Monitoring and Management):** Το Serverless Framework παρέχει εργαλεία παρακολούθησης και διαχείρισης των αναπτυγμένων συναρτήσεων, επιτρέποντας στους προγραμματιστές να παρακολουθούν την απόδοσή τους, να θέτουν ειδοποιήσεις και να βλέπουν τα αρχεία καταγραφής (logs) για να εντοπίζουν και να επιλύουν πιθανά προβλήματα.

4.3.2.4. Βάση Δεδομένων

Για βάση δεδομένων επιλέχθηκε η **PostgreSQL** η οποία προσφέρει τα ακόλουθα χαρακτηριστικά:

1. Είναι ένα σύστημα διαχείρισης σχεσιακών βάσεων δεδομένων ανοιχτού κώδικα, που σημαίνει ότι προσφέρεται καθώς και μπορεί να προσαρμοστεί ανάλογα με τις ανάγκες που υπάρχουν.
2. Η PostgreSQL είναι εξαιρετικά κλιμακώσιμη και μπορεί να διαχειριστεί μεγάλες ποσότητες δεδομένων και χρηστών.
3. Ακολουθεί τις ιδιότητες ACID (Atomicity, Consistency, Isolation, Durability) μιας (σχεσιακής) βάσης δεδομένων, διασφαλίζοντας τη συνέπεια και την αξιοπιστία των δεδομένων.
4. Παρέχει προηγμένες δυνατότητες, όπως πολύπλοκα ερωτήματα, δημιουργία ευρετηρίου και υποστήριξη για διάφορους τύπους δεδομένων, καθιστώντας την κατάλληλη για πολύπλοκες δομές δεδομένων.

4.3.2.5. Antlr

Για την επικύρωση του κώδικα των serverless συναρτήσεων χρησιμοποιήθηκε το εργαλείο **Antlr4** (**Another Tool for Language Recognition v4.11.1**), το οποίο είναι μια ισχυρή γεννήτρια ανάλυσης (analysis generator) που μπορεί να χρησιμοποιηθεί για τη δημιουργία αναλυτών, μεταγλωττιστών και διερμηνέων για γλώσσες προγραμματισμού και γλώσσες ειδικών ως προς τον τομέα (domain-

specific languages). Πιο συγκεκριμένα, χρησιμοποιήθηκε για να αναγνωρίσει τα συντακτικά σφάλματα του κώδικα των συναρτήσεων χωρίς εξυπηρετητές (serverless function) επειδή είναι μια από τις λειτουργικές απαιτήσεις της εφαρμογής αλλά και επειδή το Flowable επιτρέπει την επικύρωση του κώδικα σε λίγες scripting γλώσσες, θέτοντας ένα όριο στον αριθμό των γλωσσών που μπορούν να χρησιμοποιήσουν οι προγραμματιστές, πράγμα το οποίο δεν είναι επιθυμητό και δεν συνάδει με την λειτουργική απαίτηση σχετικά με την υποστήριξη πολλαπλών γλωσσών προγραμματισμού. Χρησιμοποιώντας λοιπόν το Antlr έχουμε έναν τρόπο να επικυρώσουμε τον κώδικα των serverless συναρτήσεων για πολλαπλές γλώσσες προγραμματισμού.

4.3.3. Κύριες Κλάσεις Υλοποίησης

Η ενσωμάτωση της λύσης με το Flowable επιτυγχάνεται μέσω της χρήσης μιας προσαρμοσμένης κλάσης που επεκτείνει την κλάση **ScriptTaskParseHandler**. Η κλάση αυτή προσαρμόζει τη διαδικασία επεξεργασίας των Script Tasks στο Flowable, επιτρέποντας την ενσωμάτωση πρόσθετης επιχειρησιακής λογικής κατά την εκτέλεση των ροών εργασίας.

Μια από τις πιο σημαντικές κλάσεις είναι η κλάση SLSScriptExecutionListener, η οποία φαίνεται στην Εικόνα 9 και Εικόνα 10. Η κλάση αυτή υλοποιεί το συστατικό Serverless Script Execution Listener, το οποίο είναι υπεύθυνο για τον συντονισμό της φάσης εκτέλεσης. Η κλάση αυτή διαθέτει τρείς μεταβλητές τύπου Expression: το scriptTaskID, που χρησιμοποιείται για την αναζήτηση της αντίστοιχης serverless συνάρτησης στη βάση δεδομένων (αν δεν οριστεί, χρησιμοποιείται το id του τρέχοντος ScriptTask): το provider, που ορίζει τον πάροχο cloud στον οποίο εκτελείται η συνάρτηση, και το local, που υποδηλώνει αν η εκτέλεση γίνεται τοπικά ή στο cloud. Οι τιμές αυτών των πεδίων ορίζονται ως χαρακτηριστικά επέκτασης στο BPMN διάγραμμα. Επιπροσθέτως, η κλάση αυτή υλοποιεί (implements) την διεπαφή ExecutionListener του Flowable, η οποία περιέχει την μέθοδο **notify()** που θα εκτελεστεί για κάθε script task στην ροή εργασίας. Κατά την κλήση της, θα καλέσει μέσω του αρχείου διαμόρφωσης του Serverless Framework, τη serverless συνάρτηση που αντιστοιχεί σε εκείνο το Script Task που έχει μοναδικό αναγνωριστικό το παρεχόμενο **scriptTaskID.** Ύστερα θα αποθηκεύσει το αποτέλεσμα της εκτέλεσης της συνάρτησης στην μεταβλητή αποτελέσματος (result variable) που έχει οριστεί στο script task αυτό, διαφορετικά θα συνεχίσει την εκτέλεση της ροής.

```java
@Component("slsScriptExecutionListener")
public class SLSScriptExecutionListener implements ExecutionListener {
    private static final Logger LOGGER = LoggerFactory.getLogger(SLSScriptExecutionListener.class);

    private Expression local;
    private Expression provider;
    // Script task id in order to execute the associated serverless function
    private Expression scriptTaskID;

    private final SLSPlatformService slsPlatformService;
    private final SLSFlowableIntegrationService slsFlowableIntegrationService;

    public SLSScriptExecutionListener(SLSPlatformService slsPlatformService,
                                      SLSFlowableIntegrationService slsFlowableIntegrationService) {
        this.slsPlatformService = slsPlatformService;
        this.slsFlowableIntegrationService = slsFlowableIntegrationService;
    }
```

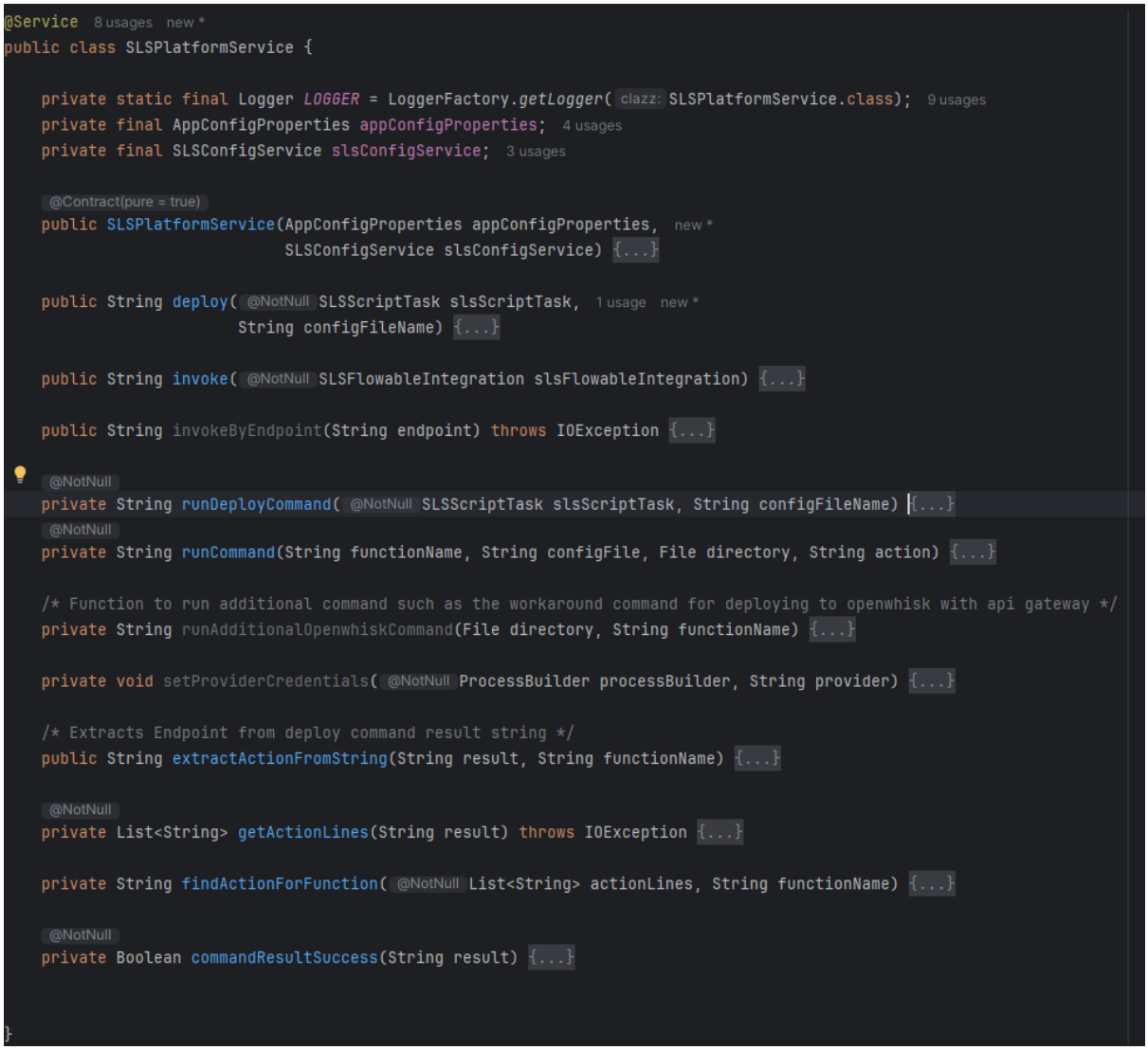
Εικονα 9 - Κλαση SslScriptExecutionListener (Μερος Πρωτο)

```java
@Override  new *
public void notify( @NotNull DelegateExecution delegateExecution) {
    FlowElement currentFlowElement = delegateExecution.getCurrentFlowElement();
    if (!(currentFlowElement instanceof ScriptTask)) {
        LOGGER.error("Current flow element is not a ScriptTask. Cannot proceed with execution.");
        return;
    }

    ScriptTask scriptTask = (ScriptTask) currentFlowElement;
    String scriptTaskId = Optional.ofNullable(scriptTaskID.getValue( variableContainer: delegateExecution)) Optional<Object>
            .orElse( other: scriptTask.getId()) Object
            .toString();

    if (scriptTaskId == null) {
        LOGGER.error("Script Task ID is null. Cannot proceed with execution.");
        return;
    }

    SLSFlowableIntegration slsFlowableIntegration = slsFlowableIntegrationService
            .findByScriptTaskId(scriptTaskId)
            .orElseThrow( exceptionSupplier: () ->
                    new FlowableException("Serverless configuration not found for scriptTaskId: " + scriptTaskId));

    String invokeResult = slsPlatformService.invoke(slsFlowableIntegration);
    LOGGER.info("Invoked cloud function: {} with result: {}", slsFlowableIntegration.getFunctionName(), invokeResult);

    String resultVariable = StringUtils.defaultIfBlank( str: scriptTask.getResultVariable(),  defaultStr: "result");
    delegateExecution.setVariable( variableName: resultVariable,  value: invokeResult);
}
}
```

ΕΙΚΟΝΑ 10 - ΚΛΑΣΗ SLSSCRIPTEXECUTIONLISTENER (ΜΕΡΟΣ ΔΕΥΤΕΡΟ)

Η μέθοδος **notify()** χρησιμοποιεί δύο spring services: την SLSFlowableIntegrationService, που αντιστοιχεί στο συστατικό Serverless WFMS Integration Service, για την ανάκτηση της διαμόρφωσης της serverless συνάρτησης από τη βάση δεδομένων βάσει του scriptTaskId, και την SLSPlatformService, που αντιστοιχείστο συστατικό Serverless Platform Service, για την εκτέλεση (invoke) της συνάρτησης στον αντίστοιχο πάροχο cloud.

Η κλάση SLSPlatformService, η οποία φαίνεται στην Εικόνα 11, είναι υπεύθυνη για να δημιουργήσει ή να θέσει προς εκτέλεση συνάρτηση χωρίς εξυπηρετητή αλλά και να αποσπάσει χρήσιμη πληροφορία (όπως το αποτέλεσμα κλήσης ή το αποτέλεσμα του πακεταρίσματος και διάταξης μιας συνάρτησης) από τις εντολές εκτέλεσης που θα τρέξουν στο Serverless Framework.



ΕΙΚΟΝΑ 11 - ΚΛΑΣΗ SLSPLATFORMSERVICE

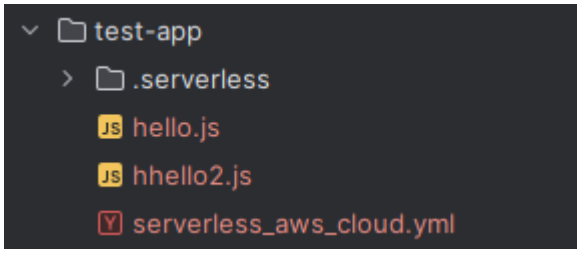
Η κλάση SLSFlowableIntegrationService είναι υπεύθυνη για την αποθήκευση/ανάκτηση της πληροφορίας που χρησιμοποιεί η επέκταση/λύση μας στη βάση δεδομένων. Στην **Error! Reference source not found.** φαίνεται ο πίνακας που χρησιμοποιήθηκε στην βάση δεδομένων για την αποθήκευση της πληροφορίας που προαναφέρθηκε.

Στον πίνακα αυτό διακρίνονται οι παρακάτω πληροφορίες που διατηρούνται από το σύστημά μας:

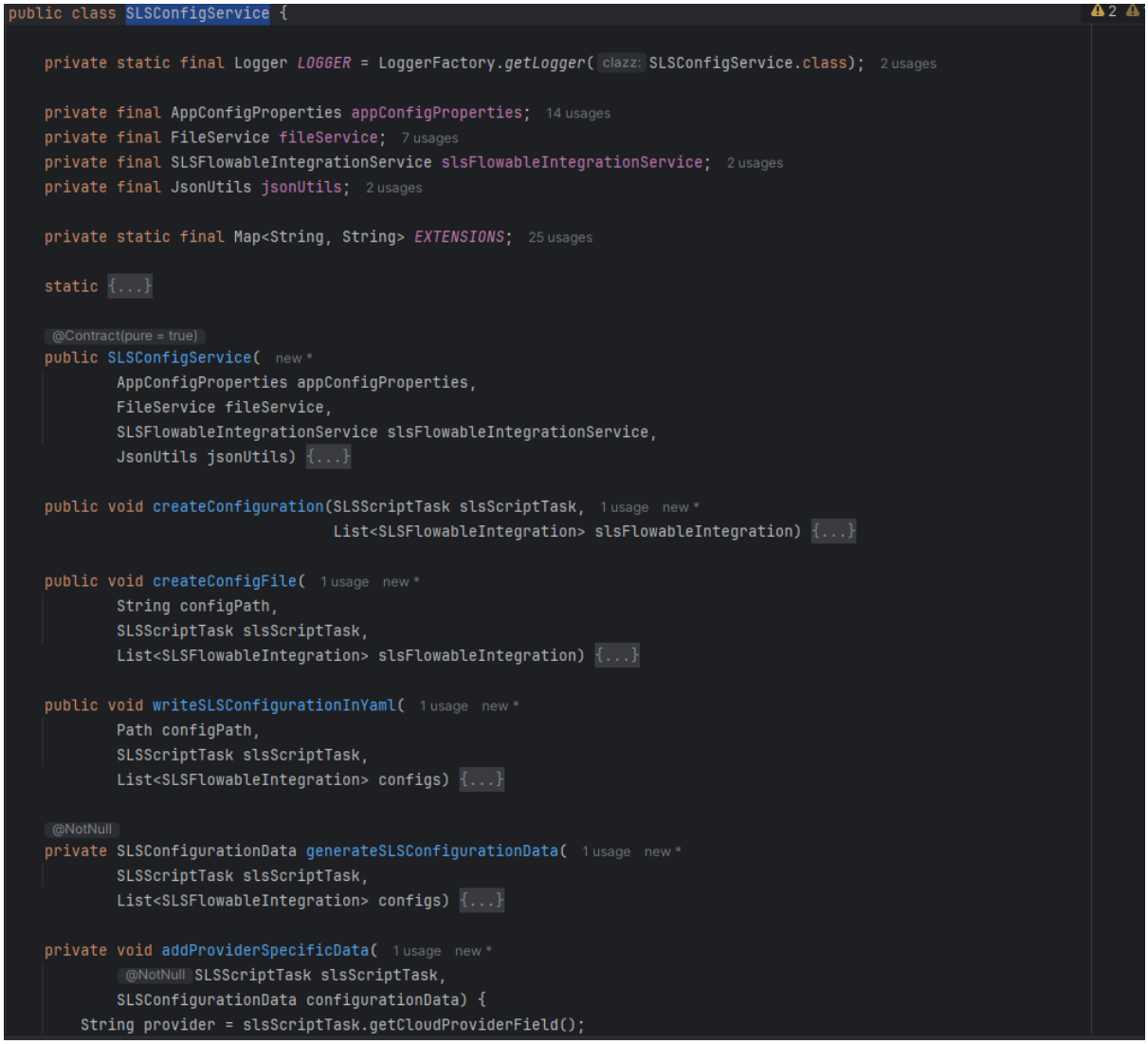
- **Script_task_id:** Αποθηκεύεται το μοναδικό αναγνωριστικό για το Script Task.
- **Provider:** Αποθηκεύεται η πληροφορία του παρόχου στον οποίο φιλοξενείται η serverless συνάρτηση. Είναι απαραίτητη για το αρχείο διαμόρφωσης του Serverless Framework.
- **Fun_name:** Το όνομα της συνάρτησης όπως δηλώθηκε στο Flowable. Αντλείται απο το πεδίο **Name του script task** όπου εντοπίζεται ο ορισμός του κώδικα της συνάρτησης..
- **Scripthash:** Αποθηκεύεται ένας κατακερματισμός (hash) SHA-256 που παράγεται από τον κώδικα της συνάρτησης. Χρησιμοποιείται για να διαπιστωθούν αλλαγές στη δήλωση των συναρτήσεων στις ροές εργασίας του Flowable.
- **Script_name:** Αποθηκεύεται πληροφορία για την ονομασία του αρχείου που περιέχει τον κώδικα.
- **Script_code:** Αποθηκεύεται ο κώδικας ως πληροφορία.

- **Endpoint:** Σε περίπτωση που έχει δηλωθεί στο αρχείο διαμόρφωσης του Serverless Framework ότι η συνάρτηση θα βρίσκεται σε κάποιο τελικό σημείο (endpoint) τότε θα αποθηκευτεί η πληροφορία αυτή (δηλαδή ένα URL) προς χρήση για την εκκίνηση της εκτέλεσης της συνάρτησης.
- **App_def_key:** Αποθηκεύεται το μοναδικό αναγνωριστικό του ορισμού εφαρμογής. Χρησιμοποιείται εσωτερικά στην επέκτασή μας για να διαπιστωθεί ο φάκελος, στον οποίο βρίσκεται μέσα όλη η πληροφορία που χρειάζεται το Serverless Framework (αρχείο διαμόρφωσης και αρχεία με τον κώδικα για κάθε cloud function)
- **Local:** Αποθηκεύεται πληροφορία που δηλώνει αν ο πάροχος βρίσκεται στο εσωτερικό δίκτυο. Χρησιμοποιείται από το αρχείο διαμόρφωσης του Serverless Framework.
- **Id:** Μοναδικό αναγνωριστικό και πρωτεύον κλειδί στον πίνακα.

Επόμενη κλάση που πρέπει να αναφερθεί είναι η SLSConfigService, η οποία είναι υπεύθυνη για την δημιουργία και επεξεργασία των αρχείων διαμόρφωσης αλλά και των αρχείων που περιέχουν τον κώδικα των συναρτήσεων. Η επέκταση χρησιμοποιεί έναν φάκελο με ονομασία ***serverless/config,*** ο οποίος εντοπίζεται εντός του ριζικού φακέλου όπου πραγματοποιείται η εκτέλεση του συστήματος. Η διαδρομή αυτή ορίζεται μέσω της παραμέτρου app.serverlessConfigDir στο αρχείο application.properties**.** Σε αυτόν τον φάκελο δημιουργούνται υποφακέλοι με όνομα το μοναδικό αναγνωριστικό του εκάστοτε ορισμού εφαρμογής. Ένας ορισμός εφαρμογής στο Flowable, όπως προαναφέρθηκε, αποτελεί μια μονάδα που ορίζει ο προγραμματιστής μέσω του Process Modeler, ομαδοποιώντας μία ή περισσότερες ροές εργασίας BPMN κάτω από ένα μοναδικό χαρακτηριστικό. Η επέκταση χρησιμοποιεί αυτό το αναγνωριστικό (appDefKey) για τη κατάλληλη οργάνωση των serverless συναρτήσεων: όλες οι συναρτήσεις που ανήκουν στον ίδιο ορισμό εφαρμογής και τον ίδιο πάροχο cloud ομαδοποιούνται σε ένα κοινό αρχείο διαμόρφωσης. Συνεπώς, ροές εργασίας που ανήκουν στον ίδιο ορισμό εφαρμογής μοιράζονται αυτόματα τις ίδιες serverless συναρτήσεις. Για κάθε ορισμό εφαρμογής δημιουργείται υποφάκελος με όνομα appDefKey εντός του φακέλου διαμόρφωσης. Κάθε τέτοιος υποφάκελος περιέχει μέσα αρχεία διαμόρφωσης για τους ξεχωριστούς παρόχους (υποστηριζόμενες πλατφόρμες) αλλά και αρχεία με τον κώδικα για κάθε συνάρτηση που χρησιμοποιούνται από το Serverless Framework. Στην Εικόνα 12 φαίνεται το αποτέλεσμα της εκτέλεσης της υποβολής του ορισμού εφαρμογής που έγινε προηγουμένως. Να σημειωθεί πως αυτό παράχθηκε με χρήση της κλάσης SLSConfigService.



ΕΙΚΟΝΑ 12 - ΦΑΚΕΛΟΣ ΜΕ ΤΟ ΑΡΧΕΙΑ ΠΟΥ ΧΡΕΙΑΖΕΤΑΙ ΤΟ SERVERLESS FRAMEWORK ΚΑΙ ΣΧΕΤΙΖΟΝΤΑΙ ΜΕ ΤΟΝ ΟΡΙΣΜΟ ΕΦΑΡΜΟΓΗΣ ΜΕ ΑΝΑΓΝΩΡΙΣΤΙΚΟ TEST-APP.



```
public class SLSConfigService {

    private static final Logger LOGGER = LoggerFactory.getLogger( clazz: SLSConfigService.class);  2 usages

    private final AppConfigProperties appConfigProperties;  14 usages
    private final FileService fileService;  7 usages
    private final SLSFlowableIntegrationService slsFlowableIntegrationService;  2 usages
    private final JsonUtils jsonUtils;  2 usages

    private static final Map<String, String> EXTENSIONS;  25 usages

    static {...}

    @Contract(pure = true)
    public SLSConfigService(  new *
            AppConfigProperties appConfigProperties,
            FileService fileService,
            SLSFlowableIntegrationService slsFlowableIntegrationService,
            JsonUtils jsonUtils) {...}

    public void createConfiguration(SLSScriptTask slsScriptTask,  1 usage  new *
                                    List<SLSFlowableIntegration> slsFlowableIntegration) {...}

    public void createConfigFile(  1 usage  new *
            String configPath,
            SLSScriptTask slsScriptTask,
            List<SLSFlowableIntegration> slsFlowableIntegration) {...}

    public void writeSLSConfigurationInYaml(  1 usage  new *
            Path configPath,
            SLSScriptTask slsScriptTask,
            List<SLSFlowableIntegration> configs) {...}

    @NotNull
    private SLSConfigurationData generateSLSConfigurationData(  1 usage  new *
            SLSScriptTask slsScriptTask,
            List<SLSFlowableIntegration> configs) {...}

    private void addProviderSpecificData(  1 usage  new *
            @NotNull SLSScriptTask slsScriptTask,
            SLSConfigurationData configurationData) {
        String provider = slsScriptTask.getCloudProviderField();
```

ΕΙΚΟΝΑ 13 - ΚΛΑΣΗ SLSCONFIGSERVICE (ΜΕΡΟΣ ΠΡΩΤΟ)

```
    public void saveSLSConfigurationToConfigFile( 1 usage  new *
            Path filePath,
            SLSConfigurationData slsConfigurationData) {...}

    private void addFunctionToConfigData( 1 usage  new *
            @NotNull SLSScriptTask slsScriptTask,
            @NotNull SLSConfigurationData slsConfigurationData) {...}

    private Function createFunctionBasedOnProvider( 2 usages  new *
            @NotNull String provider,
            String functionName,
            Boolean isLocalEnvironment) {...}

    public String getFunctionScriptFileName( @NotNull SLSScriptTask scriptTask) {...}

    public String getFunctionScriptFileName( @NotNull ScriptTask scriptTask, String provider) {...}

    public String getScriptFileNameWithExtension( 2 usages  new *
            String functionName,
            String runtime,
            String provider) {...}

    public void addExistingFunctionsToConfigData( 1 usage  new *
            @NotNull List<SLSFlowableIntegration> configs,
            String currentScriptName,
            SLSConfigurationData slsConfigurationData) {...}

    public String getProviderConfigTemplatePath( @NotNull String provider) {...}

    private String getProviderCredentialFilePath( @NotNull String provider) {...}

    public Map<String, String> getProviderCredentials(String provider) {...}

    public SLSConfigurationData getSLSConfigData( @NotNull SLSScriptTask slsScriptTask, Boolean isTemplate) {...}

    public List<String> getAllScriptNamesFromConfigs(List<SLSFlowableIntegration> configs) {...}

    public void createScriptFilesForExistingConfigs( @NotNull List<SLSFlowableIntegration> configs) {...}

    public Boolean createScriptFile(Path scriptFilePath) { return fileService.createFile(scriptFilePath); }
```

ΕΙΚΟΝΑ 14 - ΚΛΑΣΗ SLSCONFIGSERVICE (ΜΕΡΟΣ ΔΕΥΤΕΡΟ)

```
    public void removeConfigScriptFiles( @NotNull SLSScriptTask slsScriptTask) {...}

    public String getSLSConfigFilePath( @NotNull SLSScriptTask slsScriptTask) {...}

    public String getSLSConfigFileName(String provider, Boolean isLocal) {...}

    public void writeDataInScriptFile(Path scriptFilePath, String data, Boolean append) {...}

    public SLSConfigurationData removeFunctionFromConfig(SLSScriptTask slsScriptTask) {...}
}
```

ΕΙΚΟΝΑ 15 - ΚΛΑΣΗ SLSCONFIGSERVICE (ΜΕΡΟΣ ΤΡΙΤΟ)

Η προσαρμοσμένη κλάση **ScriptTaskParseHandler** επεμβαίνει στη διαδικασία ανάλυσης των Script Tasks, επιτρέποντας την εισαγωγή προσαρμοσμένης επιχειρησιακής λογικής. Αυτό περιλαμβάνει τη δημιουργία και τη διαχείριση κώδικα που προορίζεται για εκτέλεση σε περιβάλλοντα νέφους (serverless πλατφόρμες) μέσω του Serverless Framework.

Η ScriptTaskParseService, η οποία είναι υπεύθυνη την επεξεργασία (parse) όλων των Script Task κατά την διαδικασία υποβολής /δημοσίευσης μιας ροής εργασίας που έχει οριστεί. Η κλαση αυτή κληρονομεί από την κλάση ScriptTaskParseHandler του Flowable και υλοποιεί την μέθοδο parse() στην οποία εφαρμόζεται ο προσαρμοσμένος κώδικας. Στην Εικόνα 16 παρουσιάζονται οι μέθοδοι που υλοποιεί η κλάση ScriptTaskParseService.

Στη μέθοδο parse(), η προσαρμοσμένη κλάση:

- Επικυρώνει απαραίτητες πληροφορίες που θα χρειαστούν σε επόμενα βήματα. Αυτές οι πληροφορίες αφορούν τον SLSScriptExecutionListener (γίνεται έλεγχος αν υπάρχει) και τις μεταβλητές του.
- Ελέγχει αν το script task έχει τις ελάχιστες απαραίτητες πληροφορίες, όπως το μοναδικό αναγνωριστικό, τον κώδικα της συνάρτησης κ.α.
- Ελέγχει για συντακτικά λάθη στον κώδικα του script task χρησιμοποιώντας το Antlr4.
- Δημιουργεί/Ενημερώνει τα αρχεία με τον κώδικα των συναρτήσεων και τα αρχεία διαμόρφωσης (yaml) που θα χρησιμοποιηθούν απο το Serverless Framework.
- Χρησιμοποιεί το Serverless Framework για την δημιουργία/τροποποίηση των serverless συναρτήσεων.
- Αποθηκεύει απαραίτητες πληροφορίες σε εγγραφή στην βάση δεδομένων.

```
@Override  new *
public void parse(BpmnParse bpmnParse, BaseElement baseElement) {...}

private void validateAndProcessScriptTask( @NotNull BpmnParse bpmnParse,  @NotNull ScriptTask scriptTask) {...}

private void validateFlowableScriptTaskProperties(ScriptTask scriptTask) {...}

private void validateFlowableScriptSyntax( @NotNull ScriptTask scriptTask) {...}

private void processSLSScriptTask(SLSScriptTask scriptTask) { handleConfigAndDeployment(scriptTask); }

private void validateSLSScriptTaskFields( @NotNull SLSScriptTask slsScriptTask) {...}

@Contract(value = "null -> true", pure = true)
private boolean isProviderFieldEmpty(String providerField) {...}

@Contract(value = "null -> true", pure = true)
private boolean isScriptTaskCodeEmpty(String scriptTaskCode) {...}

private void handleConfigAndDeployment(SLSScriptTask slsScriptTask) {...}

private void cleanupFunctionScriptFiles(SLSScriptTask slsScriptTask) {...}

public Boolean handleConfiguration( @NotNull SLSScriptTask slsScriptTask) {...}

public String handleDeployment( @NotNull SLSScriptTask slsScriptTask) {...}

@NotNull
private Optional<String> extractFunctionEndpoint(String deployResultMsg, String scriptName) {...}
```

ΕΙΚΟΝΑ 16 - ΜΕΘΟΔΟΙ ΠΟΥ ΠΕΡΙΕΧΕΙ Η ΚΛΑΣΗ SCRIPTTASKPARSESERVICE.

Αξίζει να δούμε την μέθοδο ***validateAndProcessScriptTask(),*** η οποία περιέχει τα βήματα που ακολουθούνται (προς επικύρωση και επεξεργασία script tasks) καθώς και σχετική περιγραφή, όπως

φαίνεται στην Εικόνα 17. Διακρίνεται πως για συνεχίσει η διαδικασία του πακεταρίσματος και της διάταξης πρέπει πρώτα γίνουν οι εξής έλεγχοι:

- Έλεγχος αν το Script Task διαθέτει ExecutionListener του τύπου SLSScriptExecutionListener. Ο έλεγχος αυτός αποτελεί τον βασικό μηχανισμό διάκρισης μεταξύ ενός κανονικού Script Task και ενός Script Task που προορίζεται για serverless συνάρτηση. Κάθε script task που επιθυμεί ο προγραμματιστής να συνδέσει με serverless συνάρτηση οφείλει να ορίζει τον SLSScriptExecutionListener ως ExecutionListener στο BPMN διάγραμμα. Αν δεν υπάρχει, το task αντιμετωπίζεται ως κανονικό task του Flowable.
- Έλεγχος αν ο κώδικας του Script Task έχει συντακτικά λάθοι. Χρησιμοποιείται το εργαλείο Antlr για την διαπίστωση αυτή.
- Έλεγχος αν υπάρχουν τα απαραίτητα πεδία στον ExecutionListener που διαπιστώθηκε ότι υπάρχει στον πρώτο έλεγχο.

Κατόπιν, η μέθοδος αυτή συνεχίζει στην διαδικασία του πακεταρίσματος και της διάταξης. Κατά την διάρκεια κάθε βήματος καταγράφεται χρήσιμη πληροφορία στα αρχεία καταγραφής έτσι ώστε να είναι κατανοητό στον προγραμματιστή ροών εργασίας σε τι κατάσταση βρίσκεται αφότου έγινε η υποβολή.

```
private void validateAndProcessScriptTask( @NotNull BpmnParse bpmnParse,  @NotNull ScriptTask scriptTask) {  1 usage  new *
    String scriptTaskId = scriptTask.getId();

    // Convert Flowable Script Task object to SLSScriptTask
    // to host only relevant data across the deployment
    SLSScriptTask slsScriptTask = slsScriptTaskMapper
            .convert(scriptTask,  appDefKey: bpmnParse.getDeployment().getKey())
            .orElse( other: null);

    // Case where script task should be handled by flowable
    if (slsScriptTask == null) {
        LOGGER.warn("Could not convert script task to serverless script task");
        return;
    }

    // Set script code to empty so that flowable enigne can continue with the deployment of the app.
    // This is essential cause flowable engine will validate against its own script formating language
    // and the engine will terminate the deployment if the language does not match which is definitely
    // the case with this plugin
    scriptTask.setScriptFormat("juel");
    scriptTask.setScript(NO_OP_SCRIPT);

    // Catch case where script task references another script tasks id
    if (!Objects.equals( a: scriptTaskId,  b: slsScriptTask.getScriptTaskId())) {
        LOGGER.warn("Found script task that will run a cloud function based on another script task id: {}", scriptTaskId);
        return;
    }

    // Check for syntax errors using Antlr4
    validateFlowableScriptSyntax(scriptTask);

    // Ensure script task has valid fields before processing
    validateSLSScriptTaskFields(slsScriptTask);

    processSLSScriptTask(slsScriptTask);
}
```

ΕΙΚΟΝΑ 17 - ΜΕΘΟΔΟΣ VALIDATEANDPROCESSSCRIPTTASK ΤΗΣ ΚΛΑΣΗΣ SCRIPTTASKPARSESERVICE.

Πέρα από τις κλάσεις που περιγράφηκαν παραπάνω, η λύση περιλαμβάνει και δύο βοηθητικές κλάσεις που υποστηρίζουν την επεξεργασία των script tasks. Η κλάση FlowableUtils συγκεντρώνει ένα σύνολο μεθόδων για την εξαγωγή πληροφορίας από το BPMN μοντέλο ενός script task: ελέγχει

αν ένα script task φέρει τον SLSScriptExecutionListener (και επομένως αν αντιστοιχεί σε serverless συνάρτηση), εντοπίζει τον σχετικό execution listener και ανακτά τις τιμές των πεδίων του, όπως τον πάροχο, την ένδειξη τοπικής εκτελεσης και το αναγνωριστικό scriptTaskID. Λειτουργεί έτσι ως κοινό σημείο πρόσβασης στα στοιχεία του listener για τις υπόλοιπες κλάσεις.

Η κλάση SLSScriptTaskMapper με τη σειρά της αναλαμβάνει τη μετατροπή ενός Flowable ScriptTask στο εσωτερικό αντικείμενο SLSScriptTask, συγκεντρώνοντας σε ένα ενιαίο αντικείμενο όλη την πληροφορία της serverless συνάρτησης (συμπεριλαμβανομένου του ονόματος του αρχείου συνάρτησης) ώστε να χρησιμοποιηθεί στα επόμενα στάδια της διαδικασίας.

4.4. Δοκιμή

Για την διασφάλιση της ορθής λειτουργίας της εφαρμογής πραγματοποιήθηκαν δοκιμές μονάδας (**unit testing**) και ολοκλήρωσης (integration testing). Οι δοκιμές μονάδας επικεντρώθηκαν κυρίως στον έλεγχο της επιχειρησιακής λογικής (**business logic**) της εφαρμογής που υλοποιείται με Spring Boot, χρησιμοποιώντας εργαλεία όπως το Junit και το Mockito για την επαλήθευση της σωστής συμπεριφοράς μεμονωμένων μεθόδων και κλάσεων. Συνολικά υλοποιήθηκαν 38 δοκιμές κατανεμημένες σε 6 κλάσεις, καλύπτοντας τις ScriptTaskParseService, SLSConfigService, SLSFlowableIntegrationService, SLSPlatformService, FlowableUtils και SLSScriptTaskMapper. Οι δοκιμές κάλυψαν τις βασικές λειτουργίες κάθε κλάσης, δηλαδή την ανάλυση και επικύρωση των script tasks, τη δημιουργία και διαμόρφωση των αρχείων του Serverless Framework, την εξαγωγή πληροφορίας από το BPMN μοντέλο και την αντιστοίχιση της στα εσωτερικά αντικείμενα την λύσης. Επειδή αρκετές από τις κλάσεις αυτές συνεργάζονται με άλλες για την ολοκλήρωση της λειτουργικότητας τους χρησιμοποιήθηκε το πλαίσιο Mockito για την απομόνωση των εξαρτήσεων, ώστε κάθε δοκιμή να ελέγχει αποκλειστικά τη λογική της εκάστοτε κλάσης.

Οι δοκιμές ολοκλήρωσης στόχευσαν στις βασικές λειτουργίες του συστήματος από άκρο σε άκρο και πραγματοποιήθηκαν χειρωνακτικά. Καλύφθηκαν δύο κρίσιμα σενάρια:

A. Η δημιουργία και υποβολή μιας ροής εργασίας με serverless script tasks
B. Η εκτέλεση μιας τέτοιας ροής

Το πρώτο σενάριο πραγματοποιήθηκε μέσω της γραφικής διεπαφής του Flowable Modeler, όπου ορίστηκε μια εφαρμογή (app definition) και μια ροή εργασίας αποτελούμενη από τρία script tasks, καθένα συνδεδεμένο με μια serverless συνάρτηση μέσω του SLSScriptExecutionListener. Με τη δημοσίευση (publish) του ορισμού εφαρμογής ενεργοποιήθηκε αυτόματα η ανάλυση των script tasks, η δημιουργία των αρχείων διαμόρφωσης και το πακετάρισμα/διάταξη των συναρτήσεων μέσω του Serverless Framework. Η επιτυχία της διάταξης επαληθεύτηκε αφενός από τα logs της εφαρμογής και αφετέρου από τις κονσόλες των παρόχων νέφους, όπου επιβεβαιώθηκε η ύπαρξη των αντίστοιχων συναρτήσεων.

Το δεύτερο σενάριο αφορούσε την εκτέλεση της ροής εργασίας. Το στιγμιότυπο της διαδικασίας εκκινήθηκε μέσω κλήσης του REST API του Flowable με χρήση του Postman και στην συνέχεια παρακολουθήθηκε η διαδοχική εκτέλεση των τριών script tasks με τα αποτελέσματα κάθε serverless συνάρτησης να επιβεβαιώνονται από τα logs της εφαρμογής όσο και από τις μεταβλητές της διαδικασίας, στις οποίες ο execution listener αποθηκεύει το αποτέλεσμα κάθε κλήσης (στη μεταβλητή που ορίζεται στο πεδίο resultVariable του script task ή στην προεπιλεγμένη μεταβλητή result). Επιβεβαιώθηκε επίσης η ορθή επαναχρησιμοποίηση της ίδιας serverless συνάρτησης από δύο διαφορετικά βήματα της ροής. Τα αναλυτικά αποτελέσματα και τα σχετικά στιγμιότυπα οθόνης των παραπάνω δοκιμών παρουσιάζονται στο επόμενο κεφάλαιο.

4.5. Βαθμός Ικανοποίησης Απαιτήσεων

Σε αυτό το κεφάλαιο παρουσιάζεται ο Πίνακας 3 που επιδεικνύει τον βαθμό ικανοποίησης της κάθε απαίτησης όπως αυτές αναλύθηκαν στην Ενότητα 4.1

ΠΙΝΑΚΑΣ 3 - ΒΑΘΜΟΣ ΙΚΑΝΟΠΟΙΗΣΗΣ ΑΠΑΙΤΗΣΕΩΝ

Αριθμός	**Περιγραφή Απαίτησης**	**Βαθμός Ικανοποίησης**	**Σχόλια**
Λειτουργικές Απαιτήσεις			
1Α	Δυνατότητα συμπερίληψης κώδικα συναρτήσεων χωρίς διακομιστές σε tasks μέσα σε ροές εργασίας.	Πλήρως	Η δυνατότητα υλοποιήθηκε πλήρως για script tasks, ενώ υπάρχει περιθώριο για μελλοντική επέκταση σε service tasks.
1Β	Δυνατότητα αντιστοίχισης κάποιας ήδη υπάρχουσας **serverless συνάρτησης** σε μια αυτοματοποιημένη λειτουργία (script task) μέσα στην ροή εργασίας (workflow). Πιο συγκεκριμένα αυτή η αντιστοίχιση ορίζεται κατά την διάρκεια της δημιουργίας ενός script task μέσα στην ροή εργασίας.	Πλήρως	Η υποστήριξη της απαίτησης αυτής επιτεύχθηκε μέσω του πεδίου scriptTaskID του SLSScriptEcecutionLestener, το οποίο ορίζεται ως field extension κατά τη δημιουργία του script task και παραπέμπει στο αναγνωριστικό της ήδη υπάρχουσας function στη βάση δεδομένων.
1Γ	Δυνατότητα υποστήριξης πολλαπλών γλωσσών προγραμματισμού για την επικύρωση του κώδικα της κάθε **serverless συνάρτησης με την χρήση του ANtlr4.**	Μερικώς	Χρειάζεται περαιτέρω επέκταση εργασία για την πλήρη αυτοματοποίηση της επικύρωσης σε πολλαπλές γλώσσες πέραν από τις Javascript, Python 3, Java 9.
1Δ	Δυνατότητα ρύθμισης για τοπική ή απομακρυσμένη εκτέλεση.	Πλήρως	Η δυνατότητα καλύπτεται μέσω των ρυθμίσεων των script tasks αλλά και του αρχείου διαμόρφωσης.
1Ε	Δυνατότητα πακεταρίσματος και διάταξης όλων των serverless συναρτήσεων σε μια ροή εργασίας.	Πλήρως	Υποστηρίζεται για όλους τους παρόχους FaaS που υποστηρίζει το Serverless Framework.
1Ζ	Το σύστημα πρέπει να εντοπίζει πότε ο	Πλήρως	Το σύστημα εντοπίζει όλες τις αλλαγές και

	κώδικας μιας serverless συνάρτησης έχει τροποποιηθεί και να αναδιατάσσει μόνο αυτή τη συνάρτηση.		αναδιατάσει μαζικά τις συναρτήσεις στο πλαίσιο της (τροποποιημένης) ροής εργασίας που τις ορίζει.
1Η	Το σύστημα θα πρέπει να χειρίζεται πολλαπλές **serverless συναρτήσεις** βάσει διαφορετικών αρχείων διαμόρφωσης του **Serverless Framework** και να τις διατάσσει με ελάχιστη χειροκίνητη παρέμβαση.	Μερικώς	Ένα από τα μειονεκτήματα της τρέχουσας υλοποίησης είναι πως η παρέμβαση για πιο πολύπλοκα σενάρια χρήσης των serverless συναρτήσεων είναι απαραίτητη εκτός αν καλύπτεται από τα template αρχεία διαμόρφωσης.
1Θ	Το σύστημα θα πρέπει να αποθηκεύει την πληροφορία της **serverless συνάρτησης** μαζί με τα απαραίτητα δεδομένα από τη ροή εργασίας για την υποστήριξη λειτουργιών αυτοματοποίησης.	Πλήρως	Το σύστημα αποθηκεύει τις πληροφορίες αυτές σε πίνακα στη βάση δεδομένων.
2Ι	Αλληλεπίδραση με το Serverless Framework μέσω μεταβλητών περιβάλλοντος.	Πλήρως	Εφαρμόζεται επιτυχώς με μεταβλητές περιβάλλοντος. Δυνατότητα δυναμικής ρύθμισης από τον προγραμματιστή ροών εργασίας με την προϋπόθεση ότι έχει γνώση του Serverless Framework.
Μη Λειτουργικές Απαιτήσεις			
2Α	Εκτέλεση serverless συνάρτησης σε εύλογο χρονικό διάστημα.	Μερικώς	Η εκτέλεση των συναρτήσεων ικανοποιεί το χρονικό πλαίσιο (αντίστοιχους περιορισμούς) που έχει τεθεί. Αλλά αυτό ελέγχθηκε με περιορισμένο τρόπο και απαιτεί την εκτέλεση μη λειτουργικών δοκιμών σε ρεαλιστικά σενάρια/περιβάλλοντα.

2Β	Πακετάρισμα και διάταξη σε εύλογο χρονικό διάστημα.	Μερικώς	Ο χρόνος είναι ικανοποιητικός για τις περισσότερες περιπτώσεις. Αυτό όμως θα πρέπει να αποδειχθεί και μέσω μη λειτουργικών δοκιμών.
2Δ	Παροχή πληροφοριών για σφάλματα κατά την υποβολή και εκτέλεση.	Μερικώς	Υποστηρίζεται μέσω logs της εφαρμογής. Μπορεί να επεκταθεί για να εμφανίζει ειδοποιήσεις με το ήδη υπάρχον WFMS σύστημα.
2Ε	Εύκολη εισαγωγή/τροποποίηση πληροφορίας serverless συνάρτησης, ρυθμίσεων εκτέλεσης, και ασφαλής ρύθμιση στοιχείων λογαριασμού.	Πλήρως	Παρέχεται μέσω της διεπαφής διαχείρισης ροών και προσαρμοσμένου κώδικα.

5. Εγκατάσταση και Επίδειξη Εφαρμογής

Στο παρόν κεφάλαιο περιγράφεται η διαδικασία εγκατάστασης και επίδειξης της εφαρμογής. Στην Ενότητα **Error! Reference source not found.** παρουσιάζονται τα απαραίτητα βήματα για την εγκατάσταση του συστήματος, ενώ στην Ενότητα 5.2 παρουσιάζεται η επίδειξη της προτεινόμενης λύσης: αρχικά η χρήση και η ρύθμιση του Serverless Framework, στη συνέχεια η διαμόρφωση ενός script task ώστε να φέρει serverless συνάρτηση (τα πεδία που πρέπει να συμπληρώσει ο χρήστης και ο σχετικός Execution Listener) και τέλος ένα ενδεικτικό σενάριο χρήσης που επιδεικνύει την εκτέλεση ροής εργασίας με serverless script tasks.

5.1. Εγκατάσταση Συστήματος

Για την εγκατάσταση της εφαρμογής πρέπει να ακολουθηθούν τα εξής βήματα:

1. Εγκατάσταση **Node.js** ακολουθόντας το URL https://nodejs.org/en/download/ (v.20.17.0 είναι η έκδοση που χρησιμοποιήθηκε κατά την διαδικασία της υλοποίησης της λύσης).
2. Αφού εγκατασταθεί το Node.js στην συνέχεια πρέπει να γίνει χρήση του **npm** (Node Package Manager) για να εγκατασταθεί το πακέτο serverless με την εντολή ***npm i serverless@3.37.0 -g***.
3. Για **Java (v18.0.1.1)** πρέπει να γίνει εγκατάσταση JDK (v18.0.1.1) (σημειώνεται πως απαιτείται JDK και όχι απλώς JRE). Δείτε σχετικές οδηγίες εδώ: https://www.oracle.com/java/technologies/javase/jdk18-archive-downloads.html.
4. Εγκατάσταση της PostgreSQL (16.4 ή μεταγενέστερη) - οδηγίες εγκατάστασης: https://www.postgresql.org/download/. Μετά την εγκατάσταση πρέπει να δημιουργηθεί βάση δεδομένων με όνομα flowable χρησιμοποιώντας τον default χρήστη postgres. Τα credentials σύνδεσης ορίζονται στο αρχείο serverless-flowable-integration/src/main/resources/application.properties:
   spring.datasource.username=postgres
   spring.datasource.password=postgres
   Αν χρησιμοποιείται διαφορετικός χρήστης ή password πρέπει να ενημερωθούν οι τιμές αυτές αντίστοιχα.
5. Πρέπει να γίνει λήψη του κώδικα που βρίσκεται στην διεύθυνση εδώ: https://github.com/JasonGaliats/FlowableServerlessPlugin

Όσον αφορά την εκτέλεση της εφαρμογής, στον (ριζικό) φάκελο που φιλοξενεί τον κώδικα πρέπει να εκτελεστεί η εντολή ***./mvnw spring-boot:run.*** Αυτό θα οδηγήσει στην εκκίνηση της επεκταμένης web εφαρμογής του Flowable (Flowable Modeler) στη διεύθυνση: ***http://localhost:8080/#/processes***.

5.2. Επίδειξη Συστήματος

5.2.1. Χρήση του Serverless Framework

Στο πλαίσιο της ενσωμάτωσης της λύσης χρειάζεται να ακολουθηθούν μερικά βήματα απαραίτητα για την ορθή λειτουργία της αλλά και του Serverless Framework. Οι serverless πλατφόρμες που υποστηρίζονται για την ώρα είναι οι AWS Lambda, Google Functions & OpenWhisk. Αυτά τα βήματα είναι τα εξής:

- Ο προγραμματιστής της ροής εργασίας έχει την δυνατότητα να δημιουργήσει τρείς μεταβλητές περιβάλλοντος με την ονομασία **GOOGLE_CREDENTIALS, OPENWHISK_CREDENTIALS** και **AWS_CREDENTIALS.** Η κάθε μεταβλητή είναι απαραίτητη μόνο αν χρησιμοποιείται ο αντίστοιχος cloud πάροχος. Αν δεν οριστεί, η εφαρμογή εκκινεί κανονικά αλλά η ανάπτυξη σε εκείνον τον cloud πάροχο θα αποτύχει.

- Ο προγραμματιστής της ροής εργασίας έχει την δυνατότητα να δημιουργήσει τρείς μεταβλητές περιβάλλοντος με την ονομασία **GOOGLE_TEMPLATE_PATH, OPENWHISK_TEMPLATE_PATH και AWS_TEMPLATE_PATH.** Οι μεταβλητές αυτές είναι απαραίτητες και περιέχουν πληροφορία σχετικά με την τοποθεσία των template αρχείων διαμόρφωσης που θα χρησιμοποιηθούν στην επέκταση του Flowable με σκοπό την δημιουργία/ενημέρωση του αρχείου διαμόρφωσης (μιας συνάρτησης) με τις απαραίτητες πληροφορίες προς χρήση από το Serverless Framework. Ο προγραμματιστής της ροής εργασίας έχει την δυνατότητα να δημιουργήσει και δικά του αρχεία διαμόρφωσης και να συνδέσει τις μεταβλητές με αυτά (οπότε θα εφαρμόζονται πλέον τα δικά του). Αυτά τα αρχεία περιέχουν διαφορετικές πληροφορίες ανάλογα με την serverless πλατφόρμα που εξυπηρετούν οπότε και κάποιες από τις πληροφορίες θα πρέπει να συμπληρωθούν κατά την διάρκεια ανάπτυξης της ροής εργασίας. Ο λόγος που υποστηρίζεται αυτό είναι επειδή η επέκταση που υλοποιήθηκε δεν μπορεί να προσφέρει τις μέγιστες δυνατότητες του Serverless Framework διότι για να επιτευχθεί αυτό πρέπει να γίνει μια πλήρης 1-1 αντιστοιχία με όλες τις υποστηριζόμενες μεταβλητές του αρχείου διαμόρφωσης από την επέκταση του Flowable. Συνεπώς, αυτό θα μπορούσε να ενταχθεί ως δυνατότητα σε μελλοντική εργασία.

Στην συνέχεια παρουσιάζονται τρία από αυτά τα αρχεία διαμόρφωσης για τους προαναφερθέντες πάροχους. Στην Εικόνα 18 φαίνεται το αρχείο διαμόρφωσης για την AWS Lambda πλατφόρμα. Οι πληροφορίες που επηρεάζονται κατά την διάρκεια της εκτέλεσης για κάθε πλατφόρμα είναι οι εξής:

- **Functions:** Εδώ συμπληρώνονται αυτόματα από την λύση όλες οι πληροφορίες σχετικά με τις serverless συναρτήσεις.
- **Provider:** Εδώ συμπληρώνονται αυτόματα από την λύση όλες οι πληροφορίες σχετικά τον πάροχο και την γλώσσα προγραμματισμού των cloud functions.

Για τις υπόλοιπες πληροφορίες είναι ευθύνη του προγραμματιστή της ροής εργασίας να τις τροποποιήσει/ελέγξει έτσι ώστε να μην υπάρχει κάποιο πρόβλημα κατά την κατά την διαδικασία υποβολής της serverless function. Σε περίπτωση που υπάρχει κάποιο πρόβλημα σχετικά με το αρχείο διαμόρφωσης, αυτό θα φανεί κατά την διαδικασία υποβολής ορισμού εφαρμογής στο Flowable μέσω μηνύματος αποτυχίας. Επιπρόσθετα, στα logs της εφαρμογής θα φανούν οι πληροφορίες σχετικά με την αποτυχία. Ο διαχωρισμός σε τρία αρχεία διαμόρφωσης έγινε για λόγους απλοποίησης διότι το Serverless Framework απαιτεί διαφορετικές πληροφορίες στα αρχεία αυτά ανάλογα την πλατφόρμα.

Να σημειωθεί πως στην Εικόνα 19 και στην Εικόνα 20 υπάρχει πεδίο που ορίζει το plugin[16] που θα χρησιμοποιηθεί για τις διαφορεκτικές πλατφόρμες (Google, OpenWhisk) σε σχέση με την AWS Lambda.

[16] Το Serverless Framework χρησιμοποιεί plugins για την υποστήριξη παρόχων πέραν του AWS, ο οποίος είναι προεπιλεγμένος. Για Google Cloud Functions και OpenWhisk απαιτείται η δήλωση του αντίστοιχου plugin (serverless-google-cloudfunctions, serverless-openwhisk) στο αρχείο διαμόρφωσης.

```
org: galiats
app: test
service: test

frameworkVersion: '3'

provider:
  name: '[aws]'
  runtime:
functions:
# A function
```

ΕΙΚΟΝΑ 18 - TEMPLATE ΑΡΧΕΙΟ ΔΙΑΜΟΡΦΩΣΗΣ ΓΙΑ ΤΗΝ ΠΛΑΤΦΟΡΜΑ AWS LAMBDA

```
service: test

provider:
  name: google
  runtime:
  project: aqueous-choir-366722
  credentials:
  region: us-central1
functions:
# A function

plugins:
  - serverless-google-cloudfunctions

configValidationMode: off
```

ΕΙΚΟΝΑ 19 - TEMPLATE ΑΡΧΕΙΟ ΔΙΑΜΟΡΦΩΣΗΣ ΓΙΑ ΤΟΝ CLOUD ΠΑΡΟΧΟ GOOGLE.

```yaml
service: testtt

provider:
  name: openwhisk
  runtime:
  ignore_certs: true
functions:
# A function

plugins:
  - serverless-openwhisk

configValidationMode: off
```

ΕΙΚΟΝΑ 20 - TEMPLATE ΑΡΧΕΙΟ ΔΙΑΜΟΡΦΩΣΗΣ ΓΙΑ ΤΟΝ CLOUD ΠΑΡΟΧΟ OPENWHISK.

Να σημειωθεί πως τα αρχεία διαμόρφωσης μπορούν να γίνουν αρκετά πολύπλοκα ανάλογα με το σενάριο χρήσης. Για παράδειγμα, μπορεί μια serverless συνάρτηση να χρησιμοποιεί δεδομένα από μια απομακρυσμένη βάση δεδομένων (όπως DynamoDB στην περίπτωση του νέφους της AWS) οπότε σε αυτή την περίπτωση πρέπει να δηλωθούν οι απαραίτητες πληροφορίες στο αρχείο διαμόρφωσης. Καθώς η παρούσα λύση δεν καλύπτει το σύνολο των δυνατότήτων που προσφέρει το serverless framework, παρέχεται η δυνατότητα στον προγραμματιστή των ροών εργασίας να ορίσει το δικό του αρχείο διαμόρφωσης μέσω αντίστοιχης μεταβλητή περιβάλλοντος όπως αναφέρθηκε προηγουμένως. Αυτό αποτελεί σημαντικό χαρακτηριστικό επεκτασιμότητας καθώς επιτρέπει την υποστήριξη πολύπλοκων σεναρίων χωρίς να απαιτείται τροποποίηση του πυρήνα της εφαρμογής

Τέλος επειδή τα αρχεία διαμόρφωσης πρέπει να είναι φυσικά παρόντα στο σύστημα ώστε το Serverless Framework να μπορεί να λειτουργήσει, η λύση διατηρεί ένα ειδικό φάκελο υπό τον οποίο αποθηκεύονται τα αρχεία διαμόρφωσης οργανωμένα ανά ορισμό εφαρμογής του Flowable.

5.2.2. Δημιουργία Ροής Εργασίας με Serverless Script Tasks

Η διαδικασία δημιουργίας μιας ροής εργασίας περιλαμβάνει μια διαδικασία ενσωμάτωσης για κάθε σχετικό Script Task εντός του ορισμού της ροής. Ως διαδικασία ενσωμάτωσης νοείται το σύνολο των ενεργειών που απαιτούνται για την σύνδεση μιας serverless συνάρτησης με ένα Script Task μιας ροής εργασίας στο Flowable. Περιλαμβάνει τη ρύθμιση του Script Task στο BPMN μοντέλο ώστε να φέρει τον κατάλληλο Execution Listener και τον καθορισμό των παραμέτρων του παρόχου και περιβάλλοντος.

Ένα Script Task στο flowable ορίζεται απο το σύνολο των παρακάτω πεδίων:

Όνομα (Name): Περιγραφικός τίτλος του task εντός του μοντέλου. Βοηθά στον εντοπισμό του task στον ορισμό της διαδικασίας και στα logs εκτέλεσης.

Κωδικός (Id): Μοναδικό αναγνωριστικό του Script Task μέσα στο μοντέλο διαδικασίας. Χρησιμοποιείται προγραμματιστικά για αναφορά στο task κατα την εκτέλεση ή κατά την ελληλεπίδραση με τη ροή εργασιών.

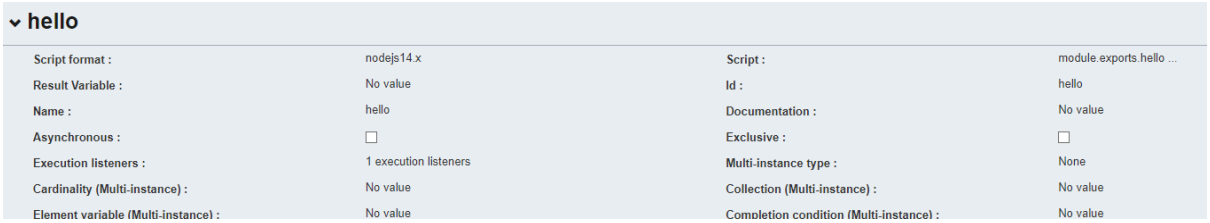
Μορφή Script (Script Format): Προσδιορίζει τη γλώσσα προγραμματισμού του script (π.χ. Javascript, Python, Java). Καθορίζει τποια μηχανή εκτέλεσης (scripting engine) θα χρησιμοποιήσει το Flowable για την ερμηνεία και εκτέλεση του script.

Script: Το ίδιο το script που εκτελείται απο το task. Περιέχει τον κώδικα της λογικής ή των ενεργειών που θα εκτελεστούν στο συγκεκριμένο σημείο της ροής εργασιών.

Όνομα Μεταβλητής Αποτελέσματος (Result variable Name): Το όνομα της μεταβλητής όπου αποθηκεύεται το αποτέλεσμα εκτέλεσης του script, επιτρέποντας την πρόσβαση σε αυτό απο επόμενα tasks της διαδικασίας.

Execution Listeners: Ειδικοί ακροατές που συνδέονται με το task και επιτρέπουν την εκτέλεση πρόσθετης λογικής σε συγκεκριμένα σημεία της εκτέλεσης. Ενεργοποιούνται βάσει γεγονότων: Start Event (έναρξη του task), End Event (ολοκλήρωση του task), ή Take Event (μετάβαση). Μπορούν να τρέξουν προσαρμοσμένο Java κώδικα, να τροποποιήσουν μεταβλητές (τοπικές ή καθολικές) και να υλοποιήσουν λογική καταγραφής ή διαχείρισης σφαλμάτων.

Τα πεδία αυτά αποτυπώνονται στην Εικόνα 21, που παρουσιάζει τις ρυθμίσεις του script task hello (πρώτου task στην ορισμένη ροή).



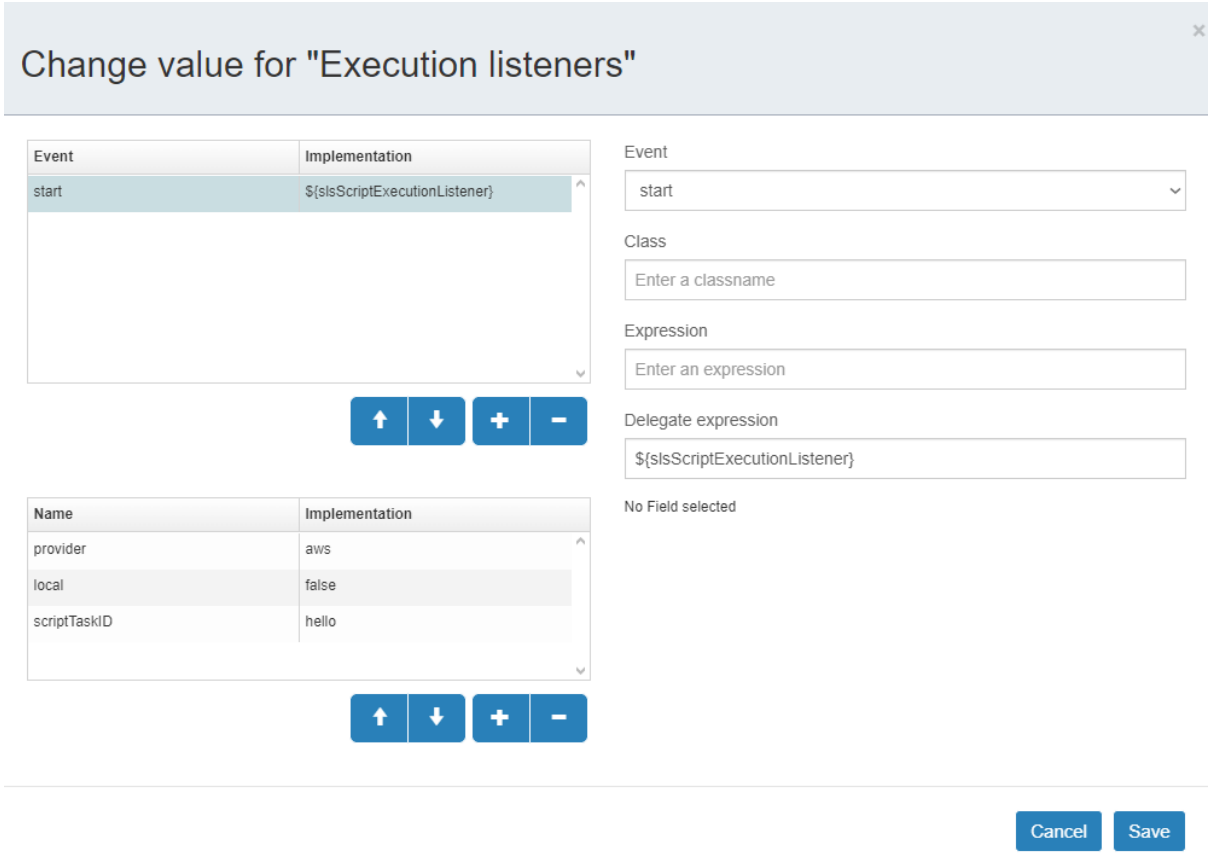
ΕΙΚΟΝΑ 21 - ΠΛΗΡΟΦΟΡΙΕΣ ΠΟΥ ΑΠΑΡΤΙΖΟΥΝ ΕΝΑ SCRIPT TASK ΣΤΟ FLOWABLE

Στο πλαίσιο της ενσωμάτωσης του Serverless Framework μέσα στο WFMS, η ενσωμάτωση επιτρέπει την εκτέλεση **serverless συναρτήσεων** απευθείας από τα **Script Tasks** της ροής εργασίας, εφόσον είναι σωστά διαμορφωμένα. Αυτή η ενσωμάτωση επιτυγχάνεται μέσω των πεδίων του **Script Task,** όπως περιγράφεται παρακάτω:

1. Το πεδίο **Script Format** καθορίζει τη γλώσσα προγραμματισμού της serverless συνάρτησης και είναι απαραίτητο στοιχείο για το αρχείο διαμόρφωσης του Serverless Framework. Η τιμή του πεδίου αντιστοιχεί απευθείας στο runtime του εκάστοτε cloud παρόχου, όπως για παράδειγμα nodels14.x για AWS. Οι υποστηριζόμενες τιμές εξαρτώνται από τον πάροχο και το περιβάλλον εκτέλεσης.
2. Στο πεδίο **Script** γράφεται ο πλήρης κώδικας της serverless συνάρτησης που θα χρησιμοποιήσει το Serverless Framework για την διάταξη και την εκτέλεσή της. Σε περίπτωση επαναχρησιμοποίησης ήδη υπάρχουσας serverless συνάρτησης, το πεδίο ορίζεται με μια τιμή no-operation (π.χ. void(0);) και αντίθετα ορίζεται η παράμετρος scriptTaskID του

SLSScriptExecutionListener ώστε να παραπέμπει στο μοναδικό αναγνωριστικό του αρχικού script task.

3. Το πεδίο **Id** είναι απαραίτητο και χρησιμοποιείται ως μοναδικό χαρακτηριστικό για κάθε serverless συνάρτηση.
4. Το πεδίο **Name** είναι απαραίτητο γιατί χρησιμοποιείται κατά την διαδικασία δημιουργίας/επαλήθευσης του αρχείου διαμόρφωσης του Serverless. Πέρα από τον ρόλο τον οποίο ήδη εξυπηρετεί στο Flowable, χρησιμοποιείται στην αναπτυγμένη λύση έτσι ώστε να δηλώσει το όνομα της serverless συνάρτησης.
5. Το πεδίο **Execution Listeners** είναι απαραίτητο διότι για να οριστεί ότι ένα script task περιέχει την υπογραφή μιας serverless συνάρτησης θα πρέπει ο προγραμματιστής της ροής εργασίας να συμπεριλάβει έναν Execution Listener, ο οποίος είναι μια κλάση που περιέχει προσαρμοσμένο κώδικα προς εκτέλεση για της κλήση της συνάρτησης. Στο 5ο βήμα της ενσωμάτωσης πρέπει να οριστεί ο Execution Listener πάνω στο script task. Αυτό επιτυγχάνεται επιλέγοντας Execution Listeners στην Εικόνα 21, προστίθεται ο SLSScriptExecutionListener με τα πεδία: provider (πάροχος νέφους) με πιθανές τιμές aws, google, openwhisk, local (τοπικό η cloud περιβάλλον) και scriptTaskID. Το πεδίο scriptTaskId ορίζεται ίσο με το Id του τρέχοντος script task εφόσον πρόκειται για νέα serverless συνάρτηση ή ίσο με το Id ενός άλλου script task εφόσον επαναχρησιμοποιείται ο κώδικάς εκείνου. Στην Εικόνα 22 φαίνονται οι απαραίτητες πληροφορίες που χρειάζεται να οριστούν έτσι ώστε να είναι δυνατή η εφαρμογή της προτεινόμενης λύσης στο πλαίσιο εκτέλεσης του script που ορίζεται εντός μιας ροής εργασίας. Οπότε, παρατηρούμε την παρακάτω διαμόρφωση:
    1. **Event (Γεγονός):**
       Το πεδίο **Event** έχει οριστεί στην τιμή **start**, δηλώνοντας ότι αυτός ο listener θα ενεργοποιηθεί όταν ξεκινήσει η εκτέλεση μιας ροής ή δραστηριότητας (σε αυτή την περίπτωση του script task).
    2. **Delegate Expression (Έκφραση Ανάθεσης):**
       Η τιμή του **Delegate expression** είναι ορισμένη ως **${slsScriptExecutionListener}.** Αυτό σημαίνει ότι εκτελείται μια έκφραση ανάθεσης που παραπέμπει σε έναν listener (ή κλάση) με το όνομα SLSScriptExecutionListener, η οποία έχει υλοποιηθεί από το σύστημά μας.
    3. **Παράμετροι (Parameters):** Στο κάτω μέρος της εικόνας, βλέπουμε μια λίστα με προσαρμοσμένες παραμέτρους που έχουν οριστεί. Αυτές είναι απαραίτητες τόσο στην διαδικασία της υποβολής αλλά και όταν εκτελείται η ροή εργασίας.
        - **Provider:** Ο cloud πάροχος / serverless πλατφόρμα που θα χρησιμοποιήσει το Serverless Framework στο αρχείο διαμόρφωσης. Η επέκταση υποστηρίζει τις τιμές **aws**, **openwhisk** και **google**.
        - **Local:** Δηλώνει αν η serverless συνάρτηση θα εκτελεστεί τοπικά ή σε απομακρυσμένη πλατφόρμα. Χρησιμοποιείται επίσης για να σημάνει τι πληροφορίες θα γραφτούν στο αρχείο διαμόρφωσης του Serverless Framework. Αν έχουμε τοπική εκτέλεση, τότε θα πρέπει ο cloud πάροχος να είναι ο **openwhisk**.
        - **scriptTaskID:** Δηλώνει το μοναδικό αναγνωριστικό του script task που θα υλοποιηθεί από τη serverless συνάρτηση. Για νέα συνάρτηση ορίζεται ίδιο με το ID του τρέχοντος script task. Για την επαναχρησιμοποίηση ήδη υπάρχουσας serverless συνάρτησης ορίζεται το ID εκείνου του script task που την ορίζει στο πεδίο script. Για παράδειγμα, αν βρισκόμαστε σε εκτέλεση ροής εργασίας και το node (τρέχων στοιχείο της ροής εργασίας) που κοιτάμε είναι Script Task με αναγνωριστικό **X** και περιέχει τον Execution Listener που προαναφέρθηκε (SLSScriptExecutionListener) με παράμετρο scriptTaskID **Y**, τότε ο listener θα καλέσει εκείνη τη συνάρτηση που ορίζεται και άρα σχετίζεται άμεσα με το script task με αναγνωριστικό **Y.**



ΕΙΚΟΝΑ 22 - ΑΝΑΔΥΟΜΕΝΟ ΠΑΡΑΘΥΡΟ ΓΙΑ ΕΙΣΑΓΩΓΗ/ΤΡΟΠΟΠΟΙΗΣΗ ΤΩΝ EXECUTION LISTENERS ΚΑΘΩΣ ΚΑΙ ΤΩΝ ΜΕΤΑΒΛΗΤΩΝ ΤΟΥΣ.

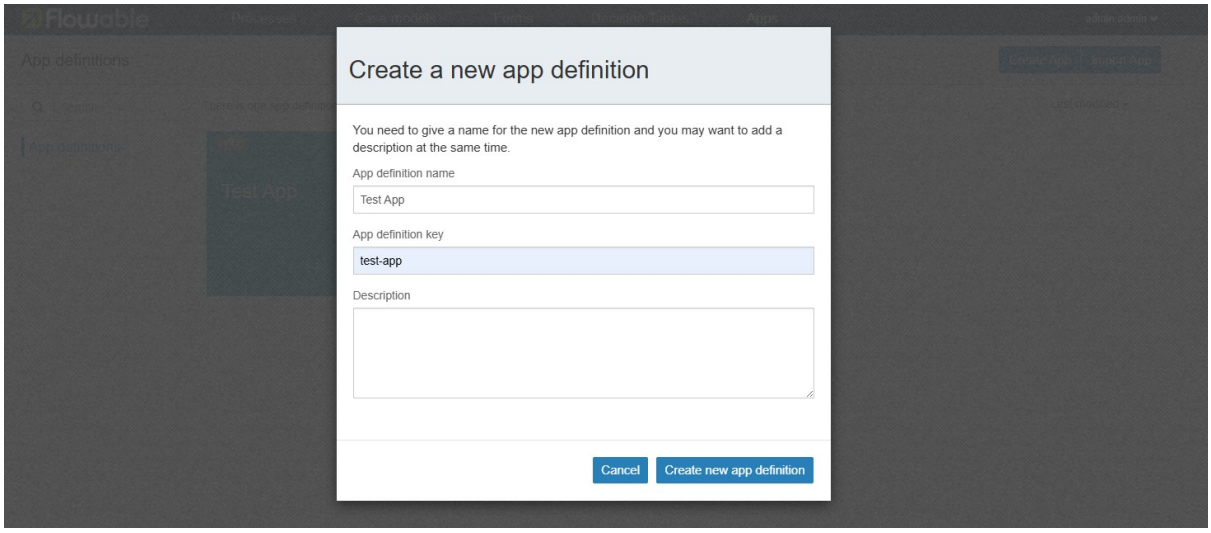
Για να αποτυπωθεί η πρακτική αξιοποίηση της προτεινόμενης λύσης, παρουσιάζεται παρακάτω ένα απλό αλλά ενδεικτικό σενάριο χρήσης. Η περίπτωση αφορά τον ορισμό και την εκτέλεση μιας ροής εργασίας η οποία αξιοποιεί serverless συναρτήσεις με στόχο την επίδειξη της ενσωμάτωσης τους και της επαναχρησιμοποίησής τους στην ίδια ροή εργασίας.

Ως προς το παράδειγμα χρήσης στην συνέχεια θα παρουσιαστεί η σχεδίαση και υλοποίηση μιας ροής εργασίας στο περιβάλλον Flowable που αποτελείται από τρία διαδοχικά βήματα. Τα δύο πρώτα βήματα ορίζουν έκαστο μία διαφορετική serverless συνάρτηση σε Javascript. Οι δύο συναρτήσεις αυτές θα φιλοξενηθούν από τους cloud παρόχους AWS και Google, αντίστοιχα. Το τρίτο βήμα επαναχρησιμοποιεί τη συνάρτηση του πρώτου βήματος, αναδεικνύοντας τη δυνατότητα επαναχρησιμοποίησης ήδη αναπτυγμένων serverless συναρτήσεων.

Βήματα για σχεδίαση του παραδείγματος χρήσης:

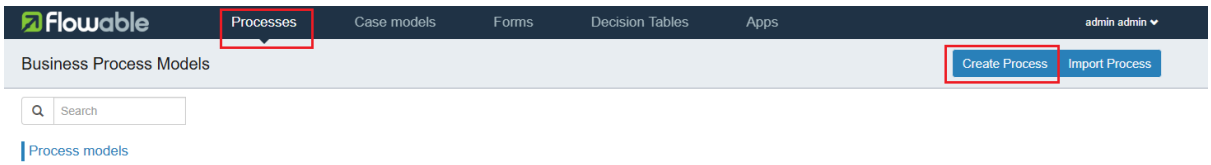
1. Πηγαίνοντας στην διεύθυνση ***http://localhost:8080/#/apps*** πρέπει να δημιουργηθεί ορισμός εφαρμογής (app definition). Στην Εικόνα 23 φαίνεται με κόκκινο περίγραμμα το κουμπί που πρέπει να πατηθεί για να ξεκινήσει η διαδικασία ορισμού εφαρμογής. Αφού πατηθεί το κουμπί για την δημιουργία ορισμού εφαρμογής, ακολουθεί η συμπλήρωση των στοιχείων όπως φαίνεται στην Εικόνα 24, και η υποβολή με το πάτημα του κουμπιού «Create new app definition».

ΕΙΚΟΝΑ 23 – ΜΕΝΟΥ ΟΡΙΣΜΕΝΩΝ ΕΦΑΡΜΟΓΩΝ (APP DEFINITIONS)

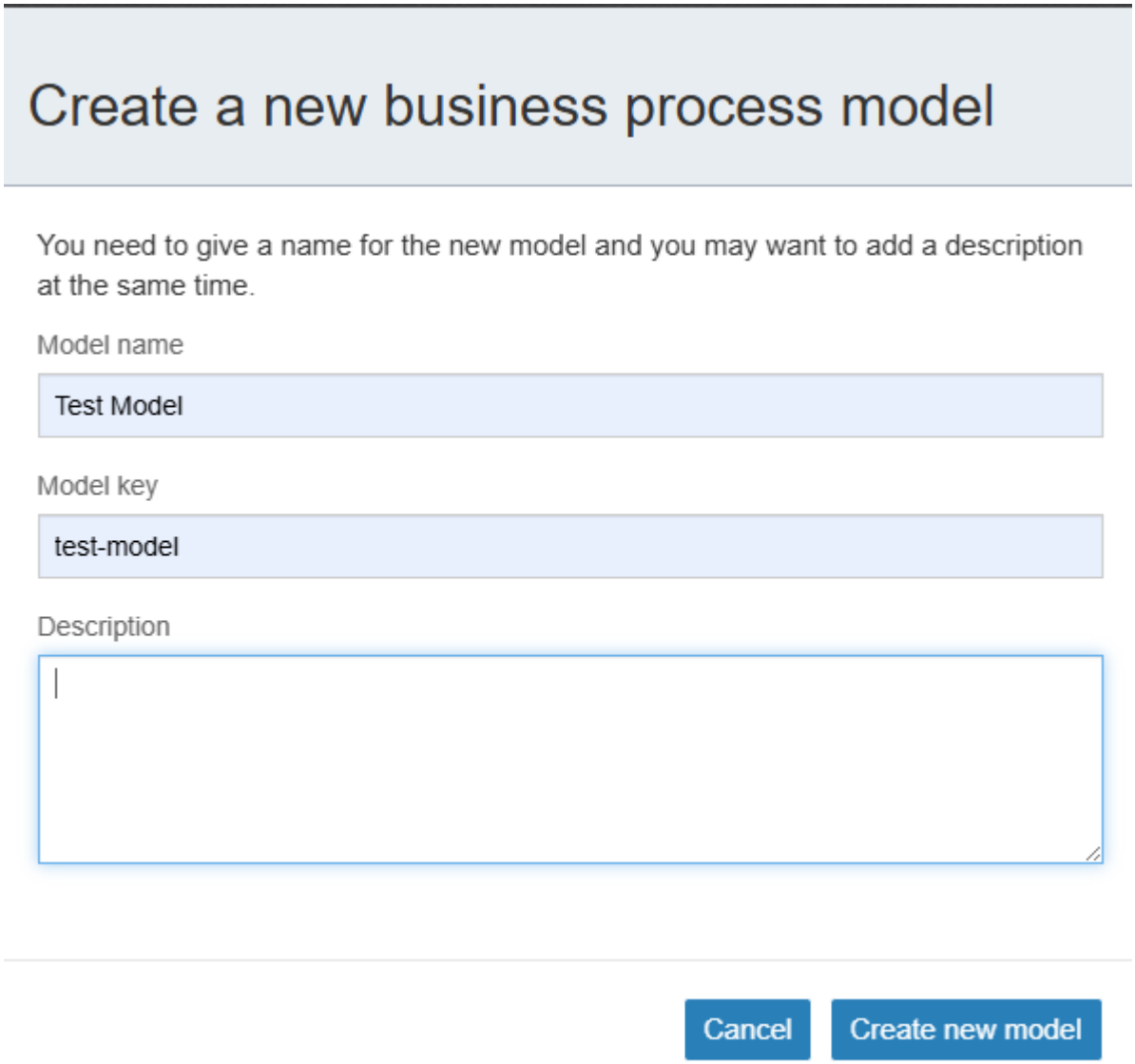


ΕΙΚΟΝΑ 24 - ΑΝΑΔΥΟΜΕΝΟ ΠΑΡΑΘΥΡΟ ΟΡΙΣΜΟΥ ΕΦΑΡΜΟΓΗΣ

2. Πηγαίνοντας στο μενού των ροών εργασίας πατώντας «Processes» και ύστερα το κουμπί «Create Process» για την δημιουργία ροής εργασίας, όπως φαίνεται στην Εικόνα 25 και Εικόνα 26, αντίστοιχα.

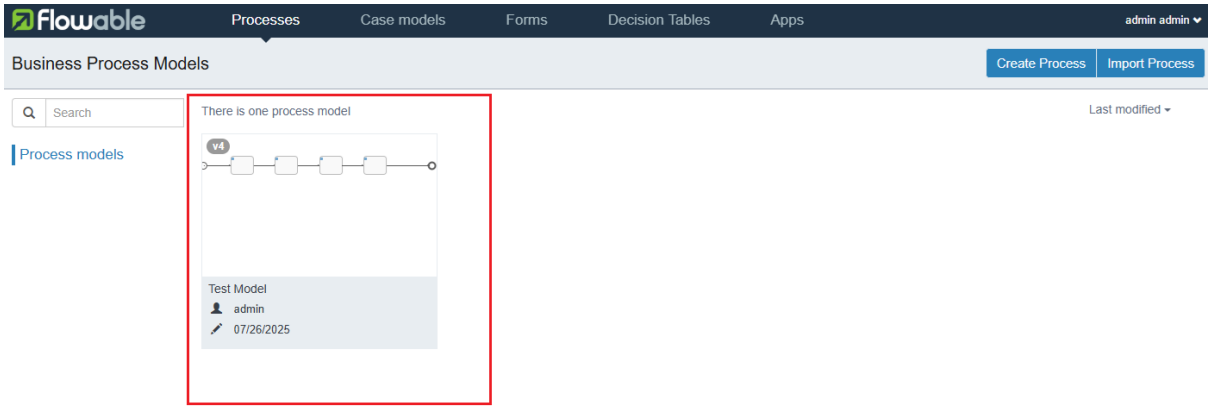


ΕΙΚΟΝΑ 25 - ΜΕΝΟΥ ΟΡΙΣΜΕΝΩΝ ΡΟΩΝ ΕΡΓΑΣΙΑΣ

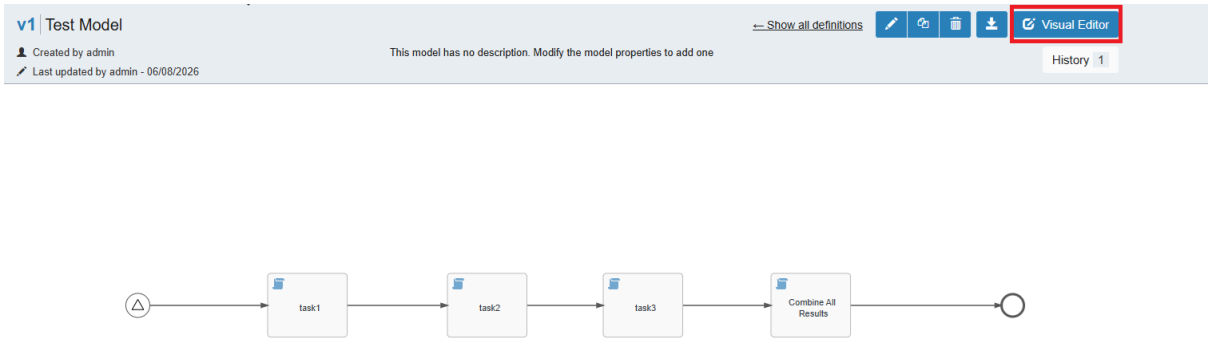


ΕΙΚΟΝΑ 26 - ΑΝΑΔΥΟΜΕΝΟ ΠΑΡΑΘΥΡΟ ΔΗΜΙΟΥΡΓΙΑΣ ΡΟΗΣ ΕΡΓΑΣΙΑΣ

3. Εφόσον δημιουργηθεί η ροή εργασίας και επιβεβαιωθεί ότι φαίνεται στο μενού ορισμένων ροών εργασίας, πρέπει να πατηθεί η ίδια η ροή, όπως φαίνεται στην Εικόνα 27. Στην συνέχεια πρέπει να πατηθεί το κουμπί «Visual Editor» μέσω του οποίου θα σχεδιαστεί η ροή εργασίας όπως φαίνεται στην Εικόνα 28.

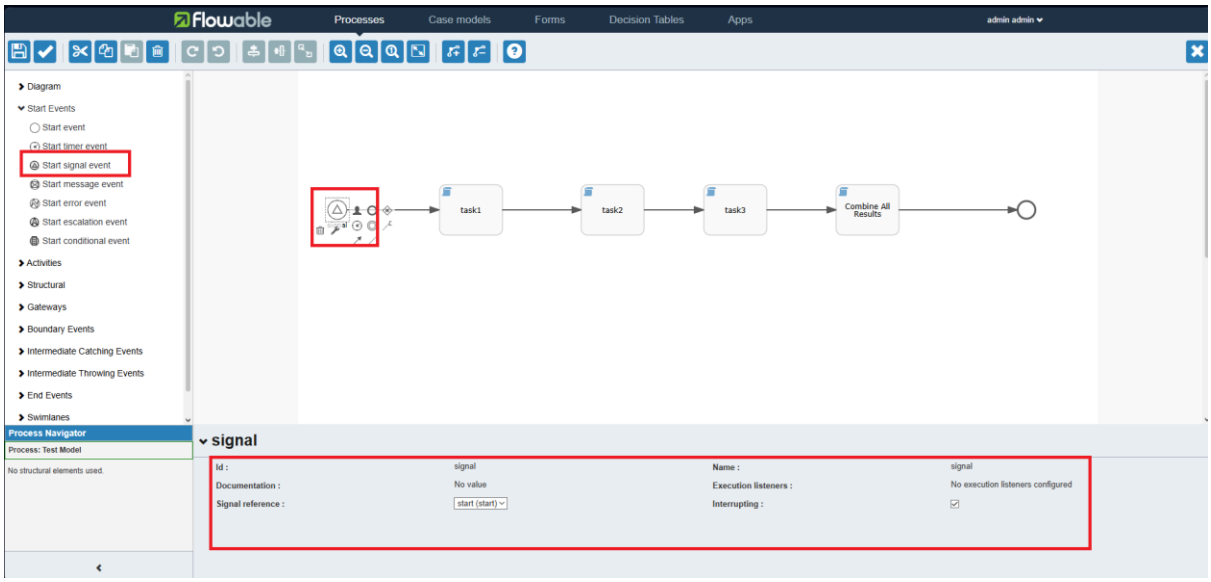


ΕΙΚΟΝΑ 27 - ΔΗΜΙΟΥΡΓΗΜΕΝΗ ΡΟΗ ΕΡΓΑΣΙΑΣ «TEST MODEL»



ΕΙΚΟΝΑ 28 - VISUAL EDITOR ΡΟΗΣ ΕΡΓΑΣΙΑΣ

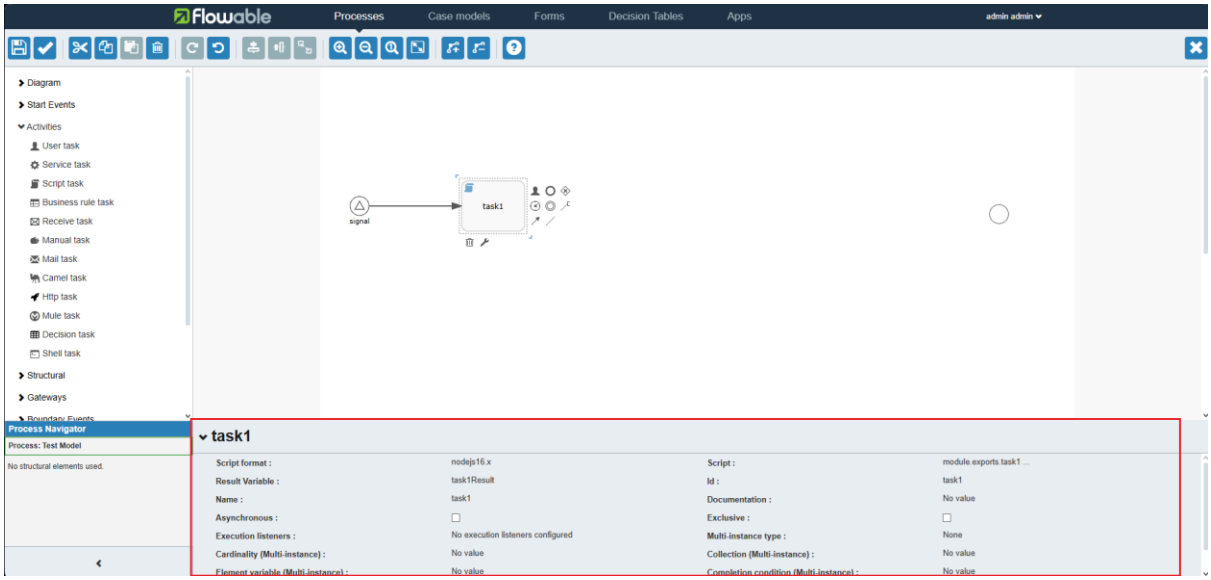
4. Δημιουργία αρχικού γεγονότος βάσει του οποίου θα ξεκινήσει να εκτελείται η ροή όπως φαίνεται στην Εικόνα 29.



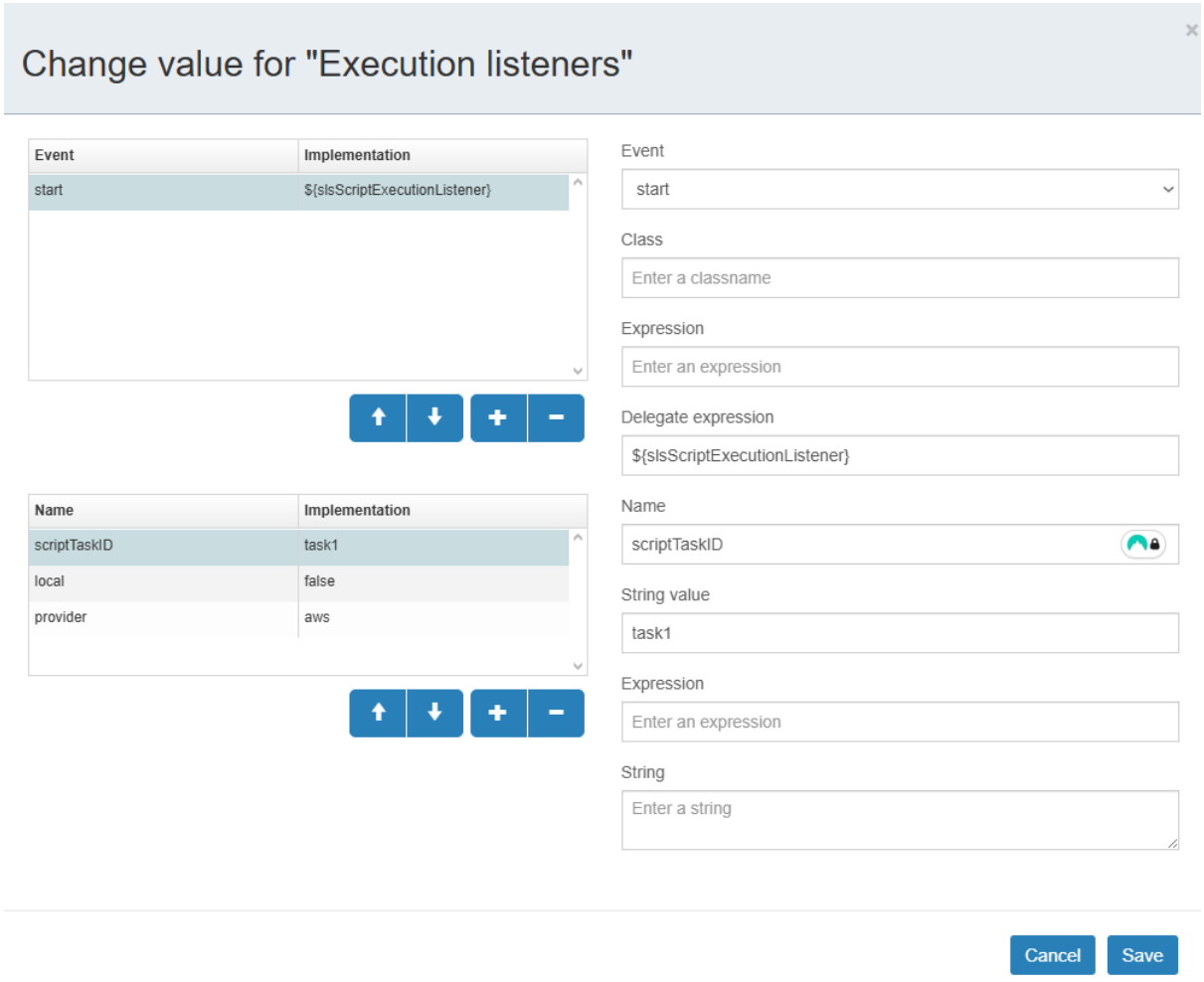
ΕΙΚΟΝΑ 29 - ΔΗΜΙΟΥΡΓΙΑ ΓΕΓΟΝΟΤΟΣ ΕΚΚΙΝΗΣΗΣ (START EVENT SIGNAL)

5. Δημιουργία ενός script task, το οποίο περιέχει δήλωση μιας serverless συνάρτησης. Για να διαμορφωθούν οι τιμές των πεδίων του, ο χρήστης πρέπει πρώτα να επιλέξει (κλικ) το script task πάνω στο διάγραμμα, ώστε να εμφανιστεί το αντίστοιχο πάνελ ιδιοτήτων στο κάτω μέρος της οθόνης. Στην Εικόνα 31 φαίνεται στο κόκκινο πλαίσιο οι πληροφορίες που απαρτίζουν το ίδιο το script task και οι οποίες πρέπει να συμπληρωθούν από τον προγραμματιστή χειροκίνητα. Τα περισσότερα πεδία συμπληρώνονται απευθείας ως απλό κείμενο μέσα στο πάνελ, με εξαίρεση το «Script» και τα «Execution listeners» που ανοίγουν το καθένα ξεχωριστό παράθυρο διαμόρφωσης.
    A. Η τιμή του «Script format» αρχικοποιείται ως **nodejs16.x** που είναι η έκδοση της γλώσσας προγραμματισμού Javascript προς χρήση.
    B. Η τιμή του «Script» περιέχει τον κώδικα, όπως φαίνεται στην Εικόνα 30.
    Γ. Η τιμή του «Id» αρχικοποιείται με task1.
    Δ. Η τιμή του «Result Variable» αρχικοποιείται με task1Result και θα περιέχει το αποτέλεσμα εκτέλεσης της serverless συνάρτησης.
    E. Η τιμή του «Name» αρχικοποιείται με task1.
    ΣΤ. Η τιμή του «Execution listeners» πρέπει να αρχικοποιηθεί όπως φαίνεται στην Εικόνα 32.

ΕΙΚΟΝΑ 30 - ΚΩΔΙΚΑΣ 1ΟΥ SCRIPT TASK



ΕΙΚΟΝΑ 31 - ΓΡΑΦΙΚΗ ΔΙΕΠΑΦΗ ΓΙΑ ΤΗΝ ΔΗΜΙΟΥΡΓΙΑ ΡΟΗΣ ΕΡΓΑΣΙΑΣ

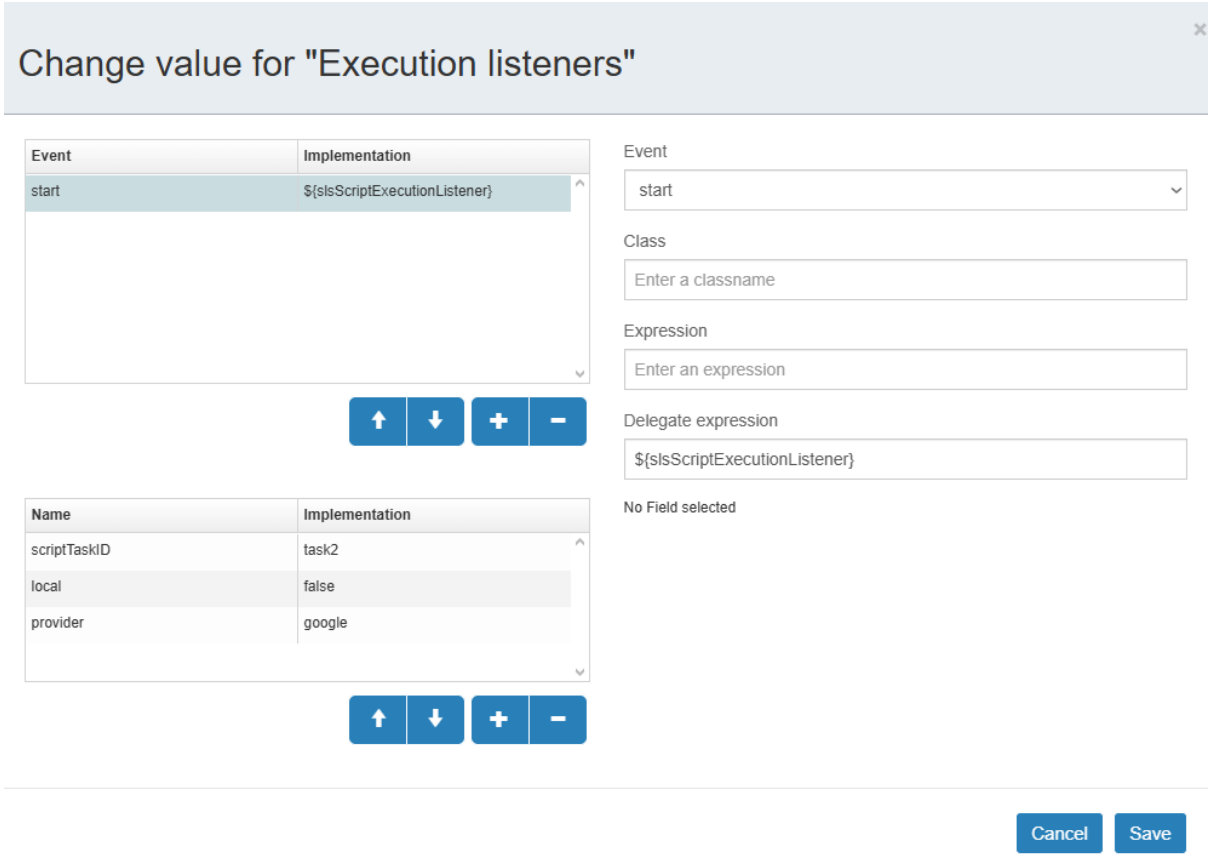


ΕΙΚΟΝΑ 32 - EXECUTION LISTENER ΓΙΑ ΤΗΝ SERVERLESS ΣΥΝΑΡΤΗΣΗ ΤΟΥ 1ΟΥ SCRIPT TASK

6. Δημιουργία ενός script task, το οποίο περιέχει δήλωση μιας serverless συνάρτησης. Οι πληροφορίες θα συμπληρωθούν με το ίδιο τρόπο όπως και στο προηγούμενο βήμα.

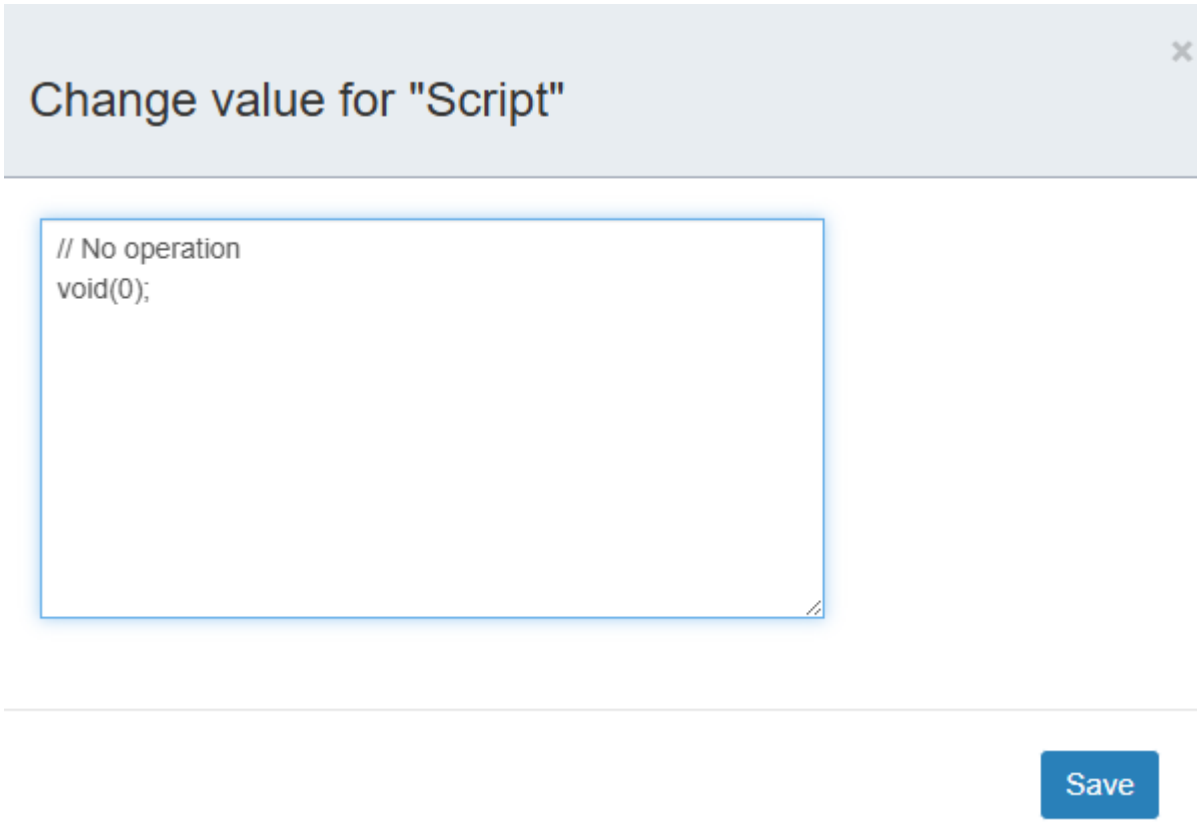
Α. Η τιμή του «Script format» αρχικοποιείται ως **nodejs20** που είναι η έκδοση της γλώσσας προγραμματισμού Javascript προς χρήση.
Β. Η τιμή του «Script» περιέχει τον κώδικα όπως φαίνεται στην Εικόνα 33.
Γ. Η τιμή του «Id» αρχικοποιείται με task2.
Δ. Η τιμή του «Result Variable» αρχικοποιείται με task2Result.
Ε. Η τιμή του «Name» αρχικοποιείται με task2.
ΣΤ. Η τιμή του «Execution listeners» πρέπει να αρχικοποιηθεί όπως φαίνεται στην Εικόνα 34.

ΕΙΚΟΝΑ 33 - ΚΩΔΙΚΑΣ 2ΟΥ SCRIPT TASK

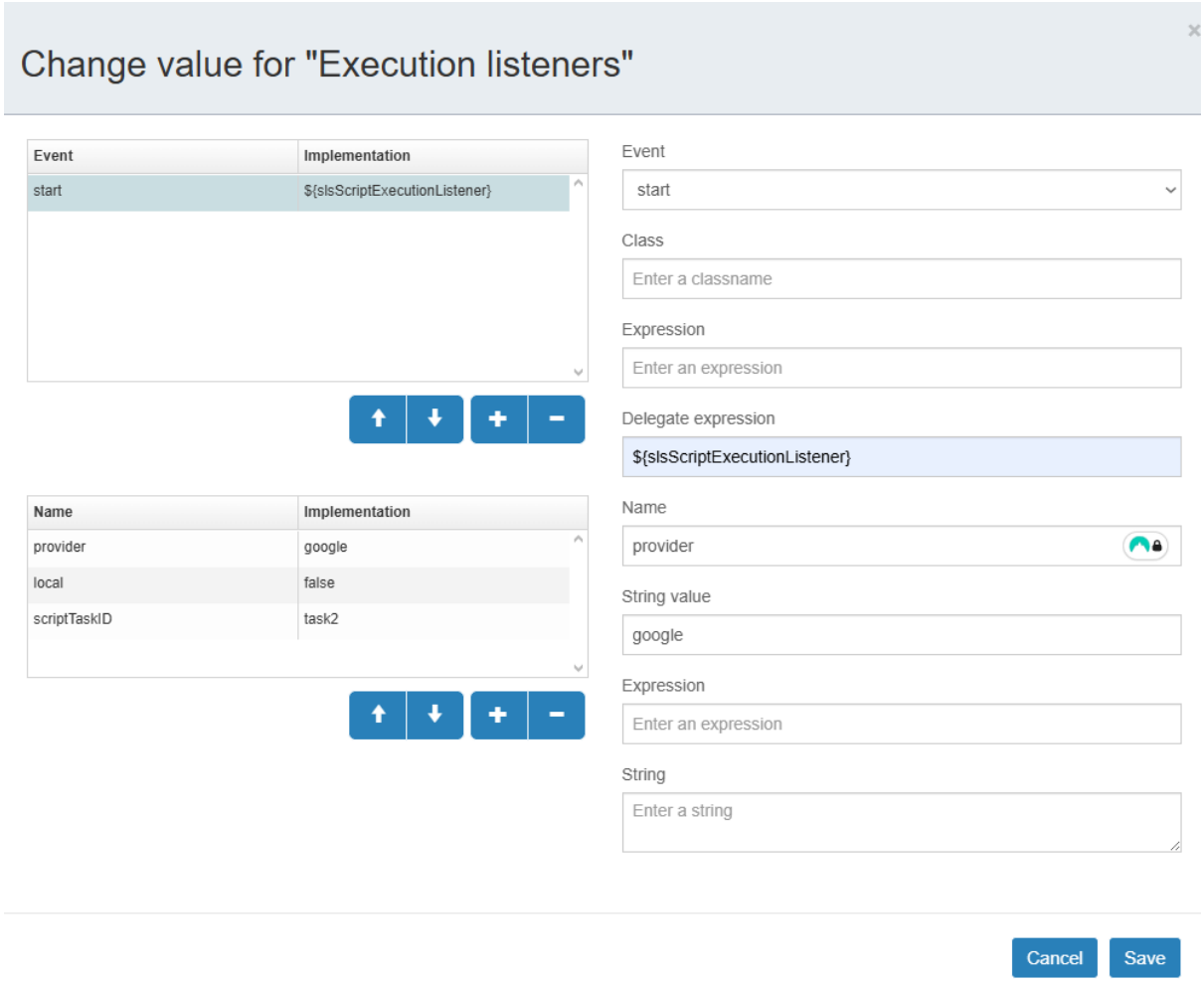


ΕΙΚΟΝΑ 34 - EXECUTION LISTENER ΓΙΑ ΤΗΝ SERVERLESS ΣΥΝΑΡΤΗΣΗ ΤΟΥ 2ΟΥ SCRIPT TASK

7. Δημιουργία ενός script task, το οποίο περιέχει δήλωση μιας ήδη υπάρχουσας serverless συνάρτησης. Οι πληροφορίες που πρέπει να συμπληρωθούν είναι οι εξής:
    A. Η τιμή του «Script» πρέπει να αρχικοποιηθεί όπως φαίνεται στην Εικόνα 35.
    B. Η τιμή του «Id» αρχικοποιείται με task3.
    Γ. Η τιμή του «Result Variable» αρχικοποιείται με task3Result.
    Δ. Η τιμή του «Name» αρχικοποιείται με task3.
    E. Η τιμή του «Execution listeners» πρέπει να αρχικοποιηθεί όπως φαίνεται στην Εικόνα 36. Με τον τρόπο αυτό θέτοντας ως scriptTaskId το taskId2, ο listener αναζητά κατά την εκτέλεση την καταχωρημένη serverless συνάρτηση του 2ου script task και την καλεί.

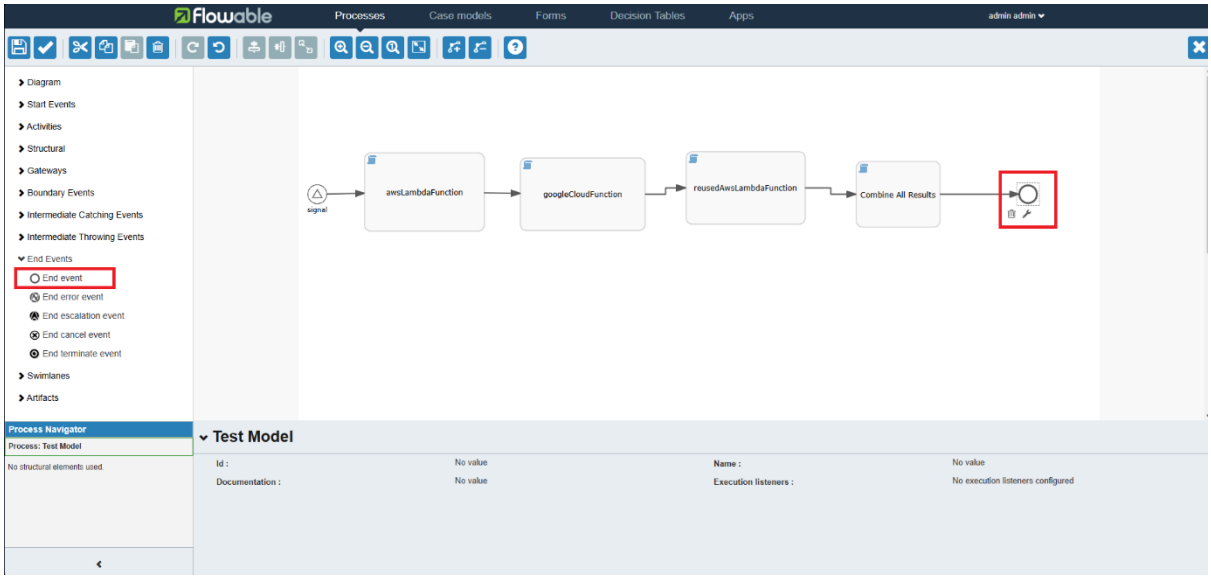


ΕΙΚΟΝΑ 35 - ΚΩΔΙΚΑΣ 3ΟΥ SCRIPT TASK



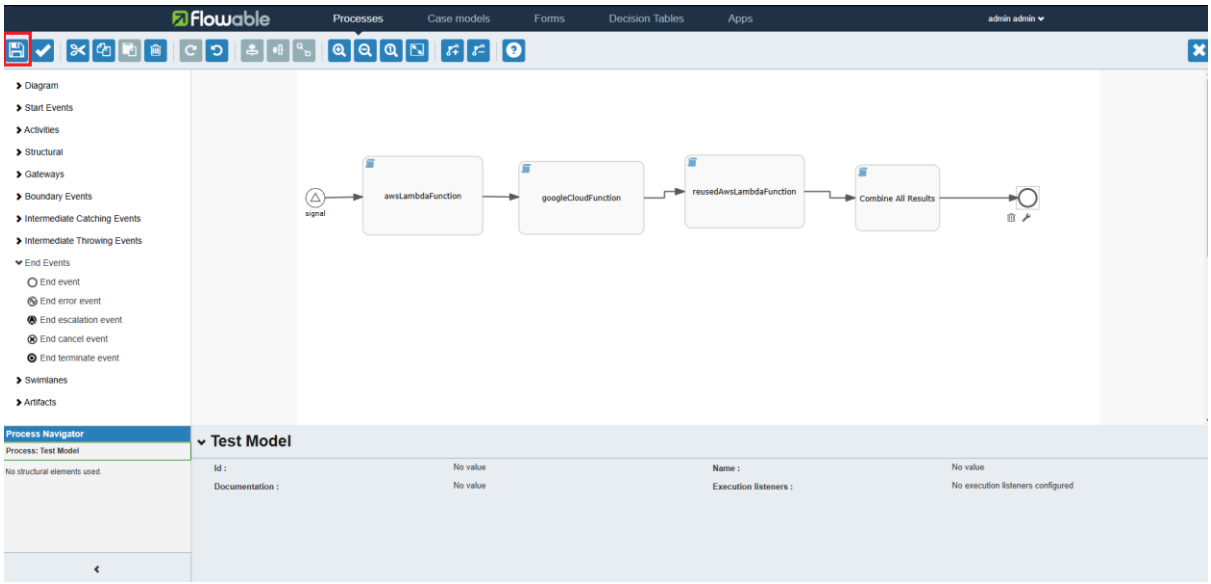
ΕΙΚΟΝΑ 36 - EXECUTION LISTENER ΓΙΑ ΤΗΝ SERVERLESS ΣΥΝΑΡΤΗΣΗ ΤΟΥ 3ΟΥ SCRIPT TASK

8. Δημιουργία τελικού γεγονότος όπως φαίνεται στην Εικόνα 37.

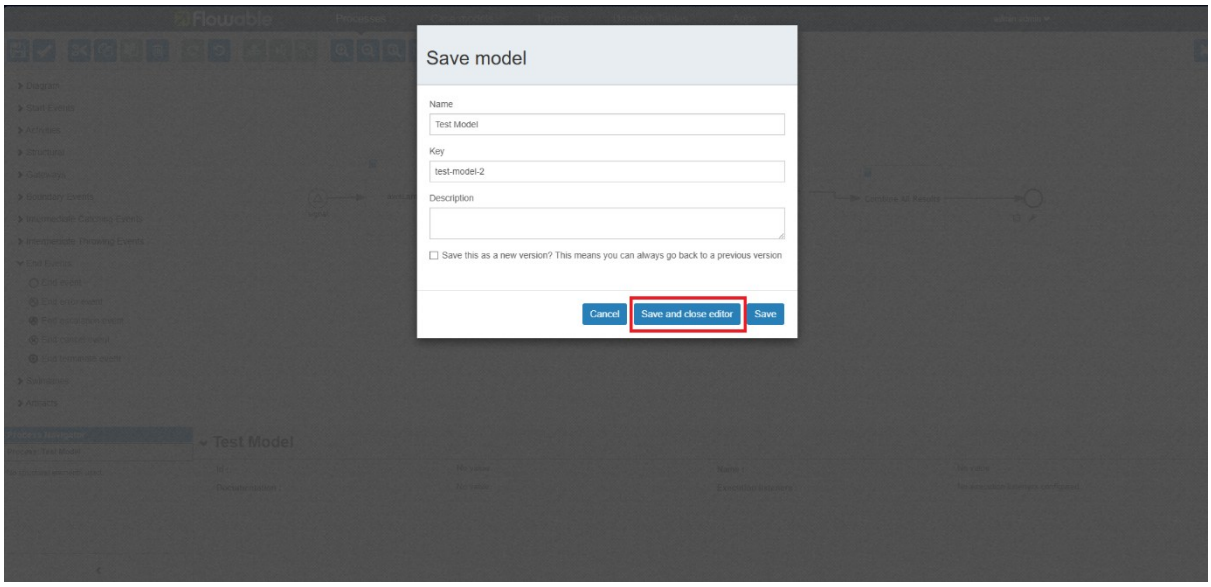


ΕΙΚΟΝΑ 37 - ΔΗΜΙΟΥΡΓΙΑ ΤΕΛΙΚΟΥ ΓΕΓΟΝΟΤΟΣ (END EVENT SIGNAL)

9. Αποθήκευση ροής εργασίας όπως φαίνεται στην Εικόνα 38 και στην Εικόνα 39.

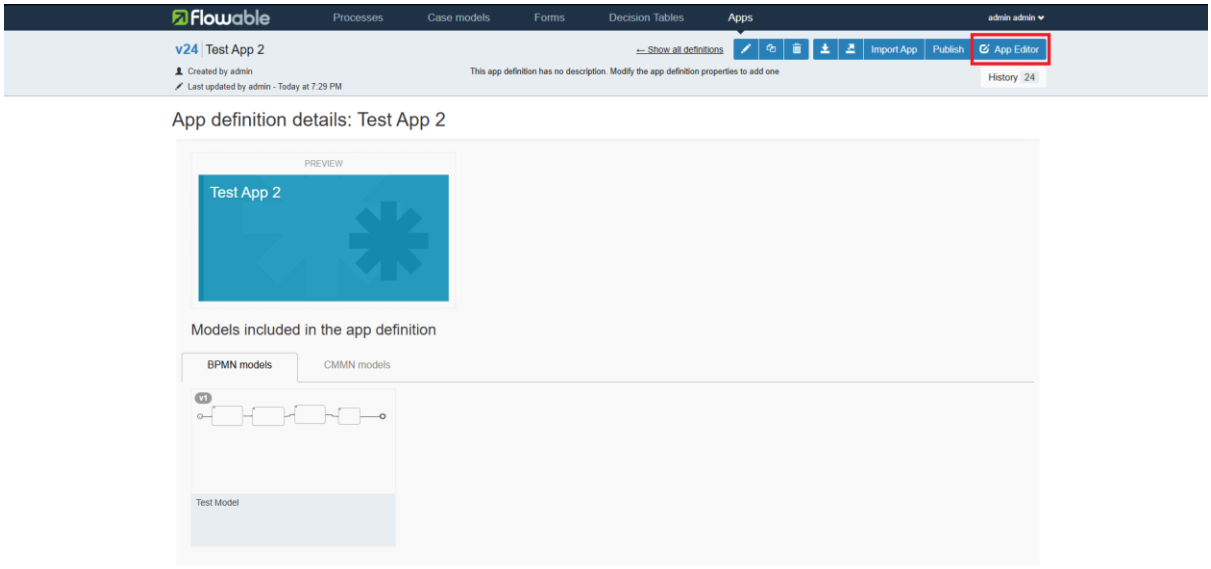


ΕΙΚΟΝΑ 38 - ΑΠΟΘΗΚΕΥΣΗ ΡΟΗΣ ΕΡΓΑΣΙΑΣ

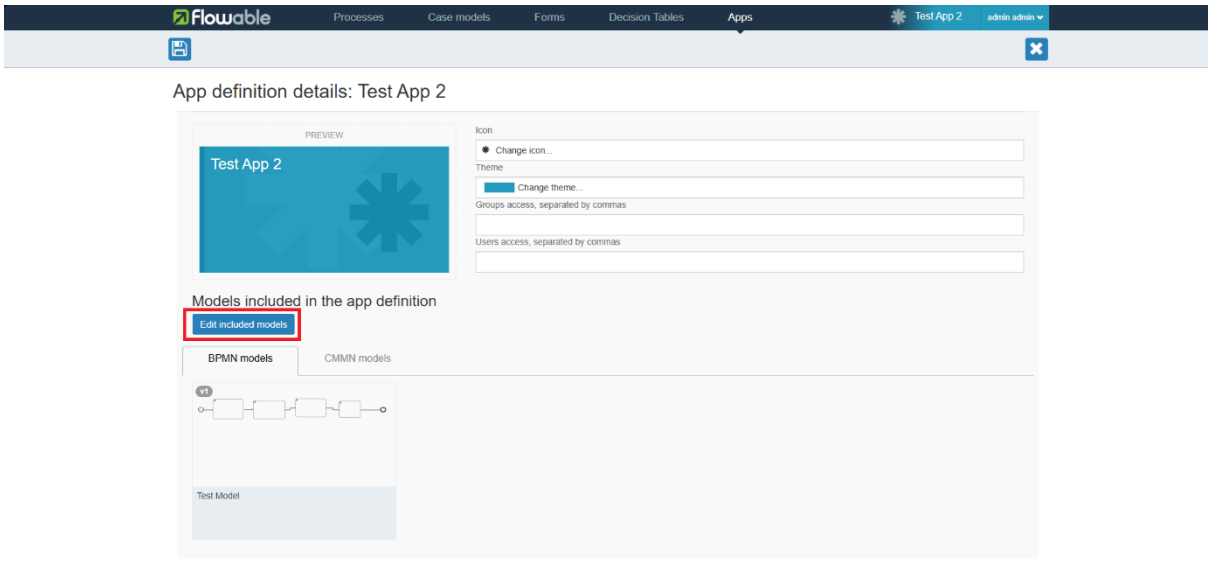


ΕΙΚΟΝΑ 39 - ΑΠΟΔΟΧΗ ΑΠΟΘΗΚΕΥΣΗΣ ΤΗΣ ΡΟΗΣ ΕΡΓΑΣΙΑΣ

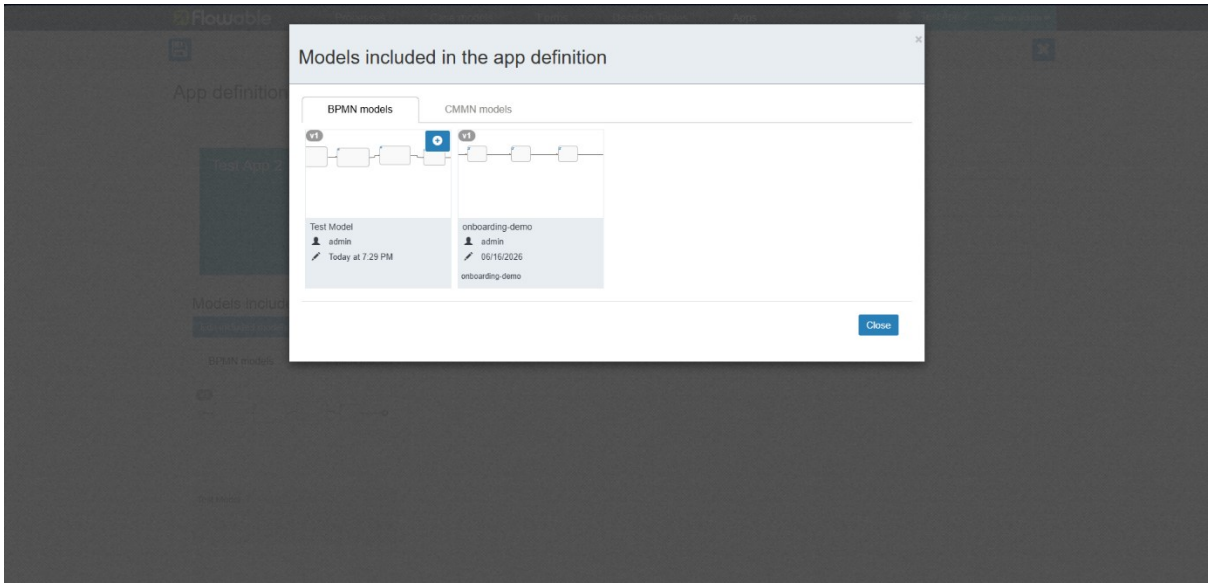
10. Αντιστοιχία ροής εργασίας με ορισμό εφαρμογής από το 1ο βήμα. Πηγαίνοντας στο μενού ορισμού εφαρμογής, όπως είδαμε στην Εικόνα 23, και πατώντας το κουμπί «App Editor» στην Εικόνα 40 ο προγραμματιστής μπορεί να τροποποιήσει τον ορισμό εφαρμογής αντιστοιχίζοντας ροές εργασίας, όπως φαίνεται στην Εικόνα 41 και στην Εικόνα 42.



ΕΙΚΟΝΑ 40 - ΤΡΟΠΟΠΟΙΗΣΗ ΟΡΙΣΜΟΥ ΕΦΑΡΜΟΓΗΣ

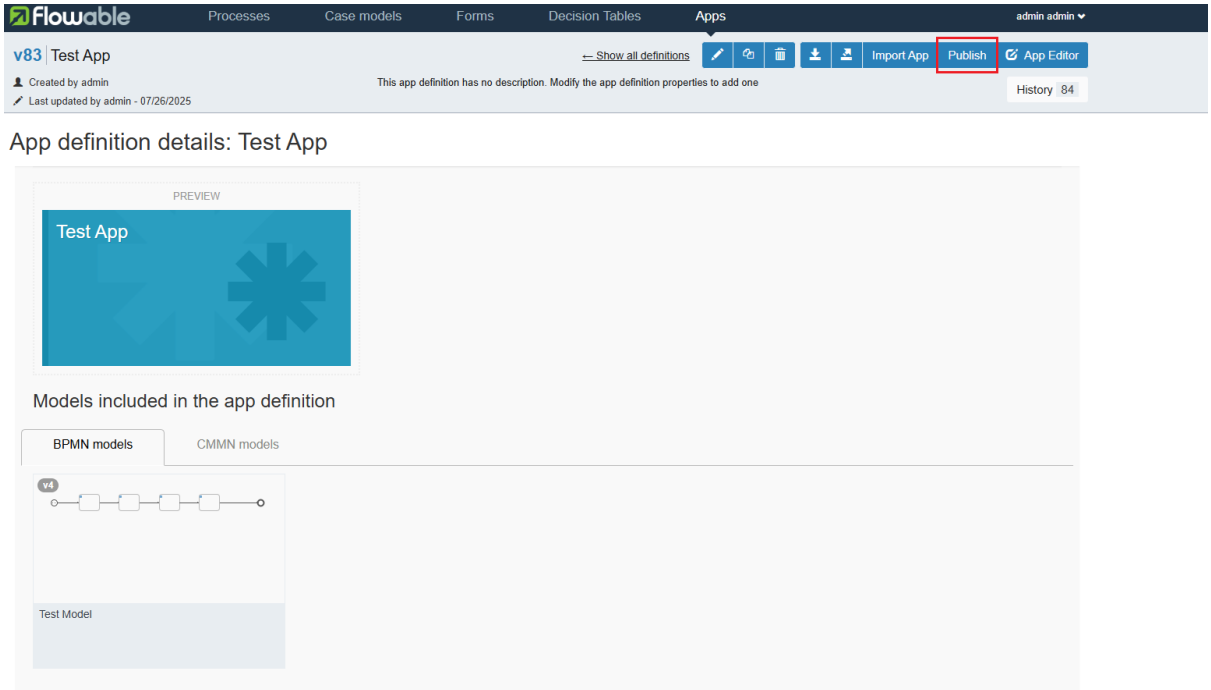


ΕΙΚΟΝΑ 41 - ΑΝΤΙΣΤΟΙΧΙΣΗ ΡΟΩΝ ΕΡΓΑΣΙΑΣ ΣΕ ΟΡΙΣΜΟ ΕΦΑΡΜΟΓΗΣ (1)

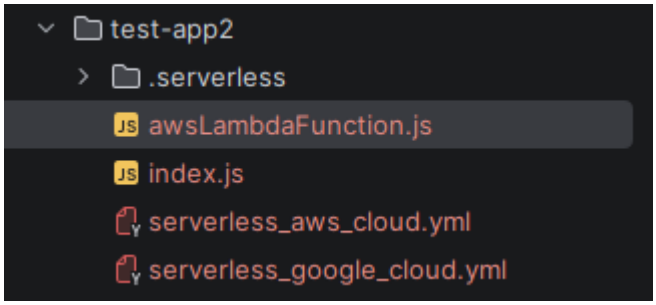


ΕΙΚΟΝΑ 42 - ΑΝΤΙΣΤΟΙΧΙΣΗ ΡΟΩΝ ΕΡΓΑΣΙΑΣ ΣΕ ΟΡΙΣΜΟ ΕΦΑΡΜΟΓΗΣ (2)

11. Δημοσίευση ορισμού εφαρμογής. Σε αυτό το βήμα μόλις πατηθεί το κουμπί «Publish», το οποίο φαίνεται στην Εικόνα 43, θα ξεκινήσει η διαδικασία πακεταρίσματος και διάταξης των serverless συναρτήσεων. Κατά τη διάρκεια της διαδικασίας υποβολής παράχθηκαν τα ακόλουθα αρχεία που περιέχουν τον κώδικα των συναρτήσεων (1 αρχείο για κάθε serverless συνάρτηση) και τα αρχεία διαμόρφωσης (1 αρχείο για κάθε cloud πάροχο που χρησιμοποιείται), όπως φαίνεται στην Εικόνα 44.



ΕΙΚΟΝΑ 43 - ΜΕΝΟΥ ΔΙΑΧΕΙΡΙΣΗΣ ΟΡΙΣΜΟΥ ΕΦΑΡΜΟΓΗΣ



ΕΙΚΟΝΑ 44 - ΠΑΡΑΓΜΕΝΑ ΑΡΧΕΙΑ ΚΩΔΙΚΑ & ΔΙΑΜΟΡΦΩΣΗΣ

Τέλος, παρουσιάζεται η επιτυχής διάταξη/πακετάρισμα της serverless συνάρτησης του task1 της ροής εργασίας ξεκινώντας από την Εικόνα 45.

```
2025-08-31 19:22:09.275  INFO 24628 --- [nio-8080-exec-3] c.f.s.SLSPlatformService                 : Deployment result: Deploying function: task1
Deployment output:
dashboard: https://app.serverless.com/galiats/apps/test/test/dev/us-east-1
endpoints:
  GET - https://d6r5v8hf0i.execute-api.us-east-1.amazonaws.com/dev/hello
  GET - https://d6r5v8hf0i.execute-api.us-east-1.amazonaws.com/dev/task1
functions:
  hello: test-dev-hello (203 kB)
  task1: test-dev-task1 (203 kB)

Deployment error output:
Deploying test to stage dev (us-east-1, "default" provider)
Warning: Skipping wrapping for Function: hello
✓ Your AWS account is currently being integrated with Serverless Framework Observability
✓ Serverless Framework's Observability features being set up on your AWS account (one-time set-up).
  An email will be sent upon completion, or view progress within the Dashboard:
  https://app.serverless.com/galiats/settings/integrations

✓ Service deployed to stack test-dev (60s)
```

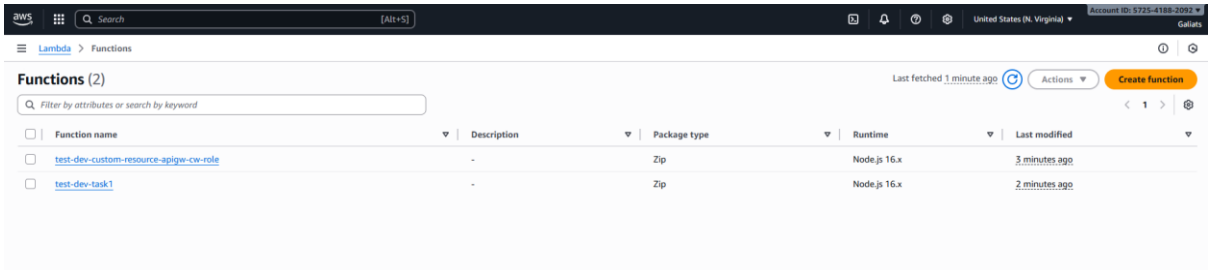
ΕΙΚΟΝΑ 45 - ΑΠΟΣΠΑΣΜΑ ΕΠΙΤΥΧΗΜΕΝΟΥ ΠΑΚΕΤΑΡΙΣΜΑΤΟΣ ΚΑΙ ΔΙΑΤΑΞΗΣ (ΜΕΡΟΣ 1ο)

Συνεχίζοντας με την Εικόνα 46 μπορείτε να παρατηρήσετε την επιτυχή διάταξη/πακετάρισμα της serverless συνάρτησης του task2.

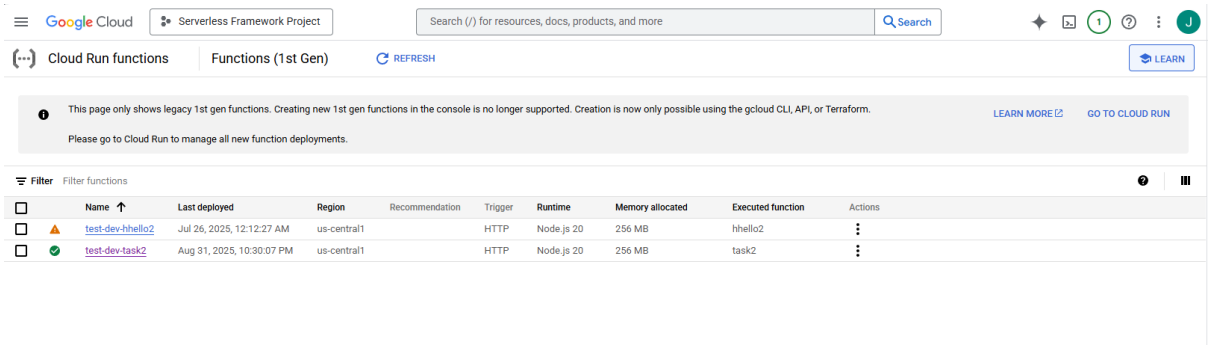
```
Compiling function "task2"...
Uploading artifacts...
Artifacts successfully uploaded...
Updating deployment...
Checking deployment update progress...
Environment: win32, node 20.17.0, framework 3.37.0, plugin 7.2.3, SDK 4.5.1
Docs:        docs.serverless.com
Support:     forum.serverless.com
Bugs:        github.com/serverless/serverless/issues
```

ΕΙΚΟΝΑ 46 - ΑΠΟΣΠΑΣΜΑ ΕΠΙΤΥΧΗΜΕΝΟΥ ΠΑΚΕΤΑΡΙΣΜΑΤΟΣ ΚΑΙ ΔΙΑΤΑΞΗΣ (ΜΕΡΟΣ 2ο)

Επιπροσθέτως στην Εικόνα 47 και Εικόνα 48 φαίνονται οι serverless συναρτήσεις στις πλατφόρμες των cloud παρόχων που δημιουργήθηκαν, αντίστοιχα.



ΕΙΚΟΝΑ 47 - ΑΠΟΣΠΑΣΜΑ SERVERLESS ΣΥΝΑΡΤΗΣΗΣ ΤΟΥ TASK1 ΣΕ AWS LAMBDA



ΕΙΚΟΝΑ 48 - ΑΠΟΣΠΑΣΜΑ SERVERLESS ΣΥΝΑΡΤΗΣΗΣ ΤΟΥ TASK2 ΣΤΟ GOOGLE CLOUD

5.2.3. Εκτέλεση Ροής Εργασίας με Serverless Script Tasks

Μια ροή εργασίας στο Flowable μπορεί να ξεκινήσει με διάφορους τρόπους:

- Μέσω του γραφικού περιβάλλοντος
- Προγραμματιστικά μέσω του RuntimeService που παρέχει το Flowable
- Μέσω REST API κλήσης για την εκκίνηση της ροής.

Στην παρούσα υλοποίηση η εκκίνηση πραγματοποιείται μέσω REST API κλήσης με χρήση curl, όπως φαίνεται παρακάτω ξεκινώντας την ροή μέσω του γεγονότος εκκίνησης:

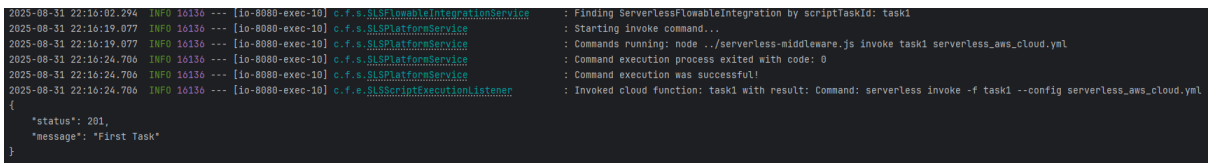
curl --location 'http://localhost:8080/flowableApp/start/process/' \

--header 'Content-Type: text/plain' \

--header 'Authorization: ••••••' \

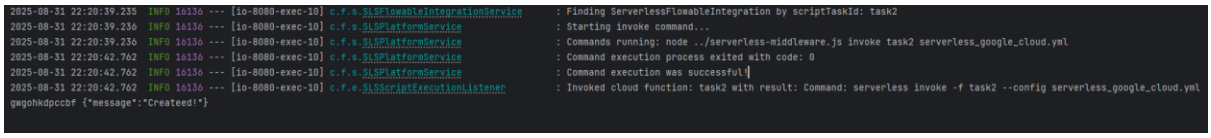
--data 'start'

Αφού λοιπόν έχει ολοκληρωθεί η διαδικασία, γίνεται η εκκίνηση της ροής εργασίας και τα αναμενόμενα αποτελέσματα φαίνονται στην Εικόνα 49, Εικόνα 50, Εικόνα 51 (όπου παρουσιάζεται ανά εικόνα η εκτέλεση της αντίστοιχης serverless συνάρτησης, καλύπτοντας με αυτό τον τρόπο τα tasks με IDs task1, task2 & task3).



ΕΙΚΟΝΑ 49 - ΕΚΤΕΛΕΣΗ ΤΗΣ SERVERLESS ΣΥΝΑΡΤΗΣΗΣ (TASK1)

ΕΙΚΟΝΑ 50 - ΕΚΤΕΛΕΣΗ ΤΗΣ SERVERLESS ΣΥΝΑΡΤΗΣΗΣ (TASK2)



ΕΙΚΟΝΑ 51 - ΕΚΤΕΛΕΣΗ ΤΗΣ SERVERLESS ΣΥΝΑΡΤΗΣΗΣ (TASK3)

6. Συμπεράσματα & Μελλοντική Εργασία

6.1. Συμπεράσματα

Το σύστημα που αναπτύχθηκε στο πλαίσιο της διπλωματικής αυτής εργασίας προσφέρει τη δυνατότητα ενσωμάτωσης κώδικα serverless συναρτήσεων σε script tasks σε ροές εργασίας BPMN, εστιάζοντας στο σύστημα διαχείρισης ροών εργασίας Flowable. Τα βασικά χαρακτηριστικά του συστήματός μας περιλαμβάνουν την υποστήριξη πολλαπλών γλωσσών προγραμματισμού για την υλοποίηση των συναρτήσεων, τη δυνατότητα τοπικής ή απομακρυσμένης εκτέλεσης τους, την επικύρωση του κώδικα των συναρτήσεων, καθώς και το αυτόματο πακετάρισμα και διάταξή τους σε οποιαδήποτε serverless πλατφόρμα, με την οποία συνεργάζεται το Serverless Framework.

Ένα από τα μεγαλύτερα πλεονεκτήματα της λύσης είναι η ευελιξία που προσφέρει στον σχεδιασμό και την εκτέλεση ροών εργασίας, ενώ παράλληλα μειώνει την εξάρτηση από συγκεκριμένους παρόχους. Επιπλέον πλεονεκτήματα είναι η σημαντική μείωση της πολυπλοκότητας στον σχεδιασμό και την εκτέλεση ροών εργασίας που περιέχουν serverless συναρτήσεις καθώς και η αύξηση της αποδοτικότητας και της προσαρμοστικότητας σε διαφορετικά σενάρια υλοποίησης.

Η εμπειρία που αποκτήθηκε μέσα από την ανάπτυξη του συστήματος ήταν εξαιρετικά σημαντική, ειδικά όσον αφορά τη χρήση εργαλείων όπως το Flowable και το Serverless Framework. Αντιμετωπίστηκαν κρίσιμα ζητήματα, όπως η διαχείριση πολλαπλών συναρτήσεων με διαφορετικές ρυθμίσεις για διαφορετικούς παρόχους/πλατφόρμες, η ορθή διαχείριση σφαλμάτων καθώς και αρκετά ζητήματα που προέκυψαν λόγω της διαφορετικής αρχιτεκτονικής που ακολουθεί κάθε πάροχος/πλατφόρμα.

Κατά την υλοποίηση, βασικά εμπόδια περιλάμβαναν την ενσωμάτωση διαφορετικών πάροχων FaaS για την επίτευξη πλήρους αυτοματοποίησης των συναρτήσεων κατά τη διαχείρισή τους. Με τη χρήση προσαρμοσμένων στρατηγικών και εργαλείων, τα προβλήματα αυτά ξεπεράστηκαν, αναδεικνύοντας τη δυναμική της serverless τεχνολογίας και την εφαρμογή της στον τομέα της αυτοματοποίησης επιχειρηματικών διαδικασιών. Εν κατακλείδι, το σύστημά μας προσφέρει μια ολοκληρωμένη, επεκτάσιμη λύση που μπορεί να προσαρμοστεί σε μελλοντικές ανάγκες και εφαρμογές.

6.2. Μελλοντική Εργασία

Υπάρχουν πολλές κατευθύνσεις για μελλοντική εργασία που θα μπορούσαν να επεκτείνουν τη λειτουργικότητα της προτεινόμενης λύσης ή να τη βελτιώσουν περαιτέρω:

1. **Υποστήριξη Service Tasks:** Επέκταση της λύσης ώστε να υποστηρίζει και service tasks, εκτός από τα script tasks, για μεγαλύτερη ευελιξία και πληρέστερη ενσωμάτωση με BPMN ροές εργασίας. Επειδή η κλήση της συνάρτησης γίνεται μέσα σε Execution Listener και όχι μέσα στο ίδιο το task, ένα σφάλμα δεν μπορεί να πιαστεί από boundary error event πάνω στο συγκεκριμένο task παρά μόνο αν υπάρχει boundary error event σε ανώτερο επίπεδο, με αποτέλεσμα μια αποτυχημένη κλήση να απαιτεί χειρισμό σφάλματος σε επίπεδο ολόκληρης διαδικασίας, αντί για μεμονωμένη επανάληψη ή εναλλακτική διαδρομή στο συγκεκριμένο task. Μελλοντική εργασία θα μπορούσε να προσθέσει υποστήριξη σε service tasks, όπου η κλήση γίνεται μέσω Java Delegate ως κύρια συμπεριφορά του task επιτρέποντας boundary error events ανά task και άρα λεπτομερή διαχείριση σφαλμάτων ανά serverless κλήση.
2. **Αποδοτικότερη διαχείριση πεδίων του FLowable:** Αν και ο Flowable Modeler υποστηρίζει ήδη τον ορισμό script tasks, η δήλωση μιας serverless συνάρτησης απαιτεί σήμερα τη χειροκίνητη συμπλήρωση διάσπαρτων πεδίων (Script, Script Format, Id, Name) και ξεχωριστά την προσθήκη και παραμετροποίηση του SLSScriptExecutionListener (Provider, Local, ScriptTaskID), μια διαδικασία επιρρεπής σε λάθη όπως λανθασμένο Name που δεν αποτελεί

έγκυρο όνομα συνάρτησης ή λανθασμένο scriptTaskId σε περίπτωση επαναχρησιμοποίησης. Μια custom φόρμα αφιερωμένη στην ένταξη serverlessσυναρτήσεων σε script tasks θα μπορούσε να ενοποιήσει τα πεδία αυτά σε ένα σημείο, να ζητά από τον χρήστη μόνο την ελάχιστη απαραίτητη πληροφορία και σε περίπτωση επαναχρησιμοποίησης να προσφέρει λίστα επιλογής (dropdown) των ήδη διατεταγμένων συναρτήσεων αντί να απαιτείται η χειροκίνητη πλληκτρολόγηση του scriptTaskId.

3. **Αυξημένη Παρακολούθηση και Αναφορές:** Ενσωμάτωση ενός γραφικού συστήματος με dashboards για την παρακολούθηση (της απόδοσης) των serverless συναρτήσεων σε πραγματικό χρόνο, καθώς και δημιουργία λεπτομερών αναφορών που περιλαμβάνουν στατιστικά απόδοσης και ανάλυση σφαλμάτων.
4. **Διεπαφή Χρήστη (UI):** Δημιουργία γραφικής διεπαφής, ενσωματωμένης στο ίδιο το Flowable (με τον ίδιο τρόπο που υπάρχουν ήδη ενότητες διαχείρισης για ορισμό εφαρμογής και για ροές εργασίας), για τη διαχείριση των serverless συναρτήσεων, όπως προβολή των ήδη διατεταγμένων συναρτήσεων απευθείας από τον πίνακα στη βάση δεδομένων, καθώς και δυνατότητες, όπως επανάληψη διάταξης ή αφαίρεση μιας συνάρτησης, χωρίς να απαιτούνται ιδιαίτερες τεχνικές γνώσεις.
5. **Δυναμική Ρύθμιση Παραμέτρων:** Το παραγόμενο αρχείο διαμόρφωσης του Serverless Framework δεν περιλαμβάνει σήμερα ρυθμίσεις πέρα από τα βασικά οπότε κάθε συνάρτηση τρέχει με τις προεπιλογές του cloud provider. Η μόνη διέξοδος για κάτι διαφορετικό είναι ένα πλήρως custom αρχείο διαμόρφωσης, που απαιτεί γνώση του Serverless Framework. Μελλοντική εργασία θα μπορούσε να εκθέσει τις πιο συνήθεις ρυθμίσεις ως απλά πεδία στο μοντέλο ή στην custom φόρμα, μεταφραζόμενα αυτόματα σε YAML, χωρίς ο προγραμματιστής να χρειάζεται να γνωρίζει το Serverless Framework.
6. **Αυτόματη παραγωγή serverless συναρτήσεων από περιγραφή σε φυσική γλώσσα:** Με την χρήση των LLMs θα επιτρέπεται σε έναν χρήστη να περιγράψει με λέξεις τι θέλει να κάνει η συνάρτηση και να παραχθεί αυτόματα ο αντίστοιχος κώδικας.
7. **Αυτόματη δημιουργία αρχείων διαμόρφωσης**: Για ανάγκες που ξεπερνούν τις συνήθεις ρυθμίσεις (π.χ. ενοποίηση με μια επιπλέον υπηρεσία, όπως πρόσβαση σε DynamoDB), ο χρήστης θα μπορούσε να περιγράψει τις ιδιαιτερότητες της διάταξης σε φυσική γλώσσα, ώστε ένα κατάλληλα καθοδηγούμενο μοντέλο AI να παράγει αυτόματα το αντίστοιχο custom αρχείο διαμόρφωσης για την επιθυμητή πλατφόρμα, αντί να απαιτείται χειροκίνητη συγγραφή του.
8. **Ενσωμάτωση AI στον κύκλο ζωής των ροών εργασίας**: Το AI μπορεί να βοηθήσει όχι μόνο στη συγγραφή και διαμόρφωση του κώδικα των serverless συναρτήσεων, αλλά και στο πώς οι συναρτήσεις συνδέονται σε μια ροή, προτείνοντας βήματα, βελτιώσεις ή εναλλακτικές.

Οι προτεινόμενες κατευθύνσεις μπορούν να ενισχύσουν σημαντικά τη λειτουργικότητα και την αποδοτικότητα της λύσης, διασφαλίζοντας την ευελιξία και τη βιωσιμότητά της σε ευρύτερες εφαρμογές .

Βιβλιογραφία

[1] Google Cloud, "What is Serverless Computing?," [Online]. Available: https://cloud.google.com/discover/what-is-serverless-computing. [Accessed: Jul. 22, 2026].

[2] Cloudflare, "What is Vendor Lock-in?," [Online]. Available: https://www.cloudflare.com/en-gb/learning/cloud/what-is-vendor-lock-in. [Accessed: Jul. 22, 2026].

[3] Wikipedia, "Business process," [Online]. Available: https://en.wikipedia.org/wiki/Business_process. [Accessed: Jul. 22, 2026].

[4] Businessmap, "Workflow vs. Process: How to Differentiate Between the Two?," [Online]. Available: https://businessmap.io/workflow-management/workflow-vs-process. [Accessed: Jul. 22, 2026].

[5] Kuali, "What's the Difference Between Workflow and Process Flow?," [Online]. Available: https://www.kuali.co/post/workflow-process-flow. [Accessed: Jul. 22, 2026].

[6] Wikipedia, "Business Process Model and Notation," [Online]. Available: https://en.wikipedia.org/wiki/Business_Process_Model_and_Notation. [Accessed: Jul. 22, 2026].

[7] Wikipedia, "Case Management Model and Notation (CMMN)," [Online]. Available: https://en.wikipedia.org/wiki/CMMN. [Accessed: Jul. 22, 2026].

[8] Wikipedia, "Decision Model and Notation (DMN)," [Online]. Available: https://en.wikipedia.org/wiki/Decision_Model_and_Notation. [Accessed: Jul. 22, 2026].

[9] Echolon, "Workflow Engines," [Online]. Available: https://www.echolon.de/en/blog/workflow-engines/. [Accessed: Jul. 22, 2026].

[10] Camunda, "Camunda," [Online]. Available: https://camunda.com/. [Accessed: Jul. 22, 2026].

[11] Bonitasoft, "Bonitasoft," [Online]. Available: https://www.bonitasoft.com. [Accessed: Jul. 22, 2026].

[12] ProcessMaker, "ProcessMaker," [Online]. Available: https://www.processmaker.com. [Accessed: Jul. 22, 2026].

[13] Flowable, "Flowable," [Online]. Available: https://www.flowable.com. [Accessed: Jul. 22, 2026].

[14] Lumenalta, "What is Serverless Computing?," [Online]. Available: https://lumenalta.com/insights/what-is-serverless-computing. [Accessed: Jul. 22, 2026].

[15] Akamai, "The Benefits of Serverless Computing Architecture," [Online]. Available: https://www.akamai.com/blog/cloud/the-benefits-of-serverless-computing-architecture. [Accessed: Jul. 22, 2026].

[16] TechTarget, "Top Benefits and Disadvantages of Serverless Computing," [Online]. Available: https://www.techtarget.com/searchcloudcomputing/tip/Top-benefits-and-disadvantages-of-serverless-computing. [Accessed: Jul. 22, 2026].

[17] IBM, "Serverless Use Cases," [Online]. Available: https://www.ibm.com/think/topics/serverless-use-cases. [Accessed: Jul. 22, 2026].

[18] Datadog, "Serverless Architecture Overview," [Online]. Available: https://www.datadoghq.com/knowledge-center/serverless-architecture/. [Accessed: Jul. 22, 2026].

[19] I. Baldini, P. Castro, K. Chang, P. Cheng, S. Fink, V. Ishakian, N. Mitchell, V. Muthusamy, R. Rabbah, A. Slominski, and P. Suter, "Serverless Computing: Current Trends and Open Problems," in *Research Advances in Cloud Computing*, Springer, 2017, pp. 1–20.

[20] E. Jonas, J. Schleier-Smith, V. Sreekanti, C.-C. Tsai, A. Khandelwal, Q. Pu, V. Shankar, J. M. Carreira, K. Krauth, N. Yadwadkar, J. Gonzalez, R. A. Popa, I. Stoica, and D. A. Patterson, "Cloud Programming Simplified: A Berkeley View on Serverless Computing," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2019-3, Feb. 2019.

[21] G. C. Fox, V. Ishakian, V. Muthusamy, and A. Slominski, "Status of Serverless Computing and Function-as-a-Service (FaaS) in Industry and Research," Technical Report, arXiv:1708.08028 [cs.DC], Aug. 2017. [Online]. Available: https://arxiv.org/abs/1708.08028. doi: 10.48550/arXiv.1708.08028.

[22] K. Kritikos and P. Skrzypek, "A Review of Serverless Frameworks," *2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)*, Zurich, Switzerland, 2018, pp. 161-168, doi: 10.1109/UCC-Companion.2018.00051.

[23] V. Yussupov, J. Soldani, U. Breitenbücher, and F. Leymann, "Standards-based modeling and deployment of serverless function orchestrations using BPMN and TOSCA," *Software: Practice and Experience*, vol. 52, no. 6, pp. 1454–1495, Jun. 2022, doi: 10.1002/spe.3073.