ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΛΟΓΙΣΜΙΚΟΥ ΚΑΙ ΠΛΗΡΟΦΟΡΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Σχεδίαση και υλοποίηση ροών εργασιών βασισμένων σε κατευθυνόμενους άκυκλους γράφους. Εφαρμογή των αλληλο-εξαρτήσεων δεδομένων και των διεργασιών των συστημάτων ερευνητικού προγράμματος

Άγγελος Κουτάνης

711161027

Επιβλέπων: Παναγιώτης Γιαννακόπουλος - Καθηγητής

Διπλωματική εργασία υποβληθείσα στο Τμήμα

ΑΙΓΑΛΕΩ, Οκτώβριος 2021

Πανεπιστήμιο Δυτικής Αττικής, Τμήμα Μηχανικών Πληροφορικής και Υπολογιστών

Άγγελος Κουτάνης

ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΛΟΓΙΣΜΙΚΟΥ ΚΑΙ ΠΛΗΡΟΦΟΡΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Η παρούσα διπλωματική εργασία παρουσιάστηκε

από τον

Άγγελο Κουτάνη

711161027

την 11η Οκτωβρίου 2021

Εγκρίθηκε από την τριμελή επιτροπή την14/10του 2021.

Παναγιώτης Γιαννακόπουλος	Γεώργιος Πρεζεράκος	Αθανάσιος Βουλόδημος
Panagiotis Giannakopoulos Digitally signed by Panagiotis Giannakopoulos Date: 2021.10.19 23:16:10 +03'00'	Georgios Prezerakos Digitally signed by Georgios Prezerakos Date: 2021.10.26 12:27:39 +03'00'	Athanasios Voulodimos Digitally signed by Athanasios Voulodimos Date: 2021.10.21 23:42:11 +03'00'

ΔΗΛΩΣΗ ΣΥΓΓΡΑΦΕΑ ΔΙΠΛΩΜΑΤΙΚΗΣ ΕΡΓΑΣΙΑΣ

Ο κάτωθι υπογεγραμμένος Κουτάνης Άγγελος του Δημητρίου, με αριθμό μητρώου 711161027 φοιτητής του Πανεπιστημίου Δυτικής Αττικής της Σχολής Μηχανικών
του Τμήματος Μηχανικών Πληροφορικής και Υπολογιστών, δηλώνω υπεύθυνα ότι:
«Είμαι συγγραφέας αυτής της διπλωματικής εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, οι όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών ή λέξεων, είτε ακριβώς είτε παραφρασμένες, αναφέρονται στο σύνολό τους, με πλήρη αναφορά στους συγγραφείς, τον εκδοτικό οίκο ή το περιοδικό, συμπεριλαμβανομένων και των πηγών που ενδεχομένως χρησιμοποιήθηκαν από το διαδίκτυο. Επίσης, βεβαιώνω ότι αυτή η εργασία έχει συγγραφεί από μένα αποκλειστικά και αποτελεί προϊόν πνευματικής ιδιοκτησίας τόσο δικής μου, όσο και του Ιδρύματος.
Παράβαση της ανωτέρω ακαδημαϊκής μου ευθύνης αποτελεί ουσιώδη λόγο για την ανάκληση του πτυχίου μου»

Ο Δηλών,
Κουτάνης Άγγελος

Η έγκριση της διπλωματικής εργασίας δεν υποδηλοί την αποδοχή των γνωμών του συγγραφέα. Κατά τη συγγραφή τηρήθηκαν οι αρχές της ακαδημαϊκής δεοντολογίας.

ΠΕΡΙΛΗΨΗ

Σχεδίαση και υλοποίηση ροών εργασιών βασισμένων σε κατευθυνόμενους άκυκλους γράφους. Εφαρμογή των αλληλο-εξαρτήσεων δεδομένων και των διεργασιών των συστημάτων ερευνητικού προγράμματος

Άγγελος Κουτάνης

Τα τελευταία χρόνια ο όγκος δεδομένων που διαχειρίζονται οι επιχειρήσεις και οργανισμοί είναι τεράστιος. Συνήθως, τα δεδομένα αυτά προκύπτουν από διάφορες πηγές και σε διάφορες μορφές και πρέπει να επεξεργαστούν προκειμένου να τροφοδοτήσουν μετέπειτα άλλα συστήματα. Για να γίνει αυτό πρέπει να εκτελεστούν κάποιες διεργασίες, οι οποίες λαμβάνουν αυτά τα δεδομένα, τα επεξεργάζονται και τα μεταφέρουν στον προορισμό τους. Το σύνολο αυτών των διεργασιών χαρακτηρίζεται ως σωληναγωγός δεδομένων (Data Pipeline). Για τον σχεδιασμό, την επίβλεψη και τον προγραμματισμό τέτοιων σωληναγωγών χρησιμοποιούνται πλατφόρμες ενορχήστρωσης ροών εργασιών (Workflow Orchestration Platforms) ή αλλιώς συστήματα διαχείρισης ροών εργασιών (Workflow Management Systems - WfMS).

Η εργασία αυτή ερευνά το πεδίο της μηχανικής δεδομένων που ασχολείται συγκεκριμένα με την κατασκευή των σωληναγωγών και εξετάζει κάποιες από τις πιο δημοφιλείς ανοιχτού κώδικα πλατφόρμες ενορχήστρωσης ροών εργασιών. Επιπλέον, επιλέγεται η κατάλληλη πλατφόρμα ενορχήστρωσης ροών εργασιών για να χρησιμοποιηθεί στην ανάπτυξη του εργαλείου Hyperion Community Engagement Tool, ενός εργαλείου που είναι μέρος του Hyperion Project, το οποίο αφορά την προστασία των χώρων πολιτιστικής κληρονομιάς. Τέλος, γίνεται ανάλυση απαιτήσεων, σχεδιασμός και υλοποίηση των διεργασιών που αποτελούν τον σωληναγωγό δεδομένων και υλοποίηση αυτού με την επιλεγμένη πλατφόρμα ενορχήστρωσης ροών εργασιών.

Λέξεις κλειδιά

[Data pipelines, Workflow Management Systems, Process Scheduling, Workflow Orchestration Platforms, Data processing]

ABSTRACT

Design and Implementation of DAG-based workflows. Application of the interdependencies according to the existing data and tasks for an H2020 project

Angelos Koutanis

In recent years the amount of data that companies and organizations are managing is enormous. Usually, these data belong to numerous data sources and can have different formats. It is important that these data will be processed and validated before feeding other systems. In order to make this happen, the orchestration and execution of tasks is necessary. These tasks are retrieving the data from all these sources, processing them and delivering them to the appropriate destination. This set of processes assembles a data pipeline. For the implementation and monitoring of data pipelines there are numerous open-source Workflow Orchestration platforms available. This thesis reviews the field of Data Engineering regarding the fundamentals of data pipelines, as well as, investigates some of the most famous Workflow orchestration platforms. Additionally, the most suitable Workflow Orchestration Platform is picked to implement a data pipeline for the Hyperion Community Engagement Tool. This tool is part of Hyperion Project, a real-life project that helps to protect cultural heritage sites. Finally, before the implementation of the data pipeline; the analysis and definition of requirements for each task is taking place and then the procedure continues further with the implementation of the data pipeline according to the most suitable workflow orchestration platform.

Keywords

[Data pipelines, Workflow Management Systems, Process Scheduling, Workflow Orchestration Platforms, Data processing]

Ευχαριστίες

Θα ήθελα να ευχαριστήσω από τα βάθη της καρδιάς μου όλους τους ανθρώπους που συνέδραμαν για να ολοκληρωθεί αυτή η διπλωματική εργασία. Συγκεκριμένα, θέλω να ευχαριστήσω τον επιβλέποντα καθηγητή μου κ. Παναγιώτη Γιαννακόπουλο για την υπομονή του και τις πολύτιμες συμβουλές του, τον εργοδότη μου κ. Στέφανο Καμαρινόπουλο, ο οποίος μου πρότεινε το συγκεκριμένο θέμα για την διπλωματική μου εργασία, καθώς και την συνάδελφό μου Θεοδώρα Κάραλη για τις πολύτιμες συμβουλές της.
Τέλος, θέλω να ευχαριστήσω τους γονείς μου που ήταν πάντα εκεί για μένα, καθώς και τα αδέρφια μου και τους φίλους μου που πίστεψαν σε μένα.

ΚΑΤΑΛΟΓΟΣ ΠΙΝΑΚΩΝ

ΚΑΤΑΛΟΓΟΣ ΣΧΗΜΑΤΩΝ ΚΑΙ ΕΙΚΟΝΩΝ

ΣΥΜΒΟΛΙΣΜΟΙ

REST	Representational state transfer
API	Application Programming Interface
ETL	Extract-Transfer-Load
ELT	Extract-Load-Transfer
AI	Artificial Intelligence
ML	Machine Learning
DW	Data Warehouse
DAG	Directed Acyclic Graph
WfMS	Workflow Management System
DG	Directed Graph
YAML	Yet Another Markup Language
JSON	JavaScript Object Notation
UI	User Interface
SDK	Software Development Kit
tctl	Temporal command-line tool
HCET	Hyperion Community Engagement Tool

HRAP	Holistic Resilience Assessment Platform
SLD	Styled Layer Descriptor
ICCS	Institute of Communication and Computer Systems
URL	Uniform Resource Locator
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol

ΑΔ	Αποθήκη Δεδομένων
ΒΔ	Βάση Δεδομένων
ΚΑΓ	Κατευθυνόμενος Άκυκλος Γράφος
ΜΜ	Μηχανική Μάθηση
ΣΔΡΕ	Συστήματα Διαχείρησης Ροών Εργασιών
ΔΡΕ	Διάγραμμα Ροής Εργασιών
ΔΕ	Διάγραμμα Εργασιών
ΔΡΔ	Διάγραμμα Ροής Διεργασιών
ΜΔ	Μηχανικός Δεδομένων

Κεφάλαιο 1: Data Pipeline

1.1 Εισαγωγή

Τα τελευταία χρόνια η ανάγκη για διαχείριση μεγάλων όγκων δεδομένων έχει αυξηθεί σημαντικά. Ο Βρετανός μαθηματικός Clive Humby είχε πει "Data is the new oil", και αυτό έχει σήμερα μεγαλύτερη αξία παρά ποτέ. Ωστόσο, τα δεδομένα είναι χρήσιμα μόνο εάν μπορούν να χρησιμοποιηθούν με κάποιον τρόπο. Η σημαντικότητα των δεδομένων είναι τεράστια καθώς χρησιμοποιούνται από πληθώρα εφαρμογών όπως για παράδειγμα στην τροφοδότηση μοντέλων μηχανικής μάθησης, στα Business Analytics και Transactional Analytics, ενώ σημαντικό ρόλο παίζουν και στις εταιρίες σε επίπεδο decision making. Επαγγελματίες που η δουλεία τους βασίζεται κατά ένα μεγάλο μέρος στα δεδομένα είναι οι Data Analysts, οι οποίοι τα χρησιμοποιούν για να κατασκευάζουν διαγράμματα των οποίων τα αποτελέσματα βασίζονται σε δεδομένα ή αποφάσεις βασισμένες σε δεδομένα, οι Data Scientists που δημιουργούν μοντέλα προβλέψεων βασισμένα σε δεδομένα κ.ά.

Λόγω της αυξημένης ζήτησης μεγάλων όγκων δεδομένων σε εφαρμογές όπως αυτές που προαναφέρθηκαν, ήρθε στο προσκήνιο ο τομέας της Μηχανικής Δεδομένων. «*Η Μηχανική Δεδομένων είναι η πρακτική σχεδίασης και κατασκευής συστημάτων τα οποία συλλέγουν, αποθηκεύουν και αναλύουν δεδομένα σε κλίμακα*.» [WP42]

Οι Μηχανικοί Δεδομένων πρέπει να έχουν πολύ καλές γνώσεις Βάσεων Δεδομένων, Λειτουργικών Συστημάτων, Κατανεμημένου Προγραμματισμού, γλωσσών προγραμματισμού αλλά και πολλών άλλων τεχνολογιών.

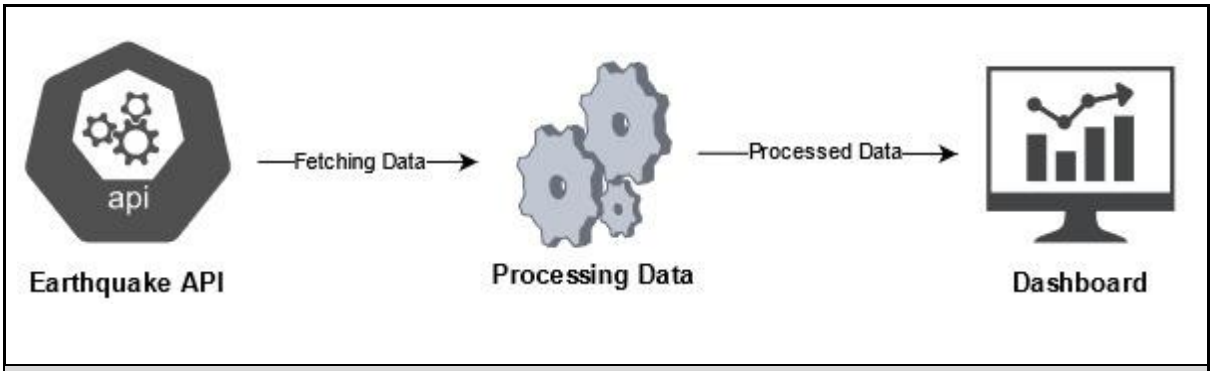
Το σύνολο των διεργασιών που εκτελούνται για να καταστούν τα ανεπεξέργαστα δεδομένα σε δεδομένα έτοιμα προς χρήση, ή και να φορτωθούν τα δεδομένα στο σύστημα προορισμού αναφέρεται συνήθως ως σωληναγωγός δεδομένων.

Σύμφωνα με τον James Densmore [1, p. 1 & p.2] "*Data pipelines are sets of processes that move and transform data from various sources to a destination where new value can be derived. They are the foundation of analytics, reporting, and machine learning capabilities*".

Ο σκοπός ενός σωληναγωγού δεδομένων, λοιπόν, είναι να εξάγει (**extract**) τα δεδομένα, να τα μετασχηματίσει (**transform**), αφαιρώντας την περιττή πληροφορία και να κρατήσει μόνο την απολύτως απαραίτητη, ενώ στην συνέχεια φορτώνει τα μετασχηματισμένα δεδομένα στο σύστημα προορισμού. Τέτοια παραδείγματα είναι η αποθήκευση κατάλληλα διαμορφωμένων δεδομένων σε μία βάση δεδομένων έτσι ώστε να καταναλωθούν για παράδειγμα από μία

συγκεκριμένη εφαρμογή ή η κατασκευή ενός γραφήματος με στατιστικά στοιχεία, το οποίο αργότερα θα παρουσιαστεί στον χρήστη.

Στο παρακάτω παράδειγμα της Εικόνας 1, ο σκοπός του σωληναγωγού δεδομένων είναι να λάβει κάποιες σεισμικές μετρήσεις από ένα API και στην συνέχεια να επεξεργαστεί τα δεδομένα αυτά έτσι ώστε να τροφοδοτήσει έναν πίνακα που βρίσκεται στο γραφικό περιβάλλον που χρησιμοποιεί ο τελικός χρήστης.



Εικόνα 1: Παράδειγμα στο οποίο λαμβάνονται δεδομένα από το Earthquake API και αφού επεξεργαστούν, προβάλλονται σε ένα Dashboard.

1.2 Πηγές Δεδομένων (Data Sources)

Οι περισσότεροι οργανισμοί και εταιρείες διαχειρίζονται ένα πλήθος πηγών δεδομένων. Τα δεδομένα αυτά προέρχονται από τρίτους που κάποιες φορές δίνουν πρόσβαση στα δεδομένα τους μέσω κάποιου REST API. Άλλες φορές τα δεδομένα μπορεί να είναι απλά txt αρχεία, csv αρχεία, μία Βάση Δεδομένων, ένα σύστημα δέσμης δεδομένων (Streaming Data) όπως το Apache Kafka [WP23] κ.ά.

Είναι σπάνιο τα δεδομένα που λαμβάνονται από τις διάφορες πηγές δεδομένων να είναι ακριβώς στην μορφή που τα χρειαζόμαστε. Για αυτόν τον λόγο υπάρχουν κάποιες τεχνικές που βοηθούν τον Μηχανικό Δεδομένων να μετασχηματίσει (Transform) τα δεδομένα, να εξάγει (Extract) τα δεδομένα από διάφορες πηγές και να τα φορτώσει (Load) στον προορισμό τους. Τέτοιες τεχνικές είναι οι ETL και ELT, τις οποίες ο μηχανικός δεδομένων (Data Engineer) καλείται να εφαρμόσει, ανάλογα με το τι θέλει τα επιτυγχάνει ο σωληναγωγός, τον οποίο δημιουργεί

1.3 Αποθήκη Δεδομένων (Data Warehouse)

1.3.1 Παραδοσιακή Αποθήκη Δεδομένων

"*Μία Αποθήκη Δεδομένων είναι ουσιαστικά μία Βάση Δεδομένων στην οποία αποθηκεύονται δεδομένα που έχουν εξαχθεί από διάφορες πηγές και έχουν μετασχηματιστεί και μοντελοποιηθεί με τέτοιο τρόπο ώστε να μπορούν να απαντήσουν συγκεκριμένα ερωτήματα (Queries) που σχετίζονται με ανάλυση (Analytics) και αναφορές (Reports)*". (James Densmore, 2021)[1 ,p.14]

Ακόμα ένας πολύ εύστοχος ορισμός της Αποθήκης Δεδομένων δόθηκε από τον William H. Inmon στο [6]: *«A data warehouse is a:*

- *subject oriented,*
- *integrated,*
- *time variant,*
- *non volatile*

collection of data in support of management's decision-making process. »

1.3.2 Αποθήκη Δεδομένων στο Νέφος (Data Warehouse in the Cloud)

Με την έλευση των Μεγάλων Δεδομένων (Big Data), του AI και του ML, δημιουργήθηκε η ανάγκη διαχείρισης μεγάλων όγκων δεδομένων. Το να αποθηκεύει μία εταιρία τα δεδομένα της στο Νέφος είναι φτηνότερο από το να συντηρεί δικό της εξοπλισμό για να αποθηκεύει τα δεδομένα. Οι εταιρίες που προσφέρουν Αποθήκες Δεδομένων στο Νέφος συνήθως χρεώνουν τους πελάτες τους βάσει της χρήσης του αποθηκευτικού χώρου που χρησιμοποιούν ή ανάλογα την χρήση που κάνουν, επομένως είναι πιο οικονομικό από τις συμβατικές Αποθήκες Δεδομένων όπου η εταιρία πρέπει να συντηρεί το δικό της σύστημα 24/7. Επιπλέον, ένα πολύ σημαντικό πλεονέκτημα που κερδίζει μία εταιρία χρησιμοποιώντας την δομή αυτή, είναι η δυνατότητα που παρέχει η ΑΔ στο Νέφος να υποστηρίζει data streams, κάνοντας εφικτή την διαχείριση δεδομένων σε πραγματικό χρόνο (real-time), όπως για παράδειγμα η εκτέλεση ερωτημάτων (Queries) σε πραγματικό χρόνο, η τροφοδοσία μοντέλων Μηχανικής Μάθησης (Machine Learning - ML) με δεδομένα για εκπαίδευση σε πραγματικό χρόνο κ.α. [WP1]

Οι μεγαλύτερες εταιρείες στον χώρο έχουν εκδώσει τις δικές τους Αποθήκες Δεδομένων στο Νέφος, τέτοιες είναι οι Google BigQuery [WP24], Amazon Redshift [WP25] και Microsoft

Snowflake [WP26]. Όλες οι προαναφερθείσες ΑΔ στο Νέφος αποθηκεύουν τα δεδομένα τους σε μορφή τύπου στήλης (Columnar format) [WP27][WP28][WP29]. Αυτό δίνει την δυνατότητα να κάνει τέτοιου είδους ΑΔ ιδιαίτερα αποτελεσματικές σε ερωτήματα αναλύσεων κ.ά.

1.3.3 Row vs Columnar Databases

Όπως αναφέρθηκε και προηγουμένως, υπάρχουν δύο διαφορετικοί τρόποι με τους οποίους μπορούν οι εγγραφές μιας ΒΔ να αποθηκευτούν σε ένα αποθηκευτικό μέσο. Στην μορφή τύπου στήλης τα δεδομένα αποθηκεύονται στο αποθηκευτικό μέσο σε στήλες, ενώ στην μορφή τύπου γραμμής τα εγγραφές της ΒΔ αποθηκεύονται μία ανά σειρά.

Employees					
	Name	Country	Position	Salary	Overtime
Block 1	John	ENG	Software Engineer	2.000 €	2 hours
Block 2	Sebastian	SPA	Graphic Designer	1.600 €	10 hours
Block 3	Nikos	GRE	Software Engineer	2.000 €	6 hours

Row-oriented Database

Employees				
Block 1	Name	John	Sebastian	Nikos
Block 2	Country	ENG	SPA	GRE
Block 3	Position	Software Engineer	Graphic Designer	Software Engineer
Block 4	Salary	2.000 €	1.600 €	2.000 €
Block 5	Overtime	2 hours	10 hours	6 hours

Column-oriented Database

Εικόνα 2: Παράδειγμα σύγκρισης αποθήκευσης εγγραφών σε ΒΔ με μορφή τύπου γραμμής (Row-oriented DB) και με μορφή τύπου στήλης (Columnar-oriented DB).

Ανάλογα με την εφαρμογή που θα χρησιμοποιήσει τα δεδομένα, ο μηχανικός καλείται να επιλέξει ανάμεσα στις δύο αυτές μορφές. Ας υποθέσουμε ότι οι παραπάνω εγγραφές ανήκουν στο προσωπικό μιας εταιρείας. Αν για παράδειγμα το λογιστικό τμήμα της εταιρείας θέλει κάθε μήνα να ενημερώνει την καρτέλα κάθε υπαλλήλου με τις υπερωρίες που έκανε κάθε μήνα και να υπολογίζει τα επιπλέον χρήματα που πρέπει να λάβει ο υπάλληλος ανάλογα με την

μισθολογική του κλίμακα και την θέση του, τότε αυτό σημαίνει ότι η εφαρμογή θα πρέπει να κάνει αυτή την διαδικασία ξεχωριστά για κάθε υπάλληλο και πιθανόν η αποθήκευση στο μέσο σε μορφή γραμμής θα ήταν μία λύση, μιας και χρειαζόμαστε τρία πεδία από κάθε εγγραφή και συγκεκριμένα τα πεδία Position, Salary, Overtime.

Αν όμως για παράδειγμα το λογιστήριο της εταιρείας ήθελε να υπολογίσει το άθροισμα των χρημάτων που δίνει κάθε μήνα σε όλους τους υπαλλήλους της εταιρείας χωρίς να λάβει υπόψη τις υπερωρίες, τότε δεν θα ενδιαφερόταν για άλλες πληροφορίες εκτός από τον μισθό των υπαλλήλων. Έτσι, διαβάζοντας μόνο ένα Block, το Block 4 σε μια columnar-oriented ΒΔ μπορεί κανείς να λάβει τους μισθούς όλων των υπαλλήλων, ενώ σε μια row-oriented ΒΔ θα χρειαζόταν να διαβάσει κάθε Block (ένα για κάθε εγγραφή (Υπάλληλο)).

1.4 ETL

Από το παράδειγμα της εικόνας 1 μπορεί κανείς να διακρίνει ένα συγκεκριμένο μοτίβο. Οι περισσότεροι σωληναγωγοί δεδομένων είναι σχεδιασμένοι βάσει το συγκεκριμένο μοτίβο, το οποίο ονομάζεται ETL (Extract Transform Load). Ουσιαστικά, το μοτίβο αυτό περιγράφει μία διαδικασία, η οποία ακολουθείται για να μετατραπούν ανεπεξέργαστα δεδομένα (Raw Data), σε δεδομένα τα οποία θα είναι με κάποιον τρόπο χρήσιμα. Σύμφωνα με [4] *«Οι διεργασίες ETL (Extraction-Transformation-Loading) ευθύνονται για την εξαγωγή των δεδομένων από ετερογενείς λειτουργικές πηγές δεδομένων, τον μετασχηματισμό των δεδομένων (μετατροπές, καθαρισμός δεδομένων, κανονικοποίηση, κλπ.) και την φόρτωση τους σε Data Warehouses (DWs).»* Συγκεκριμένα, το ETL περιγράφεται από τις παρακάτω λειτουργίες:

1.4.1 Εξαγωγή (Extract)

Η εξαγωγή των δεδομένων είναι το πρώτο βήμα κατά το ETL. Σκοπός της λειτουργίας αυτής είναι οι εξαγωγή δεδομένων από διάφορες πηγές δεδομένων. Είναι πολύ σημαντικό η εξαγωγή των δεδομένων από τις διάφορες πηγές δεδομένων να γίνει ανώδυνα, χωρίς να καθυστερείται η πηγή του συστήματος παραπάνω από όσο πρέπει. *«Αυτή η διαδικασία πρέπει να ενσωματώνει αποτελεσματικά διαφορετικούς τύπους συστημάτων που έχουν για διαφορετικά συστήματα βάσεων δεδομένων, διαφορετικές πλατφόρμες, διαφορετικά λειτουργικά συστήματα και διαφορετικά επικοινωνιακά πρωτόκολλα.»*[3] Αφού γίνει η εξαγωγή των δεδομένων από τις πηγές, τα δεδομένα οδηγούνται στο επόμενο βήμα που είναι ο μετασχηματισμός (Transformation) των δεδομένων.

1.4.2 Μετασχηματισμός (Transform)

Ο μετασχηματισμός δεδομένων είναι το σημαντικότερο βήμα στο ETL. Στην πραγματικότητα εδώ πραγματοποιείται η απαραίτητη επεξεργασία των δεδομένων και η επικύρωσή τους (Validation). Το μειονέκτημα όταν υπάρχουν πολλές διαφορετικές πηγές δεδομένων είναι ότι τα δεδομένα δεν είναι πάντα στην μορφή που τα καθιστά χρήσιμα και για αυτό πρέπει να μετασχηματιστούν. Άλλα πιθανά προβλήματα είναι:

1. Λάθος διαμόρφωση δεδομένων λόγω διαφορετικής τοποθεσίας (π.χ. Νόμισμα, Μορφοποίηση ώρας/ημερομηνίας, μεγεθών(μέτρα, πόδια) κ.α.
2. Λάθος κωδικοποίηση χαρακτήρων (π.χ. umlaut στα γερμανικά - ä, ö, ü)
3. Αντικατάσταση της υποδιαστολή (,) σε δεκαδικούς αριθμούς με τελεία (Από 3,14 σε 3.14)

Προφανώς, τα παραπάνω προβλήματα δεν είναι τα μόνα. Πρέπει πάντα ο σχεδιαστής του σωληναγωγού δεδομένων να περιμένει ότι τα δεδομένα που θα λάβει από τις πηγές δεδομένων θα έχουν πιθανά προβλήματα επικύρωσης. Αφού τα δεδομένα επικυρωθούν περνάνε στο επόμενο στάδιο, αυτό του μετασχηματισμού.

Τέτοιοι μετασχηματισμοί για παράδειγμα είναι:

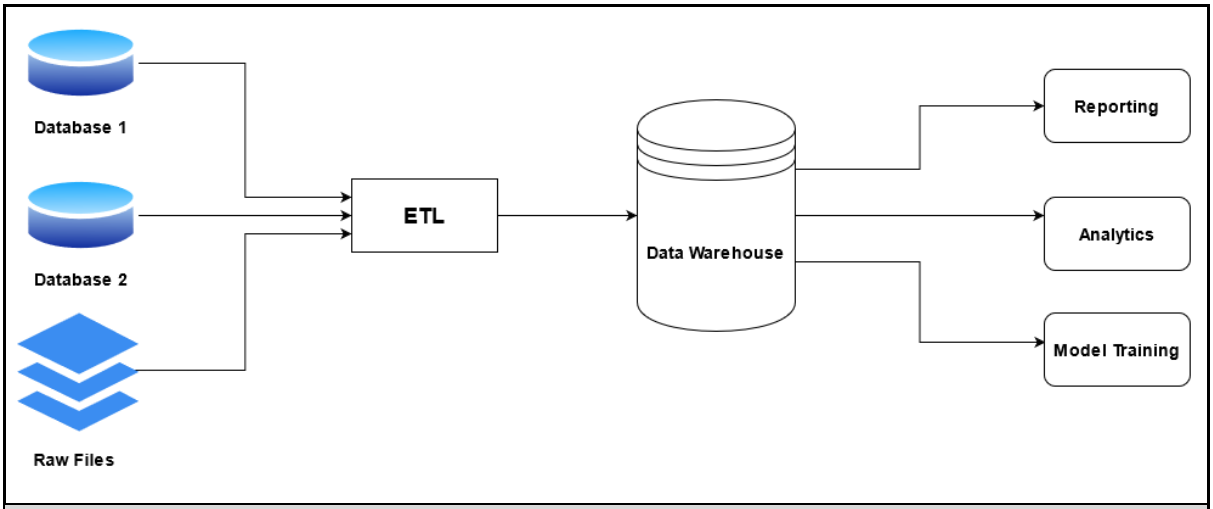
1. Απόρριψη των δεδομένων που δεν χρειάζονται
2. Διάσπαση κάποιων πεδίων σε μικρότερα (π.χ. εάν υπάρχει πεδίο address με διεύθυνση example street 2 ZIP CODE 12345, να σπάσει σε μικρότερα πεδία όπως Street name, Street number, ZIP code)
3. Ομαδοποίηση δεδομένων που ανήκουν σε διαφορετικές πηγές δεδομένων

Γενικά, οι μετασχηματισμοί έχουν σκοπό να μετατρέψουν τα ανεπεξέργαστο σετ δεδομένων (Raw Dataset) σε μία μορφή, η οποία τα καθιστά έτοιμα προς κατανάλωση.

1.4.3 Φόρτωση (Load)

«Όταν τα δεδομένα εξαχθούν και μετασχηματιστούν σύμφωνα με τις απαιτήσεις του Data Warehouse που προορίζονται, τα δεδομένα υποθετικά είναι έτοιμα να φορτωθούν. Ωστόσο, πρέπει να μελετηθούν διάφορες πτυχές πριν τα δεδομένα φορτωθούν στο Data Warehouse, όπως

τον τρόπο με τον οποίο τα δεδομένα θα φορτωθούν, αλλά και τις επιπτώσεις που μπορεί να έχει αυτό καθώς και πως πρέπει να διαχειριστούν αυτές οι επιπτώσεις πριν φορτωθούν στο Data Warehouse.» [5] Ο προορισμός των δεδομένων, βέβαια, δεν είναι πάντα μία δομή δεδομένων, θα μπορούσε να ήταν για παράδειγμα ένας άλλος σωληναγωγός.



Εικόνα 3: Παράδειγμα ETL και φόρτωσης των δεδομένων σε ένα DW. Διάφορες δραστηριότητες χρησιμοποιούν τα μοντελοποιημένα δεδομένα.

1.5 ELT

Το ELT είναι μία παραλλαγή του ETL. Κατ' αυτή την έννοια οι ενέργειες του μετασχηματισμού (**Transformation**) και της φόρτωσης (**Load**) αντιμετατίθενται. Το ELT *"επιτρέπει πρώτα την εξαγωγή και φόρτωση των δεδομένων, και ύστερα εφαρμόζει τον μετασχηματισμό των δεδομένων σύμφωνα με τις επιχειρησιακές ανάγκες."*[8] Αυτό είναι ιδιαιτέρως χρήσιμο όταν ο σωληναγωγός δεδομένων πρέπει να διαχειριστεί πολύ μεγάλο όγκο δεδομένων (Big Data). Στο ELT αφού τα ανεπεξέργαστα δεδομένα εξαχθούν από τις πηγές δεδομένων αποθηκεύονται σε μία δομή που ονομάζεται **Λίμνη Δεδομένων (Data Lake)**. Σύμφωνα με [7] "*Οι **Λίμνες Δεδομένων** κατασκευάζονται συνήθως με σκοπό να διαχειριστούν μεγάλους όγκους δεδομένων και αδόμητα δεδομένα που φορτώνονται γρήγορα στο μέσο (σε αντίθεση με τα δεδομένα που βρίσκονται στις Αποθήκες Δεδομένων, οι οποίες περιέχουν δομημένα δεδομένα) από τα οποία μπορούν να αντληθούν περαιτέρω πληροφορίες . Συνεπώς, οι Λίμνες Δεδομένων χρησιμοποιούν δυναμικές εφαρμογές ανάλυσης. Τα δεδομένα σε μια Λίμνη Δεδομένων είναι άμεσα διαθέσιμα με το που δημιουργούνται.*"

Characteristics	Data Warehouse	Data Lake
Data	Relational from transactional systems, operational databases, and line of business applications	Non-relational and relational from IoT devices, web sites, mobile apps, social media, and corporate applications
Schema	Designed prior to the DW implementation (schema-on-write)	Written at the time of analysis (schema-on-read)
Price/Performance	Fastest query results using higher cost storage	Query results getting faster using low-cost storage
Data Quality	Highly curated data that serves as the central version of the truth	Any data that may or may not be curated (ie. raw data)
Users	Business analysts	Data scientists, Data developers, and Business analysts (using curated data)
Analytics	Batch reporting, BI and visualizations	Machine Learning, Predictive analytics, data discovery and profiling

Εικόνα 4: Σύγκριση μεταξύ μιας Λίμνης Δεδομένων και μιας Αποθήκης Δεδομένων. [WP41]

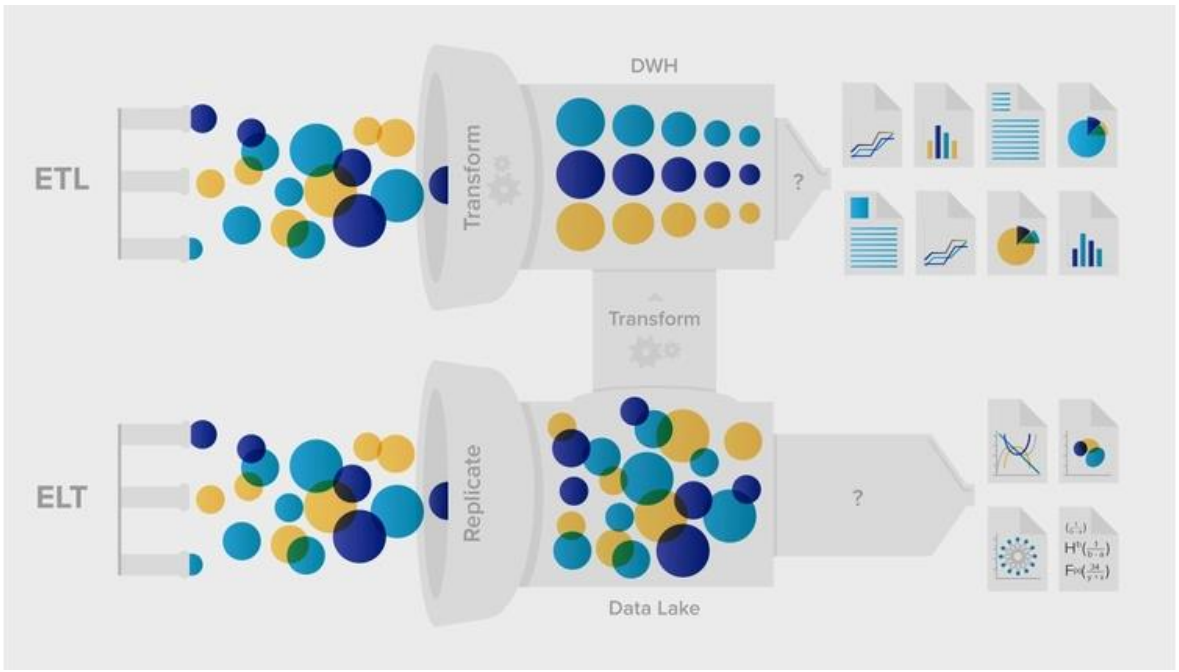
1.6 ELT vs ETL

Συνοψίζοντας, το ELT και ETL είναι μοτίβα τα οποία υλοποιούνται για να μεταφέρουν δεδομένα από ένα σύνολο πηγών σε κάποιον προορισμό, ο οποίος συνήθως είναι μια Αποθήκη Δεδομένων. Παρότι που για πάρα πολύ καιρό το ETL ήταν ο δημοφιλέστερος τρόπος σχεδιασμού σωληναγωγών δεδομένων, πλέον το ELT προτιμάτε έναντι αυτού. Μερικοί σημαντικοί λόγοι που συμβαίνει αυτό είναι:

1. **Analytics:** Αυξήθηκε η ζήτηση σε Machine Learning, Data Mining και λοιπές εφαρμογές, οι οποίες καταναλώνουν μεγάλο πλήθος δεδομένων.
2. **Μέγεθος δεδομένων:** το ELT αποθηκεύει τα δεδομένα σε υποδομές νέφους (Cloud Infrastructure), το οποίο το καθιστά καταλληλότερο για στην διαχείριση μεγάλων όγκων δεδομένων.
3. **Υψηλή διαθεσιμότητα δεδομένων:** Τα δεδομένα είναι διαθέσιμα άμεσα αφού φορτώνονται κατευθείαν από την πηγή στον προορισμό χωρίς να χρειάζεται να επεξεργαστούν πρώτα
4. **Ευελιξία:** Δεν χρειάζεται να αποφασίσει κανείς εξ' αρχής τι θα κάνει με τα δεδομένα αυτά. Μπορεί έπειτα αφού φορτωθούν στο μέσο και ανάλογα την περίσταση να κάνει τους κατάλληλους μετασχηματισμούς δεδομένων.

Σήμερα, η πλειοψηφία των ΑΔ βασίζεται σε columnar-oriented ΒΔ, οι οποίες μπορούν να αποθηκεύσουν, να διαχειριστούν και να μετασχηματίσουν μεγάλες δομές δεδομένων με ευκολία.(James Densmore, 2021) [1, p.23] Έτσι, μεγάλοι όγκοι δεδομένων μπορούν να

αποθηκευτούν απευθείας στο μέσο και να μετασχηματιστούν όποτε υπάρξει αυτή η ανάγκη (βλέπε εικόνα 5).



Εικόνα 5: Αλληλεπίδραση μεταξύ ELT και ETL. [WP2]

1.7 Data Ingestion

Η διαδικασία εξαγωγής των ανεπεξέργαστων δεδομένων από τις πηγές δεδομένων και η φόρτωση αυτών σε μια δομή δεδομένων, λέγεται **Data Ingestion**.

Στην αγορά υπάρχουν διάφορα εργαλεία με τα οποία μπορεί κανείς να κάνει Data Ingestion. Ένα από αυτά για παράδειγμα είναι το **Singer** [WP51]. Το **Singer** είναι ένα ανοικτού κώδικα ETL εργαλείο με το οποίο μπορεί κάποιος να εξάγει ανεπεξέργαστα δεδομένα από μία πηγή και να τα φορτώσει για παράδειγμα σε ένα αρχείο CSV, μία βάση δεδομένων, αρχείο Google Sheet κ.ά. Το **Singer** αποκαλεί τα Scripts που εξάγουν δεδομένα από την πηγή ως **taps** και τα Scripts που φορτώνουν δεδομένα σε διάφορες μορφές αρχείων ή δομών, **targets** και υποστηρίζει ήδη taps που εξάγουν δεδομένα από διάφορες πηγές όπως Google Analytics, Jira, Facebook Ad, PostgresSQL, Exchange Rates API κ.α.

Παρακάτω παρουσιάζεται ένα παράδειγμα από την επίσημη σελίδα [WP51] του Singer κατά το οποίο εξάγονται δεδομένα από ένα tap του Singer, το exchangerateapi και φορτώνονται σ' ένα target το οποίο έχει την δομή csv.

```
› pip install target-csv tap-exchangeratesapi
› tap-exchangeratesapi | target-csv

INFO Replicating the latest exchange rate data from exchangeratesapi.io
INFO Tap exiting normally

› cat exchange_rate.csv

AUD,BGN,BRL,CAD,CHF,CNY,CZK,DKK,GBP,HKD,HRK,HUF,IDR,ILS,INR,JPY,KRW,MXN,MYR
,NOK,NZD,PHP,PLN,RON,RUB,SEK,SGD,THB,TRY,ZAR,EUR,USD,date

1.3023,1.8435,3.0889,1.3109,1.0038,6.869,25.47,7.0076,0.79652,7.7614,7.0011
,290.88,13317.0,3.6988,66.608,112.21,1129.4,19.694,4.4405,8.3292,1.3867,50.
198,4.0632,4.2577,58.105,8.9724,1.4037,34.882,3.581,12.915,0.9426,1.0,2017-
02-24T00:00:00Z
```

Το να χρησιμοποιεί κανείς frameworks όπως είναι το Singer για να κάνει Data Ingestion είναι μεν ιδανικό όταν έχουμε να κάνουμε με δημοφιλές πηγές δεδομένων, ιδίως για άτομα που συνήθως έχουν ελάχιστες γνώσεις προγραμματισμού, όπως οι Αναλυτές Δεδομένων, αλλά παρόλα αυτά μερικές φορές χρειάζεται ο μηχανικός δεδομένων να κάνει μόνος του αυτή την δουλειά.

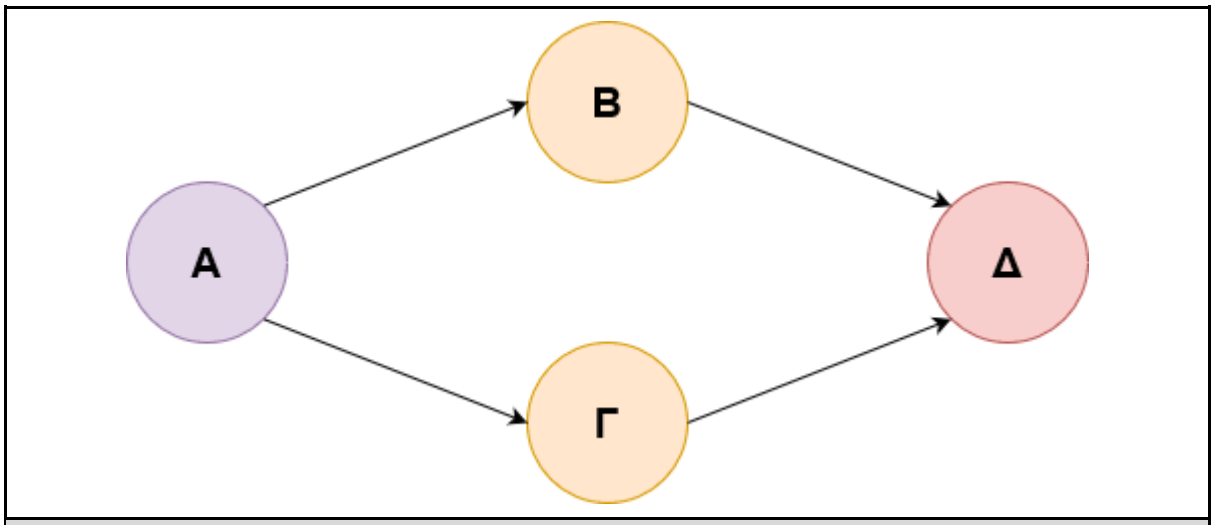
Το να εξάγει κανείς δεδομένα μόνος του χωρίς την χρήση κάποιου framework όπως το Singer δίνει την πολυτέλεια στον χρήστη να επεξεργαστεί τα δεδομένα (αν θέλει) προτού τα αποθηκεύσει στην δομή. Ωστόσο όπως αναφέρθηκε και προηγουμένως αυτό δεν είναι απαραίτητο να γίνει κατά την εξαγωγή των δεδομένων αλλά μπορεί να γίνει και σε μετέπειτα στάδιο.

1.8 Κατευθυνόμενοι Άκυκλοι Γράφοι (DAGs)

Ένας σωληναγωγός δεδομένων αναπαρίσταται με Άκυκλους Κατευθυνόμενους Γράφους (**Directed Acyclic Graphs - DAGs**). *"Όταν ένα Διάγραμμα Ροών Εργασιών περιγράφεται ως ένας Κατευθυνόμενος Άκυκλος Γράφος (ΚΑΓ - DAG), οι κόμβοι του γράφου αναπαριστούν τις διεργασίες του Διαγράμματος Ροής Διεργασιών και οι ακμές αναπαριστούν τις αλληλο-εξαρτήσεις μεταξύ τους."* [9]. Επιπλέον, οι Γράφοι αυτοί λέγονται άκυκλοι διότι όταν η διεργασία που συμβολίζεται από έναν κόμβο του γράφου εκτελεστεί μία φορά, ο Άκυκλος Κατευθυνόμενος Γράφος είναι σχεδιασμένος έτσι ώστε να μην υπάρξει μελλοντικό μονοπάτι

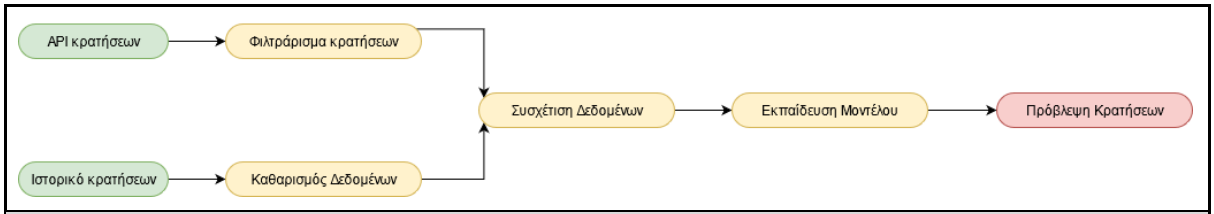
που να ξανά καταλήγει σε αυτόν τον κόμβο. Συνεπώς, κάθε διεργασία εκτελείται ακριβώς μία φορά.

Στο παράδειγμα της **Εικόνας 6** παρουσιάζεται μια Ροή Διεργασιών, η οποία απεικονίζεται χρησιμοποιώντας έναν Κατευθυνόμενο Άκυκλο Γράφο. Η διεργασία Α είναι η πρώτη που εκτελείται, ενώ οι Β και Γ "τρέχουν" παράλληλα. Για να ξεκινήσει η εκτέλεση όμως των διεργασιών Β και Γ, πρέπει οπωσδήποτε να έχει ολοκληρωθεί η διεργασία Α. Το ίδιο ισχύει και για την διεργασία Δ. Για να εκτελεστεί η διεργασία Δ, θα πρέπει να έχουν ολοκληρωθεί οι διεργασίες Β και Γ. Χρησιμοποιώντας, λοιπόν, τις συμβάσεις αυτές μπορούμε να σχεδιάσουμε το σύνολο των διεργασιών ενός σωληναγωγού δεδομένων, ο οποίος προορίζεται για να εκπαιδεύσει για παράδειγμα ένα μοντέλο μηχανικής μάθησης.



Εικόνα 6: Παράδειγμα ενός απλού Διαγράμματος Ροής Εργασιών που απεικονίζεται με Κατευθυνόμενο Άκυκλο Γράφο.

Στο παρακάτω παράδειγμα, μία εταιρία ενοικίασης αυτοκινήτων θέλει να προβλέψει τα πιθανά έσοδα που θα έχει το ερχόμενο καλοκαίρι βάσει των κρατήσεων αεροπορικών εισιτηρίων.



Εικόνα 7: Αναπαράσταση σωληναγωγού με DAG που προβλέπει το σύνολο κρατήσεων.

Στο παράδειγμα της Εικόνας 7 υπάρχουν δύο πηγές δεδομένων. Η πρώτη είναι ένα API, από το οποίο λαμβάνονται πληροφορίες σχετικά με τις κρατήσεις αεροπορικών και ακτοπλοϊκών εισιτηρίων, ενώ η δεύτερη πηγή δεδομένων παρέχει πληροφορίες σχετικά με το ιστορικό ενοικιάσεων οχημάτων τα περασμένα χρόνια και είναι αποθηκευμένες σε μία Βάση Δεδομένων.

Ας υποθέσουμε ότι μία εγγραφή από τα δεδομένα των κρατήσεων των ακτοπλοϊκών εισιτηρίων είναι όπως παρακάτω:

Όνομα	**Επίθετο**	**Τόπος Αναχώρησης**	**Προορισμός**	**Ημερομηνία Αναχώρησης**
Γιάννης	Παπαδόπουλος	Πειραιάς	Σύρος	14/09/2021

Ενώ μία εγγραφή από τα δεδομένα του ιστορικού ενοικιάσεων είναι όπως παρακάτω:

Όνομα	**Επίθετο**	**Όχημα**	**Ημερομηνία ενοικίασης**
Δημήτρης	Χατζής	Chevrolet Aveo	01/09/2019

Αφού ληφθούν τα δεδομένα ακολουθούν κάποιες διαδικασίες με τις οποίες κρατιούνται από αυτά μόνο οι απαραίτητες πληροφορίες. Από τις κρατήσεις εισιτηρίων κρατούνται μόνο αυτές που συμπίπτουν στην χρονική περίοδο και τον τόπο για τον οποίο θα γίνει η πρόβλεψη (Φιλτράρισμα), ενώ από το ιστορικό κρατήσεων κρατούνται μόνο οι πληροφορίες σχετικά με τις ενοικιάσεις που έγιναν στο παρελθόν μία συγκεκριμένη περίοδο και συγκεκριμένα διαμορφώνονται τα δεδομένα έτσι ώστε να αποκόπτονται οι πληροφορίες που δεν είναι χρήσιμες και να κρατούνται μόνο οι χρήσιμες. Στη συγκεκριμένη περίπτωση κρατούνται μόνο οι ημερομηνίες ενοικίασης από το ιστορικό κρατήσεων και το όχημα, μιας και τα υπόλοιπα στοιχεία δεν μας ενδιαφέρουν (Π.χ. Όνομα και Επίθετο).
Έπειτα, όταν τα δεδομένα είναι στην απαιτούμενη μορφή, τα δύο datasets συσχετίζονται μεταξύ τους και στην συνέχεια τροφοδοτούν τον μοντέλο MM με το πλήθος των δεδομένων,

έτσι ώστε να εκπαιδεύσουν το μοντέλο. Τέλος, αφού εκπαιδευτεί το μοντέλο γίνεται η πρόβλεψη πωλήσεων.

Όπως φαίνεται στην Εικόνα 7, η επεξεργασία δεδομένων των δύο διαφορετικών Datasets γίνεται παράλληλα. Αυτό είναι εφικτό επειδή οι δύο αυτές διαδικασίες δεν έχουν κάποια αλληλεξάρτηση.

Κεφάλαιο 2: Συστήματα Διαχείρισης Ροών Εργασιών (WfMS)

2.1 Εισαγωγή

Τα Συστήματα Διαχείρισης Ροών Διεργασιών (**Workflow Management Systems - WfMS**) ή αλλιώς Πλατφόρμες Ενορχήστρωσης Διαγραμμάτων Ροής Εργασιών (**Workflow Orchestration Platforms**) παρέχουν μία υποδομή, η οποία βοηθάει τον χρήστη να σχεδιάσει, να οργανώσει, να προγραμματίσει και να επιβλέψει μία ροή διεργασιών. Συνήθως, τα ΣΔΡΕ δίνουν την δυνατότητα στον χρήστη να σχεδιάσει σωληναγωγούς δεδομένων ή γενικά μια ροή διεργασιών (**Workflows**).

Μία ροή διεργασιών αποτελείται από τις διεργασίες που είναι απαραίτητες ώστε να ολοκληρωθεί μία εργασία. Τέτοιες διεργασίες θα μπορούσαν να είναι για παράδειγμα η λήψη δεδομένων από μία ΒΔ, η αποστολή ενός email μέσω του ηλεκτρονικού ταχυδρομείου, ο μετασχηματισμός δεδομένων μέσω μιας γλώσσας προγραμματισμού όπως η Java [WP48] κ.α. Σε μία ροή εργασίας, πολλές διεργασίες μπορούν να εκτελούνται παράλληλα αν η μία δεν εξαρτάται από την άλλη.

Ένα διάγραμμα ροής δεδομένων αναπαρίσταται με έναν Κατευθυνόμενο Γράφο (**DG**), του οποίου οι κόμβοι αναπαριστούν διεργασίες, οι οποίες πρέπει να εκτελεστούν, ενώ οι ακμές προσδιορίζουν την φορά που ακολουθεί η εκτέλεση του ΔΕ και συνεπώς και τις εξαρτήσεις μεταξύ των διεργασιών. Η διαφορά μεταξύ των DAG και των DG είναι ότι τα δεύτερα μπορούν να είναι κυκλικά. Αυτό σημαίνει ότι, εάν ένας κόμβος έχει ήδη επισκεφθεί δεν σημαίνει ότι στο μέλλον η ροή του ΔΕ δεν θα ξανά περάσει από τον ίδιο κόμβο.

Κάθε Πλατφόρμα Ενορχήστρωσης Ροών Διεργασιών έχει τον δικό της τρόπο με τον οποίο ο χρήστης καλείται να σχεδιάσει τον ΑΚΓ που περιγράφει το ΔΡΕ. Οι δημοφιλέστεροι τρόποι καθορισμού ΔΡΕ περιλαμβάνουν την γλώσσα προγραμματισμού Python [WP45], την XML [WP46], την JSON [WP47] και την YAML. Ανάλογα την πλατφόρμα που θα χρησιμοποιήσει ο Μηχανικός Δεδομένων θα πρέπει να χρησιμοποιήσει και τον αντίστοιχο τρόπο.

Για παράδειγμα στην πλατφόρμα Azkaban [WP3] του LinkedIn τα ΔΡΕ ορίζονται χρησιμοποιώντας YAML [WP4]. Το YAML σύμφωνα με [WP43] είναι μία γλώσσα μετατροπής δεδομένων παραλληλίας-σειράς (**data serialization**), η οποία έχει κατασκευαστεί με τέτοιο τρόπο ώστε διαβάζεται και γράφεται εύκολα από ανθρώπους. Επιπλέον, επειδή η YAML είναι υπερσύνολο της JSON, χρησιμοποιεί παρόμοια δομή για να περιγράψει π.χ. λίστες, maps, κ.ά με αυτή της JSON, ενώ η χρήση του newline και των κενών χαρακτήρων είναι ιδιαιτέρως σημαντική όπως και στην Python.

```
nodes:
  - name: jobD
    type: command
    config:
      command: echo "This is the last node of the flow"
    dependsOn:
      - jobB
      - jobC

  - name: jobA
    type: command
    config:
      command: echo "This is the start of the flow."

  - name: jobB
    type: noop
    dependsOn:
      - jobA

  - name: jobC
    type: noop
    dependsOn:
      - jobA
```

Εικόνα 8: Ορισμός του DAG της Εικόνας 6 με YAML στο Azkaban.

Αντίστοιχα, στο Netflix Conductor [WP44] τα DAG περιγράφονται μέσω ενός αρχείου JSON, το οποίο περιέχει τις περιγραφές κάθε task.

```
{
  "name": "Workflow_Example_aa2a",
  "description": "A Simple workflow and task definition",
  "version": 1,
  "tasks": [
        {
        "name": "TaskA",
        "taskReferenceName": "TaskA",
        "type": "SIMPLE"
        },
        {
        "name": "fork_join",
        "taskReferenceName": "fork_join",
        "type": "FORK_JOIN",
        "forkTasks": [
        [
        {
        "name": "TaskB",
        "taskReferenceName": "TaskB",
        "type": "SIMPLE"
        }
        ],
        [
        {
        "name": "TaskC",
        "taskReferenceName": "TaskC",
        "type": "SIMPLE"
        }
        ]

        ]
        },
        {
        "name": "join",
        "taskReferenceName": "join2",
        "type": "JOIN",
        "joinOn": [
        "TaskB",
        "TaskC"
        ],
        "ownerEmail" : "admin@test.gr"
        },
```

```
        {
        "name": "TaskD",
        "taskReferenceName": "TaskD",
        "type": "SIMPLE",
        "inputParameters": {
        "fileLocation": [
        "${TaskB.output.TaskB}",
        "${TaskC.output.TaskC}"
        ]
        }

        }
 ],
 "ownerEmail": "admin@test.com"
}
```

Εικόνα 9: Ορισμός του DAG της εικόνας 6 με JSON στο Netflix Conductor.

2.2 Airflow

Το Apache Airflow [WP14] είναι μια open-source πλατφόρμα συγγραφής, προγραμματισμού και επίβλεψης ροών διεργασιών. Το Airflow ξεκίνησε τον Οκτώβρη του 2014 από τον Maxime Beauchemin που εργαζόταν στην Airbnb και στην συνέχεια υιοθετήθηκε από την Apache. Οι περισσότερες πληροφορίες που αναγράφονται στο κεφάλαιο 2.2 έχουν παρθεί από το επίσημο Documentation του Apache Airflow [WP31]

Στο Airflow μπορεί κανείς να περιγράψει μία ροή διεργασιών συγγράφοντας ένα DAG, όπου περιγράφει κάθε επιμέρους διεργασία που περιέχει η ροή διεργασιών και τις αλληλοεξαρτήσεις αυτών με κώδικα Python. Επιπλέον, το Airflow παρέχει έναν **Scheduler** με τον οποίο μπορεί κανείς να προγραμματίσει την ημερομηνία εκτέλεσης των διεργασιών και να μοιράσει τις διεργασίες σε **workers**, οι οποίοι εκτελούν τις διεργασίες σεβόμενοι της αλληλο-εξαρτήσεις τους. Κάποια προνόμια που παρέχει στον χρήστη το Apache Airflow είναι η πλούσια Διεπαφή Χρήστη **(User Interface)** που του παρέχεται μέσω του Web Server, με το οποίο μπορεί κανείς να παρακολουθήσει την πορεία της ροής των διεργασιών, να διαβάσει τα log files για να βρει την αιτία που κάποια διεργασία αποτυγχάνει, να ελέγξει την διάρκεια εκτέλεσης κάθε διεργασίας με σκοπό την βελτίωση της και πολλά ακόμα.

Επιπρόσθετα, το Apache Airflow συνδυάζει κάποια πολύ ιδιαίτερα χαρακτηριστικά που το καθιστούν ιδιαίτερα εύχρηστο και αποτελεσματικό. Ένα από αυτά είναι ότι ο χρήστης λόγω του ότι ορίζει τα DAGs χρησιμοποιώντας κώδικα σε Python, μπορεί να ορίσει **δυναμικά (Dynamically)** τις διεργασίες που θα περιέχει το DAG. Επιπλέον, το Airflow είναι **κλιμακούμενο (Scalable)**, δηλαδή μπορεί ο Μηχανικός Δεδομένων να ορίσει όσους Workers χρειάζεται για να εκτελέσει παράλληλα τις διεργασίες, πετυχαίνοντας έτσι την παράλληλη εκτέλεση των διεργασιών και συνεπώς πιο γρήγορη εκτέλεση των διεργασιών του DAG. Τέλος, ένα πολύ χρήσιμο χαρακτηριστικό του Apache Airflow είναι το XCOM, καθώς καθιστά δυνατή την ανταλλαγή δεδομένων μεταξύ των διεργασιών που περιέχονται σε ένα DAG. Ωστόσο, περιορίζεται στην ανταλλαγή μικρών όγκων δεδομένων.

2.2.1 Βασικά δομικά στοιχεία του Airflow

Στο Airflow η κάθε διεργασία που περιέχεται σε ένα DAG υλοποιείται με κάποιον Operator. Ανάλογα με το τι θέλει ο χρήστης να εκτελέσει η διεργασία, επιλέγει και τον κατάλληλο Operator. Μερικοί από αυτούς τους Operators είναι οι BashOperator, DockerOperator, EmailOperator, mySQLOperator κ.ά., ενώ ο χρήστης μπορεί να φτιάξει τον δικό του CustomOperator, πράγμα που καθιστά το Apache Airflow **επεκτάσιμο (Extensible).**

Στο παρακάτω παράδειγμα στην εικόνα 10 ένας χρήστης θέλει να πραγματοποιήσει ένα http GET request και στην συνέχεια ανάλογα με το αν λάβει ή όχι αποτελέσματα να τυπώσει με τον κατάλληλο Operator ένα μήνυμα στον χρήστη. Για την υλοποίηση θα χρειαστούν οι Operators SimpleHttpOperator, BashOperator, PythonOperator, DummyOperator, BranchPythonOperator.

Στο Α μέρος είναι ο ορισμός του DAG. Εδώ, προστίθενται χρήσιμες πληροφορίες όπως το μοναδικό ID του DAG, η ημερομηνία εκκίνησης του DAG, κάθε πότε θα εκτελείται το DAG κ.α. Στο Β μέρος είναι συναρτήσεις που εκτελούν οι διεργασίες που απαρτίζουν το DAG και είναι δηλωμένες στο Γ μέρος. Στο Γ μέρος δηλώνονται οι διεργασίες του DAG και ανάλογα το ποιός Operator έχει χρησιμοποιηθεί, πρέπει να συμπληρωθούν και οι αντίστοιχες μεταβλητές. Τέλος, στο Δ μέρος ορίζεται η σειρά εκτέλεσης των διεργασιών και οι αλληλοεξαρτήσεις τους.

```
dag = DAG(                                                                    A
    dag_id="pluggy_get_exh",
    start_date=days_ago(2),
    schedule_interval="@daily"
)

def branchIfStoriesNotFound(**context):                                       B

    request_res = json.loads(context['task_instance'].xcom_pull(task_ids='get_exhibition'))
    logging.info(request_res)
    if request_res['data']['total'] == 0:
        return "not_found"
    else:
        return "found"

def printExhibition(**context):

    request_res = context['task_instance'].xcom_pull(task_ids='get_exhibition')
    print('The following exhibitions were found: \n ' + request_res)

with dag:                                                                     Γ
    t1 = SimpleHttpOperator(
        task_id="get_exhibition",
        http_conn_id="pluggy_api",
        method="GET",
        endpoint= 'api/v1/exhibitions',
        headers={"Content-Type":"application/json"},
        data="tags=19-07-2021"
    )

branchingTask = BranchPythonOperator(task_id="branching", python_callable=branchIfStoriesNotFound)

t2 = BashOperator(
    task_id='not_found',
    bash_command='echo There are not any exhibitions available'
)

t3 = PythonOperator(task_id='found', python_callable=printExhibition)

complete = DummyOperator(task_id='complete', trigger_rule=TriggerRule.NONE_FAILED)

t1 >> branchingTask >> t2 >> complete                                         Δ
t1 >> branchingTask >> t3 >> complete
```

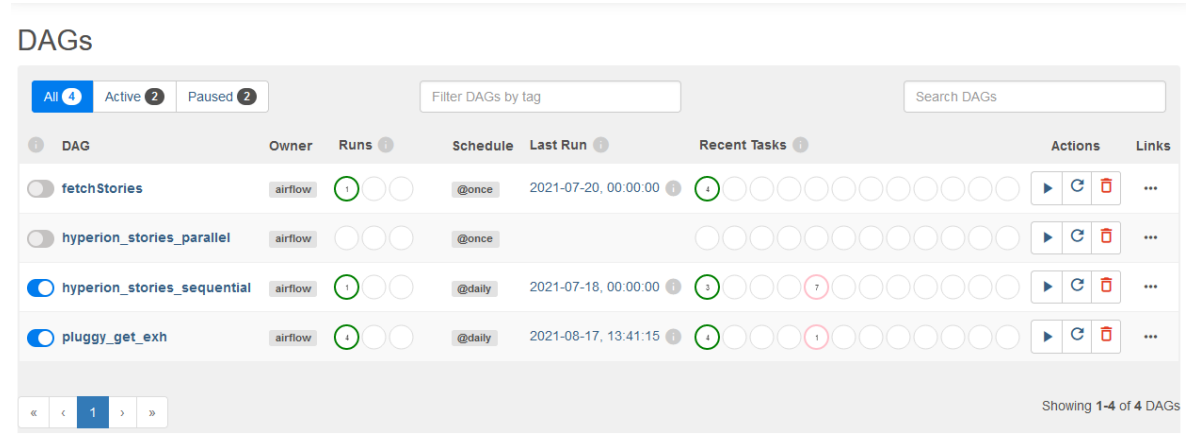
Εικόνα 10: Ορισμός ενός DAG στο Airflow.

2.2.2 Airflow web server και UI

Τα σημαντικότερα χαρακτηριστικά που προσφέρει το UI του Airflow είναι:

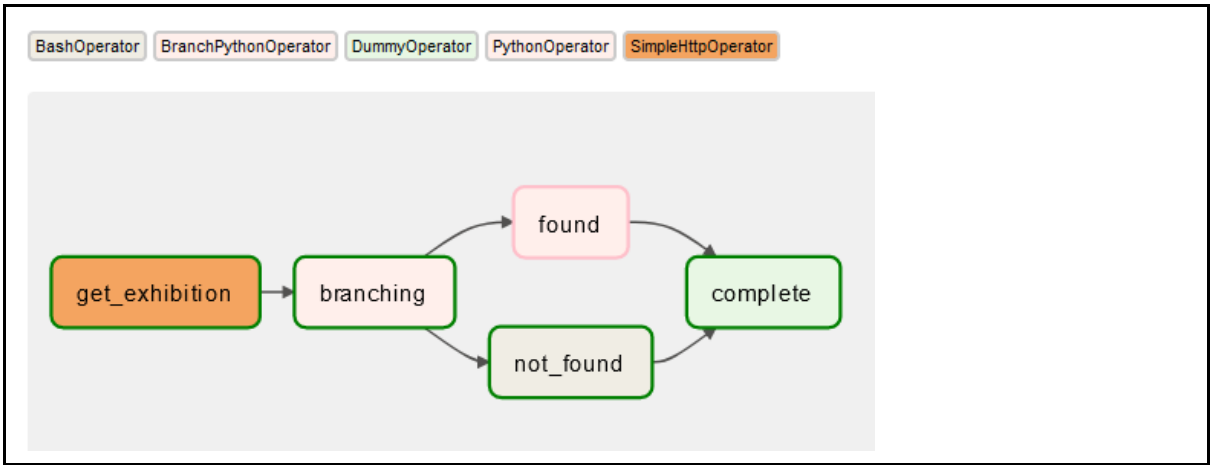
- Λίστα με ενεργά και μη ενεργά DAGs (Εικόνα 11)
- Δενδρική προβολή του ΔΡΔ και παρακολούθηση εκτέλεσης των επιμέρους διεργασιών (Εικόνα 12)

- Προβολή του διαγράμματος ροής διεργασιών σε μορφή γράφου και προβολή των αλληλεξαρτήσεων τους (Εικόνα 13)
- Παρακολούθηση της εκτέλεσης κάθε διεργασίας μέσω της καρτέλας Log (Εικόνα 14)
- Παρακολούθηση χρονικής διάρκειας κάθε διεργασίας (Εικόνα 15)



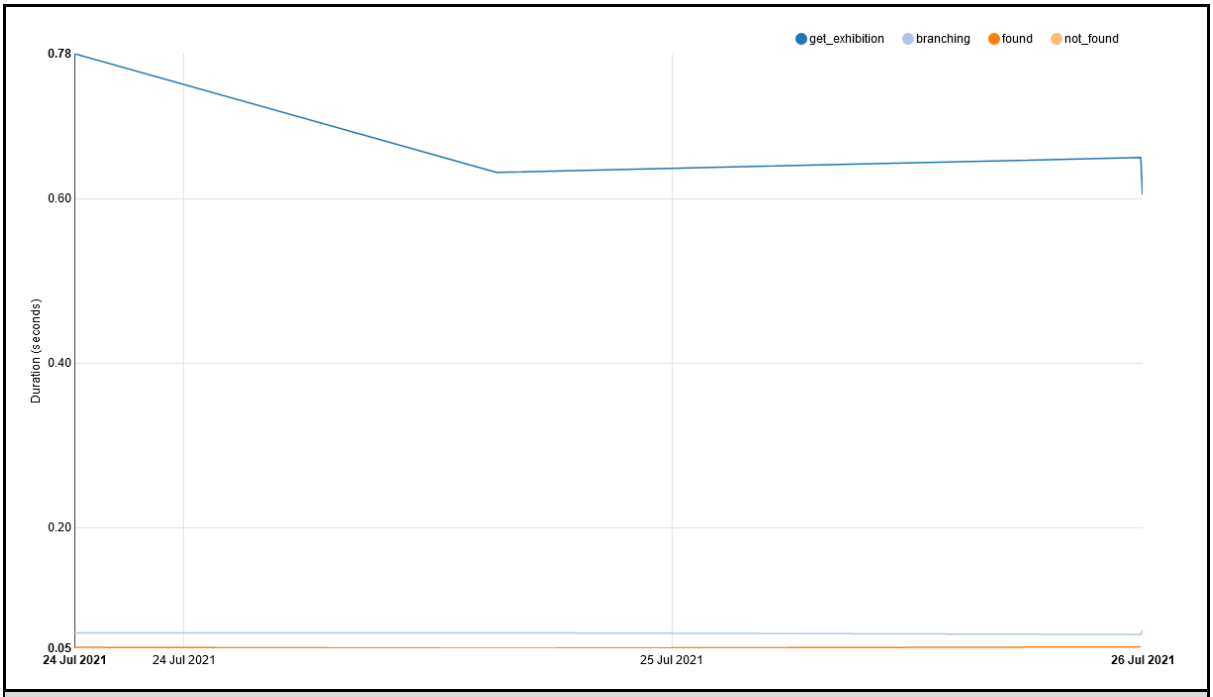
Εικόνα 11: Το UI του Airflow. Η λίστα με τα DAGs.

Εικόνα 12: Απεικόνιση ενός DAG σε δενδρική μορφή (αριστερά). Status κάθε διεργασίας (Δεξιά).



Εικόνα 13: Απεικόνιση τους ΔΡΑ σε μορφή γράφου.

Εικόνα 14: Παρακολούθηση του Log File της κάθε διεργασίας.

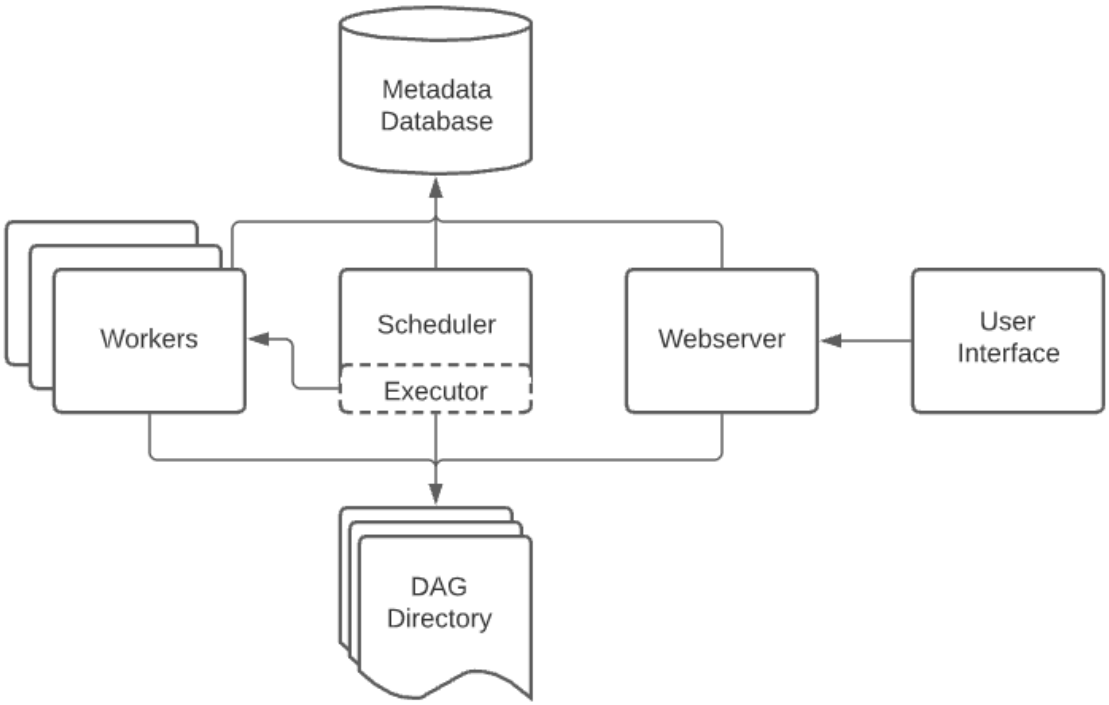


Εικόνα 15: Χρονική διάρκεια κάθε διεργασίας που μπορεί να δει κανείς στον Web Server.

2.2.3 Αρχιτεκτονική του Airflow

Εγκαθιστώντας το Airflow θα πρέπει να υπάρχουν σύμφωνα με [WP30] τουλάχιστον τα εξής:

- Ο **scheduler**, ο οποίος εκκινεί τα ΔΡΔ και αναθέτει διεργασίες στον **executor**.
- Ο **executor**, ο οποίος είναι υπεύθυνος για την εκτέλεση των **Tasks**. Ενώ μπορεί να ρυθμιστεί κατάλληλα για να αναθέτει τις διεργασίες προς εκτέλεση σε **workers** (π.χ. Celery Executors)
- Ο **Webserver** που όπως προαναφέρθηκε παρέχει στον χρήστη ένα χρήσιμο περιβάλλον για την παρακολούθηση των διεργασιών καθώς και πολλές ακόμα λειτουργίες.
- Ένας φάκελος που περιέχει τα DAGs σε αρχεία Python και ορίζεται κατά την εγκατάσταση από τον χρήστη.
- Μια βάση δεδομένων με metadata στην οποία ο scheduler, ο executor και ο web server αποθηκεύει τις καταστάσεις των διεργασιών.



Εικόνα 16: Η αρχιτεκτονική του Apache Airflow.[WP30]

Το Airflow έχει δύο είδη executor, αυτούς που τρέχουν τις διεργασίες τοπικά (**locally**) και αυτούς που τρέχουν τις διεργασίες εξ αποστάσεως (**remotely**), συνήθως με εξωτερικούς εργάτες (**Workers**). Στην προεπιλεγμένη εγκατάστασή του, το Airflow είναι ρυθμισμένο να λειτουργεί με τον **SequentialExecutor**, ο οποίος είναι ένας τοπικός executor και εκτελεί τις διεργασίες ενός DAG ακολουθιακά. [WP32] Αν ο χρήστης θέλει οι διεργασίες ενός DAG να τρέχουν παράλληλα σε τοπικό περιβάλλον, θα πρέπει να χρησιμοποιήσει τον Local executor, ο οποίος μπορεί να τρέξει παράλληλα και τοπικά διεργασίες δημιουργώντας πολλαπλά subprocesses (αντί για workers που χρησιμοποιούνται στους remote executors).

Οι remote executors που παρέχει το Airflow είναι οι CeleryExecutor και KubernetesExecutor. Με τον πρώτο μπορεί ο χρήστης να κλιμακώσει τον αριθμό των εργατών και έτσι δίνεται η δυνατότητα της παράλληλης εκτέλεσης διεργασιών, ενώ με τον KubernetesExecutor, δημιουργείται για κάθε στιγμιότυπο μιας διεργασίας ένα pod.

Σύμφωνα με [WP33] , *"Pods are the smallest deployable units of computing that you can create and manage in Kubernetes. A Pod (as in a pod of whales or pea pod) is a group of one or more containers, with shared storage and network resources, and a specification for how to run the containers."*
Οι τοπικοί executors και συγκεκριμένα ο LocalExecutor είναι ιδανικός όταν οι διαθέσιμοι πόροι δεν περιλαμβάνουν παραπάνω από ένα μηχάνημα. Ενώ οι remote executors είναι ιδανικοί όταν απαιτείται η εγκατάσταση πολλαπλών μηχανημάτων ή εγκατάσταση σε δομή νέφους (Cloud).

2.2.4 Εκτέλεση ενός ΔΡΔ στο Apache Airflow

Εφόσον έχει οριστεί κατάλληλα το DAG που περιγράφει την ροή κάποιον διεργασιών, πρέπει μέσω ενός τερματικού να εκτελεστούν οι εξής εντολές.

1. `airflow db init`, για να φορτωθεί στην βάση δεδομένων του Apache Airflow όλη η πληροφορίας σχετικά με τα διαθέσιμα DAGs. (Εάν χρησιμοποιείται LocalExecutor, τότε θα πρέπει να τεθεί σε λειτουργία και το service της Βάσης Δεδομένων. π.χ. `sudo service postgresql start`, σε περίπτωση που η βάση είναι postgreSQL [WP39])
2. `airflow scheduler`, για να ξεκινήσει ο Scheduler.
3. `airflow webserver`, για να ξεκινήσει ο Webserver

Στην συνέχεια μέσω του Web Server UI ο χρήστης μπορεί να πυροδοτήσει ένα workflow. Η ροή εκτέλεσης των διεργασιών φαίνεται στην εικόνα 13, ενώ το αποτέλεσμα της εκτέλεσης της διεργασίας φαίνεται στην εικόνα 14. Τέλος, το Apache Airflow προσφέρει ένα API από το οποίο μπορεί κανείς να πυροδοτήσει την εκκίνηση ενός DAG, να δει πληροφορίες για την κατάσταση των διεργασιών, να παύσει την εκτέλεση του DAG κ.α.

2.3 Luigi by Spotify

Το Luigi [WP15] είναι μία πλατφόρμα ενορχήστρωσης διεργασιών και έχει αναπτυχθεί από την εταιρεία Spotify. Είναι ιδανικό για να διαχειριστεί μεγάλους όγκους δεδομένων, Streaming δεδομένα, δεδομένα που εξάγονται από βάσεις δεδομένων κ.α. Επιπλέον, παρέχει στον χρήστη ένα γραφικό περιβάλλον στο οποίο μπορεί να παρακολουθήσει την ροή των διεργασιών, τις

αλληλοεξαρτήσεις τους και πολλές ακόμα λειτουργίες. Οι περισσότερες πληροφορίες που βρίσκονται στην ενότητα 2.3 και αφορούν το Luigi ανακτήθηκαν από το επίσημο documentation της πλατφόρμας που βρίσκεται εδώ. [WP15]

2.3.1 Δομικά στοιχεία του Luigi

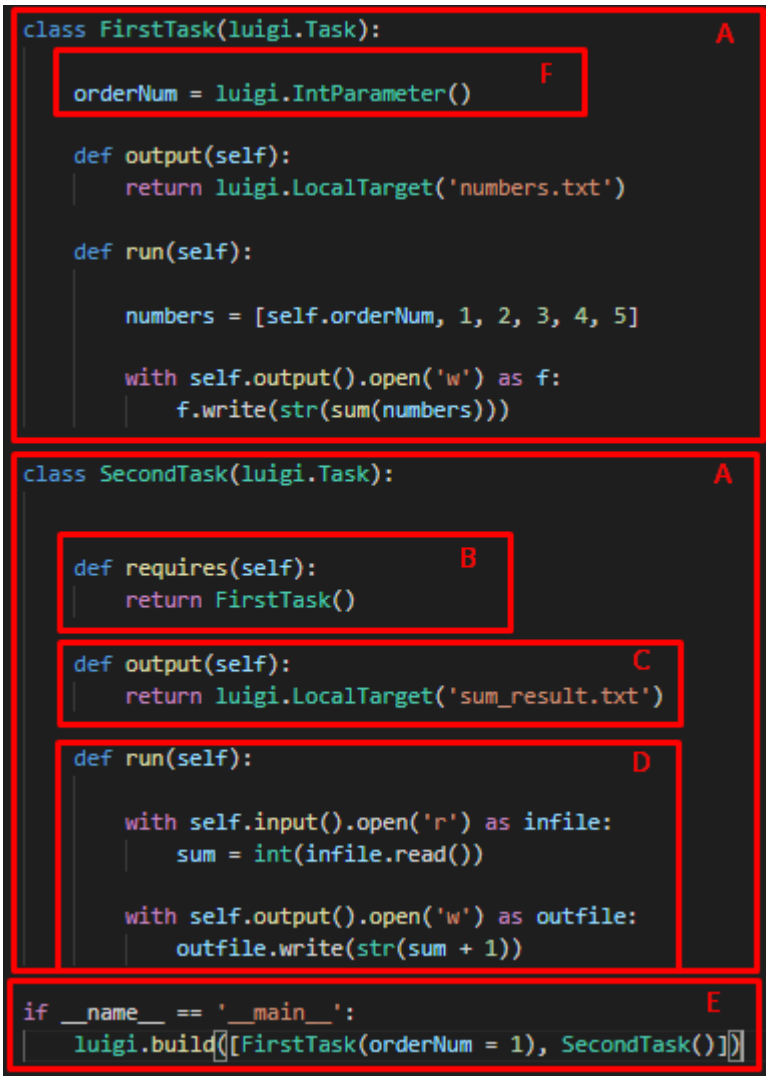
Στο **Luigi** [WP15] τα κύρια δομικά στοιχεία για την κατασκευή ενός **Workflow** είναι οι κλάσεις **Task** και **Target**. Οι κλάσεις **Task** και **Target** είναι Abstract κλάσεις και για αυτό θα πρέπει να υλοποιήσουν τις αντίστοιχες συναρτήσεις.

Η κλάση **Target** ουσιαστικά αντιστοιχεί σε κάποια δεδομένα. Αυτά μπορεί να είναι ένα αρχείο, μία βάση δεδομένων κ.α. Το **Luigi** έχει ήδη υλοποιημένες υποκλάσεις της **Target**, οι οποίες δίνουν πρόσβαση στις πιο δημοφιλές πηγές δεδομένων, όπως για παράδειγμα σε Amazon S3 [WP34], MySQL [WP35] κ.α.

Στην κλάση **Task** γίνεται η εκτέλεση της διεργασίας. Κάθε **Task** αντιστοιχεί σε μία διεργασία και συνήθως υλοποιεί τις μεθόδους **run()**, **requires()**, **input()** και **output()**. Στο σώμα της μεθόδου **run()** περιέχεται ο κώδικας που θα εκτελέσει η διεργασία, στην μέθοδο **requires()** ορίζονται οι εξαρτήσεις της διεργασίας με άλλες διεργασίες. Τέλος, η **input()** υποδεικνύει τις κατάλληλες **Target** κλάσεις σύμφωνα με της αλληλοεξαρτήσεις των δεδομένων που ορίζονται στην r**equires()**, ενώ η **output()** το που αποθηκεύονται τα δεδομένα της διεργασίας και τα αντιστοιχεί σε μία υποκλάση της **Target**.

Η κλάση **Parameter** χρησιμοποιείται για να περαστεί σε μία διεργασία μία μεταβλητή σαν όρισμα κατά την εκτέλεση της (βλέπε F στην εικόνα 17).
Στην εικόνα 17 παρουσιάζεται ο τρόπος με τον οποίο δηλώνεται ένα Workflow στο Luigi. Οι δύο κλάσεις που συμβολίζονται με A αντιστοιχούν στις διεργασίες που περιέχει το Workflow. Με B συμβολίζεται στην δεύτερη διεργασία η εξάρτηση που έχει με την πρώτη διεργασία. Το C υποδεικνύει την δομή στην οποία θα γραφτεί το output της διεργασίας. Στο D εκτελείται η κύρια λογική της διεργασίας. Στο F ορίζεται το περιεχόμενο μιας μεταβλητής, η οποία περνιέται σαν όρισμα στην διεργασία και την ορίζει ο χρήστης κατά την εκτέλεση. Τέλος, στο

Ε εκτελείται η μέθοδος build, η οποία δέχεται σαν όρισμα το σύνολο των διεργασιών του ΔΡΔ και τα ορίσματά τους.

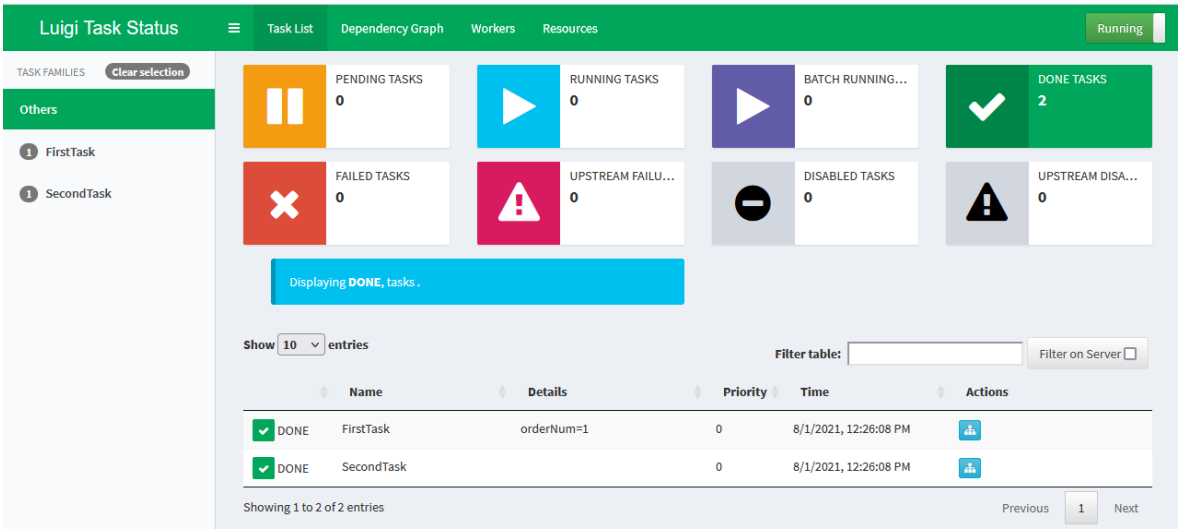


Εικόνα 17: Ορισμός ενός Workflow στο Luigi.

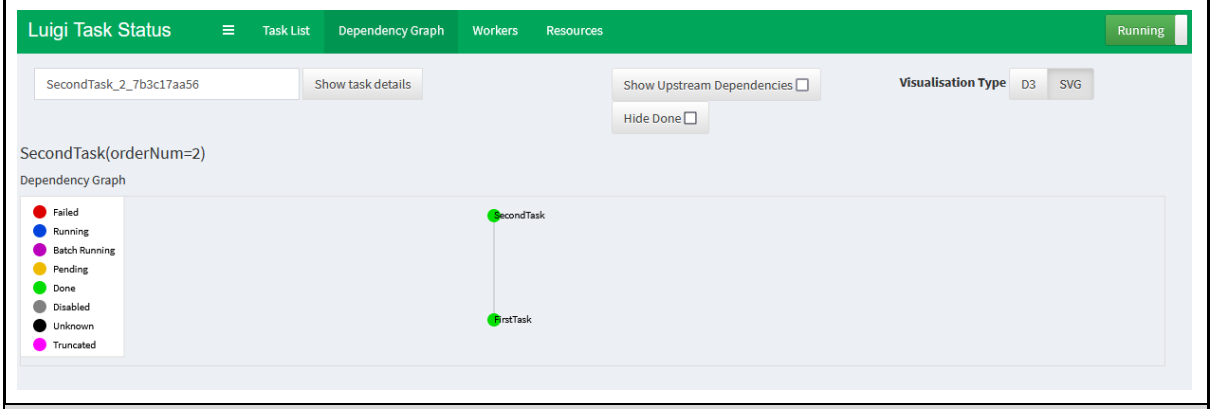
Οπότε, εκτελώντας το Διάγραμμα Ροής Διεργασιών του παραδείγματος της εικόνας 17 με παράμετρο orderNum = 2 ένα αρχείο .txt με το όνομα numbers.txt, το οποίο θα περιέχει το άθροισμα των orderNum, 2, 3, 4, 5. Στην συνέχεια θα παραχθεί το αρχείο sum_result.txt από την διεργασία SecondTask και θα έχει σαν αποτέλεσμα το άθροισμα του αριθμού που βρίσκεται στο αρχείο numbers.txt με τον αριθμό 1.

2.3.2 Web Server Interface

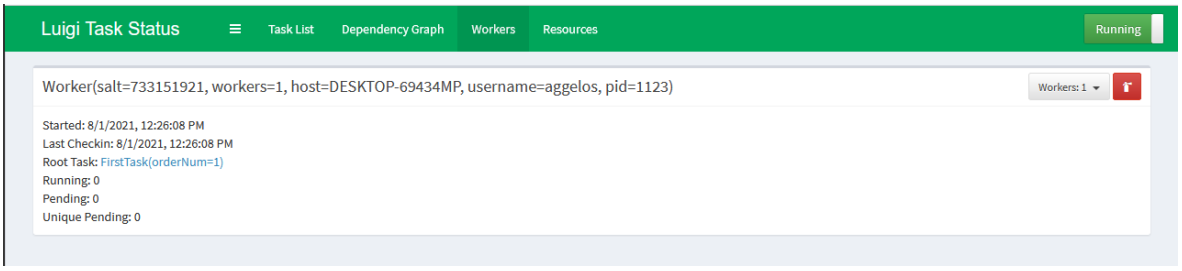
Το Luigi παρέχει έναν Web server, ο οποίος ονομάζεται Luigi Task Visualizer και δίνει την δυνατότητα στον χρήστη να παρακολουθεί μέσω αυτού την πορεία των διεργασιών και την κατάστασή τους. Το περιβάλλον αυτό αποτελείται από τέσσερα tabs, το Task List, το Dependency Graph, το Workers και το Resources. Στο Task List υπάρχει ένα βασικό περιβάλλον στο οποίο ο χρήστης μπορεί να δει ποιες διεργασίες εκτελέστηκαν με επιτυχία, πόσες διεργασίες ολοκληρώθηκαν, πόσες εκκρεμούν, πόσες εκτελούνται αυτή την στιγμή κ.ά. Στο Dependency Graph παρουσιάζεται το διάγραμμα ροής διεργασιών σε δεντρική μορφή με το οποίο μπορεί ο χρήστης να παρακολουθήσει την εκτέλεση κάθε διεργασίας και να δει τις αλληλεξαρτήσεις μεταξύ των διεργασιών. Ακολουθεί το tab με τους workers, στο οποίο βρίσκεται μία λίστα με τους διαθέσιμους workers, οι οποίοι είναι υπεύθυνοι για την εκτέλεση των διεργασιών, ενώ παρουσιάζονται κάποιες πληροφορίες σχετικά με το ποια διεργασία εκτέλεσε ο worker, πόσες τρέχει αυτή την στιγμή, πόσες εκκρεμούν κ.α.



Εικόνα 18: Ο Luigi Task Visualiser και η αρχική καρτέλα Task List.



Εικόνα 19: Η καρτέλα που υποδεικνύει το status κάθε διεργασίας και τις αλληλοεξαρτήσεις τους.



Εικόνα 20: Η καρτέλα με τους Workers.

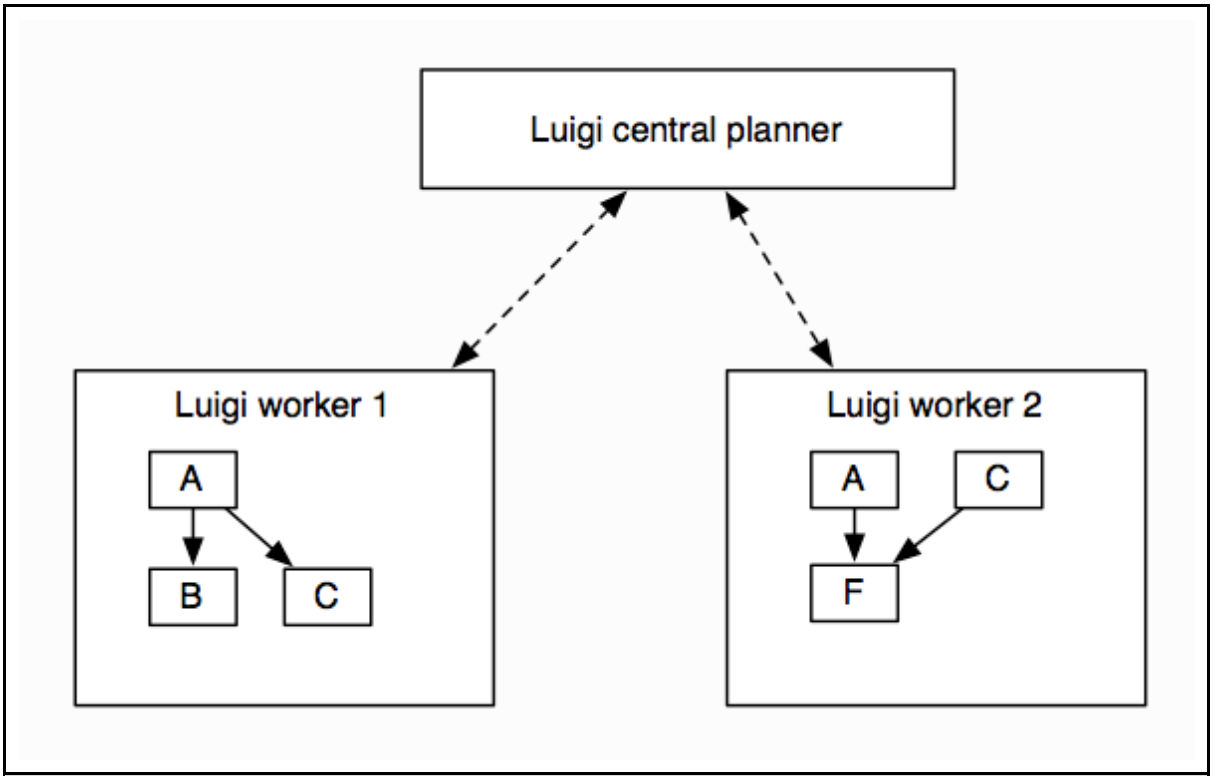
2.3.3 Αρχιτεκτονική του Luigi

Η αρχιτεκτονική του Luigi είναι σχετικά απλή σε σύγκριση με άλλα Workflow Orchestration Platforms. Το Luigi έχει δύο είδη scheduler, τον **local scheduler** και τον **central scheduler**. Ο local scheduler είναι ιδανικός για development και τον χειρίζεται κανείς μέσω της γραμμής εντολών, ενώ ο central scheduler χρησιμοποιείται κατά την παραγωγή (**Production**). Το γραφικό περιβάλλον που παρουσιάστηκε στην προηγούμενη υποενότητα είναι διαθέσιμο μόνο για τον central scheduler. Παράλληλα, ο central scheduler εγγυάται ότι δεν θα τρέξουν ταυτόχρονα δύο στιγμιότυπα της ίδιας διεργασίας [WP36] .

Δυστυχώς, στον Luigi δεν γίνεται να εκτελέσει κανείς διεργασίες περιοδικά. Αν ο Μηχανικός Δεδομένων θέλει να εκτελέσει συγκεκριμένες διεργασίες ανά συγκεκριμένα χρονικά διαστήματα, θα πρέπει τότε να προγραμματίσει κάποια **cron job**.

Επιπλέον, όταν στο Luigi εκτελείται ένα Workflow, τότε ο εργάτης (Worker) που θα το αναλάβει, θα εκτελέσει και όλες τις διεργασίες (Tasks) που περιέχει το συγκεκριμένο

Workflow. Παρόλα αυτά, μπορούν κατά την εκτέλεση του Workflow να οριστούν πάνω από ένας εργάτες (Workers) έτσι ώστε να μπορούν να εκτελούνται πάνω από μία διεργασίες ταυτόχρονα. [WP22]



Εικόνα 21: Η καρτέλα με τους Workers. [WP37]

2.3.4 Εκτέλεση ενός Workflow

Ένα πολύ σημαντικό χαρακτηριστικό του Luigi είναι ότι η κάθε διεργασία του είναι ταυτοδύναμη (**idempotent**). Αυτό σημαίνει ότι όσες φορές και αν εκτελεστεί η διεργασία, το αποτέλεσμα θα είναι πάντα το ίδιο. Επιπλέον, στο Luigi κάθε φορά πριν μία διεργασία τρέξει, ελέγχεται αν πληρούνται οι απαραίτητες προϋποθέσεις για την εκτέλεση της διεργασίας, δηλαδή αυτά που περιέχονται στην μέθοδο **requires()**. Αν δεν υπάρχουν οι κατάλληλες προϋποθέσεις, τότε εκτελείται πρώτα η διεργασία που προηγείται της διεργασίας αυτής.

Για παράδειγμα, στο διάγραμμα ροής διεργασιών της εικόνας 22, την πρώτη φορά που το **Luigi** θα εκτελέσει την διεργασία **SecondTask**, θα εκτελέσει αναγκαστικά και την διεργασία **FirstTask**, διότι η διεργασία **SecondTask** χρησιμοποιεί το **output** της διεργασίας **FirstTask**.

```
===== Luigi Execution Summary =====

Scheduled 2 tasks of which:
* 2 ran successfully:
    - 1 FirstTask(orderNum=2)
    - 1 SecondTask(orderNum=2)

This progress looks :) because there were no failed tasks or missing dependencies

===== Luigi Execution Summary =====
```

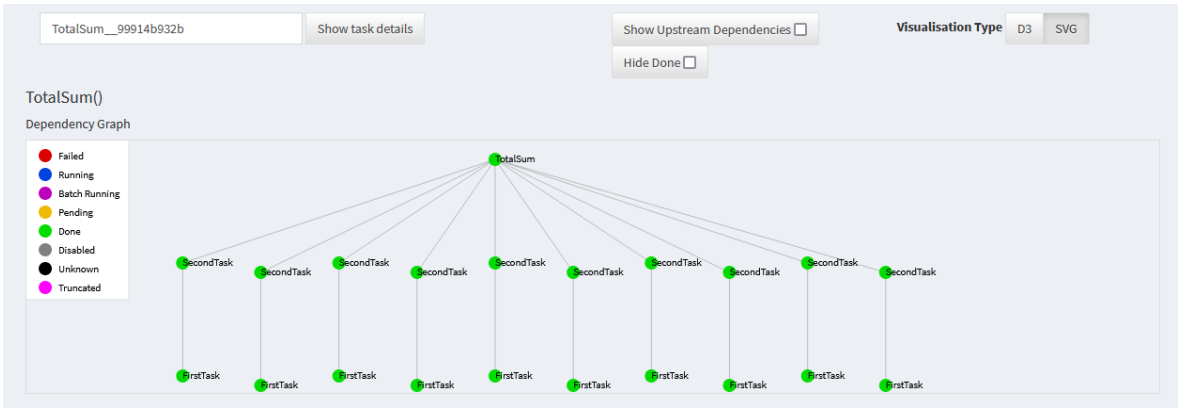
Εικόνα 22: Εκτέλεση της διεργασίας SecondTask, χωρίς να έχει εκτελεστεί στο παρελθόν η FirstTask.

Εάν η διεργασία FirstTask είχε εκτελεστεί κάποια στιγμή στο παρελθόν, όταν το Luigi θα προσπαθήσει να εκτελέσει την διεργασία SecondTask σε μεταγενέστερη ημερομηνία, θα ελέγξει αν τα προαπαιτούμενα για την εκτέλεση της διεργασίας υπάρχουν ήδη. Στην συγκεκριμένη περίπτωση επειδή η διεργασία FirstTask είχε ήδη εκτελεστεί στο παρελθόν και άρα είχε παραχθεί το output(), το Luigi θα εκτελέσει μόνο την διεργασία SecondTask.

```
===== Luigi Execution Summary =====

Scheduled 2 tasks of which:
* 1 complete ones were encountered:
    - 1 FirstTask(orderNum=2)
* 1 ran successfully:
    - 1 SecondTask(orderNum=2)

This progress looks :) because there were no failed tasks or missing dependencies

===== Luigi Execution Summary =====
```

Εικόνα 23: Εκτέλεση διεργασίας SecondTask, αφού έχει προηγηθεί κάποια στιγμή η εκτέλεση της FirstTask.

Το Luigi μπορεί πολύ εύκολα να εκτελέσει το ίδιο workflow παράλληλα δίνοντας διαφορετικά δεδομένα εισόδου κάθε φορά και επιτρέποντας έτσι τον παραλληλισμό διεργασιών όπως φαίνεται παρακάτω στην εικόνα 24.



Εικόνα 24: Παράλληλη εκτέλεση Διαγραμμάτων Ροών Διεργασιών.

2.4 Temporal

Στην ενότητα αυτή παρουσιάζονται πληροφορίες σχετικά με την χρήση, την λειτουργία και την αρχιτεκτονική του Temporal [WP20], ενός γενικής χρήσης Workflow Orchestrator με τον οποίο μπορεί κανείς να αναπτύξει γενικού σκοπού Workflows. Οι περισσότερες πληροφορίες που παρουσιάζονται σε αυτή την ενότητα έχουν παρθεί από το επίσημο documentation του Temporal [WP52]. Το Temporal μπορεί να εφαρμοστεί σε πολλές περιπτώσεις, όπως για παράδειγμα σε εφαρμογές όπου απαιτείται η ενορχίστρωση μικροϋπηρεσιών (Microservices Orchestration), η κατασκευή σωληναγωγών (Data Pipelines) και η εκτέλεση μακροπρόθεσμων διεργασιών (Long Running Processes), οι οποίες υπάρχει ανάγκη να τρέχουν για μήνες, ακόμα και χρόνια.

Είναι διαθέσιμο σε διάφορα SDKs γλωσσών προγραμματισμού όπως για παράδειγμα σε Golang, Java και PHP, πράγμα που το καθιστά αρκετά ευέλικτο. Επιπλέον, παρέχει έναν open source Server, ο οποίος μπορεί να ενσωματωθεί σε διάφορα περιβάλλοντα, όπως για παράδειγμα σε περιβάλλον νέφους (Cloud environment).

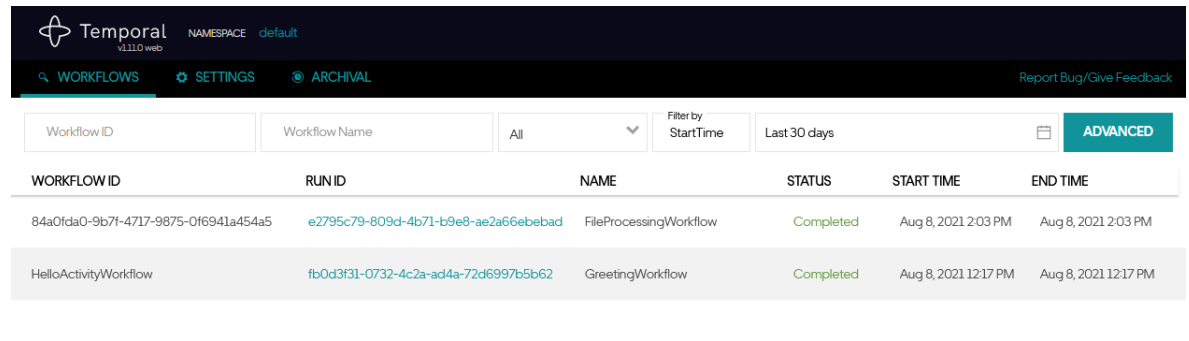
2.4.1 Κύρια χαρακτηριστικά του Temporal

Το Temporal περιλαμβάνει κάποια κύρια δομικά στοιχεία τα οποία είναι διαθέσιμα μέσω του SDK του και χρησιμοποιούνται για την ανάπτυξη εφαρμογών. Τα **Workflows** είναι ένα από αυτά τα δομικά στοιχεία και χρησιμοποιούνται σαν σημείο εκκίνησης της εφαρμογής. Τα **Activities** είναι μέθοδοι, οι οποίες μπορούν να καλεστούν μόνο μέσα σε ένα **Workflow** και

συνήθως χρησιμοποιούνται για να διαχειριστούν εξωτερικές πηγές, όπως για παράδειγμα την επικοινωνία της εφαρμογής με ένα API ή το κατέβασμα ενός αρχείου από έναν Web Server. Ένα ακόμα βασικό δομικό στοιχείο του Temporal είναι οι **Workers**. Οι **Workers** είναι ουσιαστικά υπηρεσίες (Services) και σκοπός τους είναι να εκτελούν τον κώδικα που βρίσκεται μέσα στα **Activities** αλλά και τον κώδικα των **Workflows**. Το Temporal παρέχει επιπλέον τα **Task Queues,** στα οποία μπαίνουν στην ουρά τα Tasks για να εκτελεστούν, δηλαδή ο εκτελέσιμος κώδικας. Τέλος, υπάρχουν τα **Signals** που χρησιμοποιούνται για να ανανεώσουν τις τιμές μεταβλητών σε ένα τρέχον Workflow, ενώ τα **Queries** χρησιμοποιούνται για να ανακτήσουν τιμές από μεθόδους που βρίσκονται μέσα σε ένα Workflow.

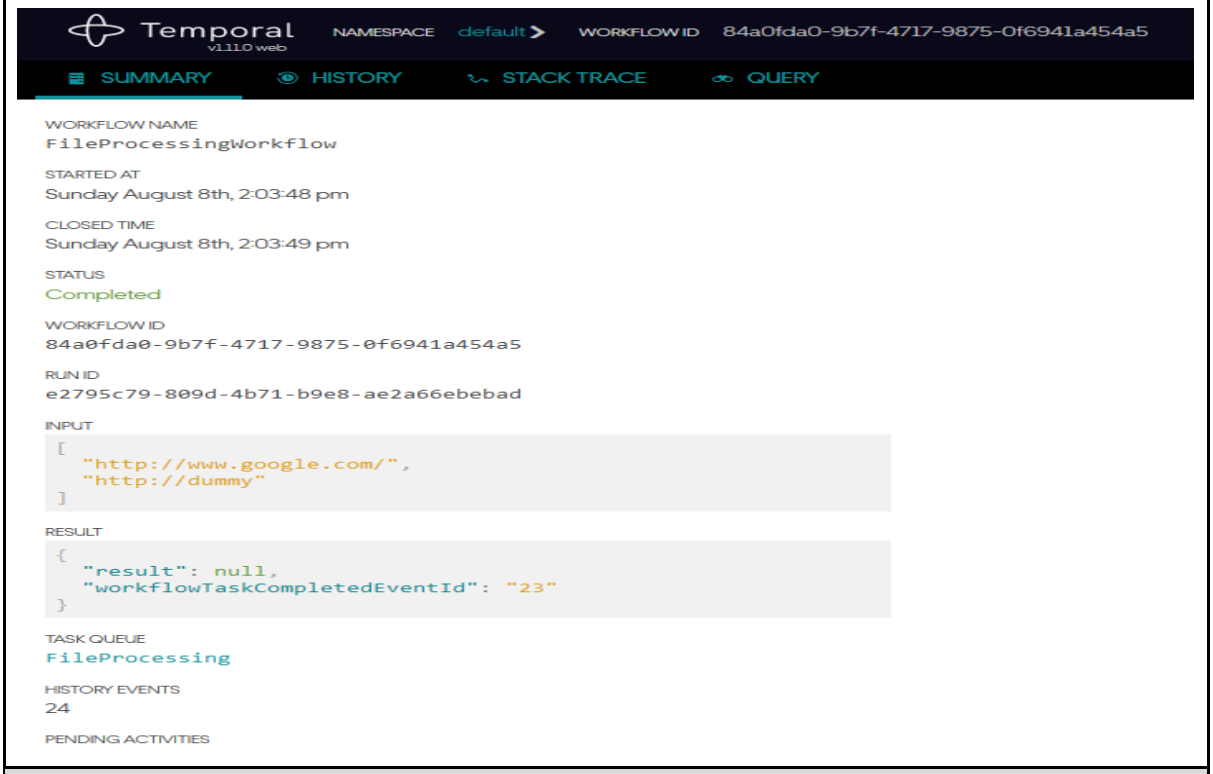
2.4.2 Temporal Server

Το Temporal παρέχει έναν Web Server από τον οποίο μπορεί κανείς να παρακολουθήσει την εκτέλεση τον Workflows, να δει τα αποτελέσματα της εκτέλεσης τους, τα βήματα που εκτελέστηκαν διαδοχικά για την εκτέλεση του Workflow κ.α. Αποτελείται από ένα αρκετά απλοϊκό design, το οποίο παρέχει τα απολύτως απαραίτητα εργαλεία για την παρακολούθηση των Workflows. Στις εικόνες 25 και 26 παρουσιάζεται το UI του Temporal Web Server χρησιμοποιώντας μερικά παραδείγματα από το Temporal Documentation.



Εικόνα 25: Η αρχική σελίδα του Temporal UI παρέχει μία λίστα με τα Workflows.

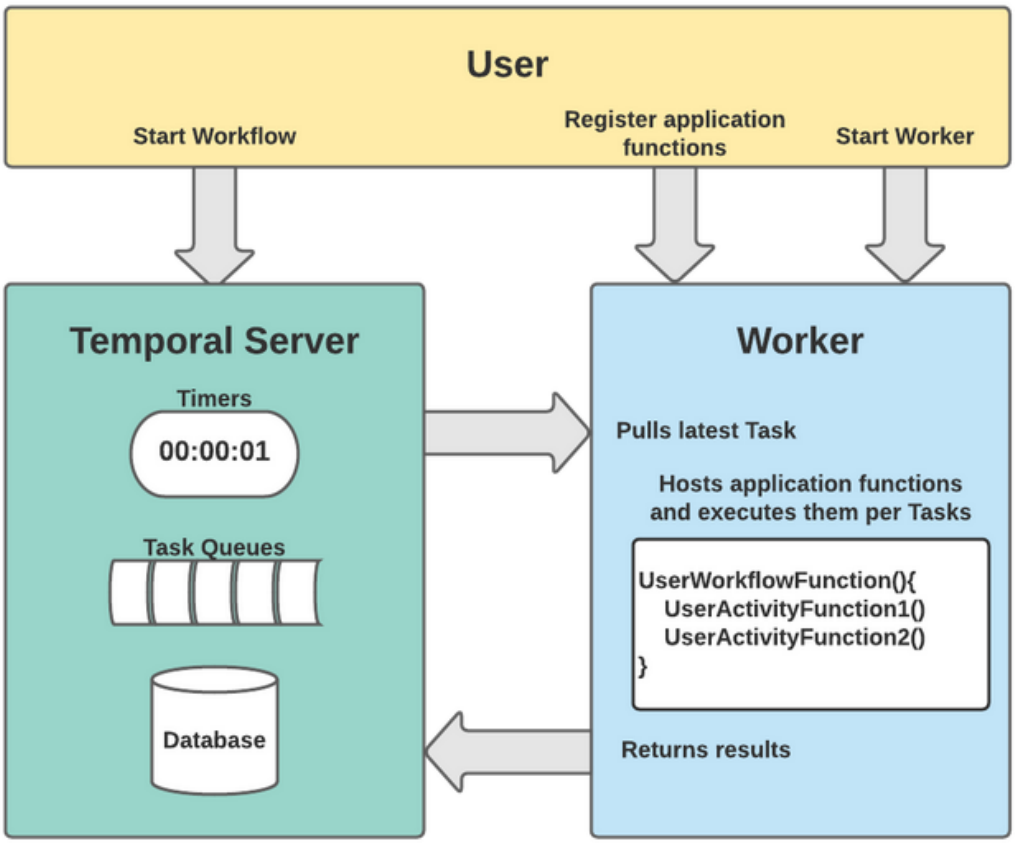
Επιλέγοντας κανείς ένα από τα Workflows μπορεί να δει περαιτέρω πληροφορίες όπως τα δεδομένα εισαγωγής σε ένα Workflow, δεδομένα τα οποία είναι αποτέλεσμα της εκτέλεσης του Workflow, το Status του Workflow, δηλαδή αν εκτελέστηκε με επιτυχία ή όχι, πότε εκτελέστηκε και πότε ολοκληρώθηκε κ.α.



Εικόνα 26: Λεπτομέρειες σχετικά με την εκτέλεση ενός Workflow.

2.4.3 Αρχιτεκτονική του Temporal

Σε υψηλό επίπεδο στο Temporal η αλληλεπίδραση του χρήστη με τους Workers και τον Temporal Server γίνεται ως εξής: ο χρήστης μέσω της γραμμής εντολών (tctl) έχει την δυνατότητα να δώσει ένα σήμα στον Server για να ξεκινήσει ένα Workflow. Τα Tasks μπαίνουν στην ουρά για εκτέλεση και με την πρώτη ευκαιρία ο Worker ανακτά το πρώτο διαθέσιμο Task που βρίσκεται στην ουρά και εκτελεί τον κώδικά του. Όταν ο Worker τελειώσει με το Task που έλαβε, επιστρέφει στον Server το αποτέλεσμα της εκτέλεσης. Ο χρήστης θα πρέπει να διαθέσει όσους Workers χρειαστεί για την εκτέλεση των διεργασιών. Ο Temporal Server διαθέτει επίσης μία βάση δεδομένων (Cassandra[WP38], MySQL [WP35], PostgreSQL[WP39]) στην οποία αποθηκεύονται όλα τα δεδομένα ενός Workflow.

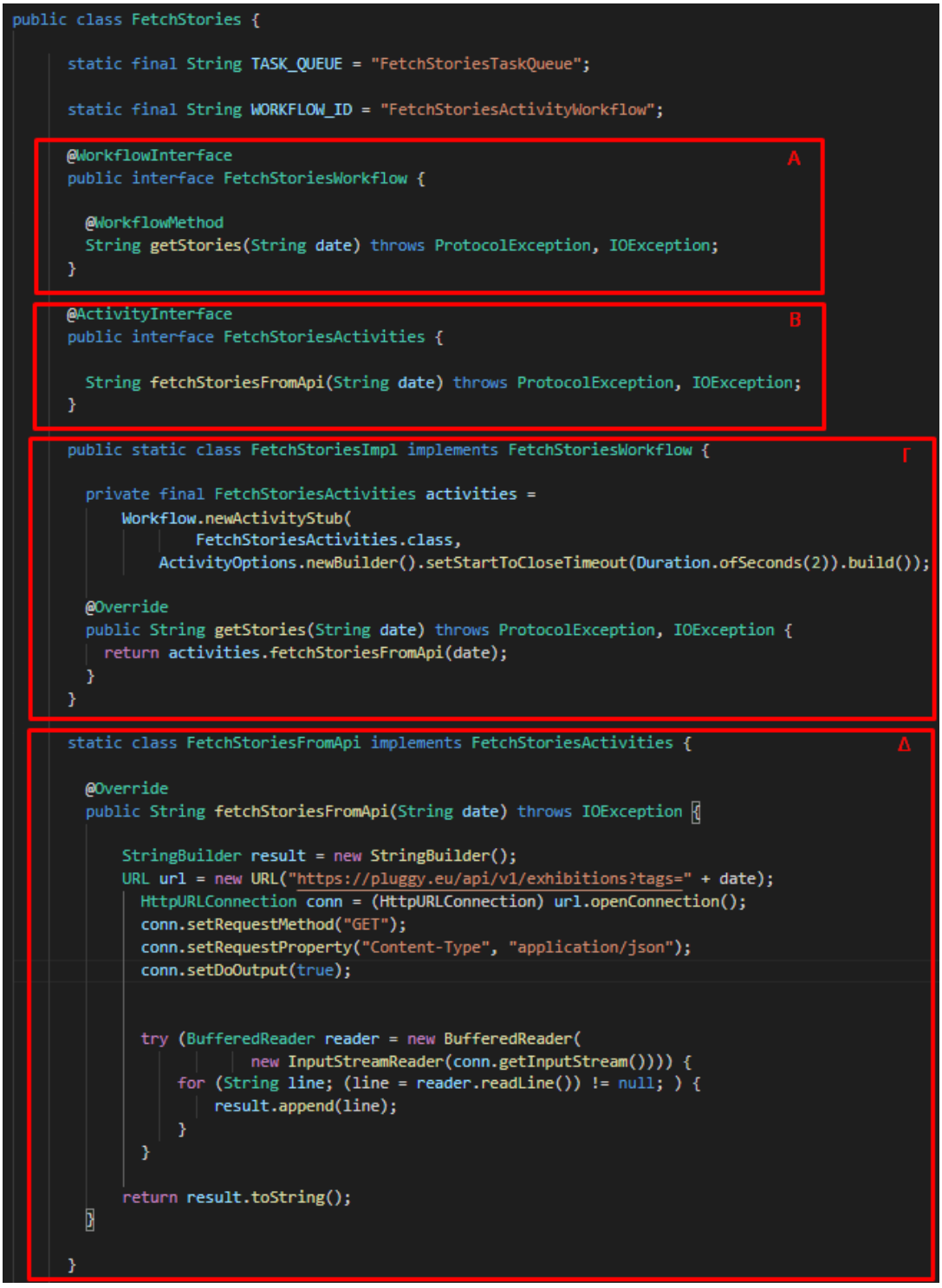


Εικόνα 27: Αλληλεπίδραση μεταξύ Temporal Service, Worker και χρήστη. [WP20]

2.4.4 Εκτέλεση ενός workflow στο Temporal

Ένα πολύ σημαντικό στοιχείο του Temporal είναι ότι παρέχει στον χρήστη πολλά SDK από διάφορες γλώσσες προγραμματισμού. Έτσι, μπορεί κάποιος να αναπτύξει το Workflow χρησιμοποιώντας το SDK της αρεσκείας του. Παρακάτω δημιούργησα ένα Workflow χρησιμοποιώντας το Java SDK του Temporal και τον Eclipse IDE[WP40]. Το Workflow αυτό παίρνει σαν όρισμα μία ημερομηνία στο entry point του workflow και αποτελείται από μία διεργασία η οποία εκτελεί μία http μέθοδο GET και λαμβάνει δεδομένα από το PLUGGY API σύμφωνα με την ημερομηνία που δόθηκε σαν παράμετρος.

Στην εικόνα 28 παρουσιάζεται η κύρια δομή ενός Workflow στο Temporal χρησιμοποιώντας το Java SDK. Στο μπλοκ Α βρίσκεται η διεπαφή FetchStoriesWorkflow, η οποία περιέχει την μέθοδο getStories(). Η μέθοδος αυτή είναι ουσιαστικά το entry point του Workflow και όταν ο χρήστης εκτελεί το workflow θα πρέπει να δώσει το όρισμα που ταιριάζει στην παράμετρο της μεθόδου αυτής. Στο μπλοκ αυτό δεν θα πρέπει να γίνονται επιζήμιοι υπολογισμοί που καταναλώνουν πολλούς πόρους, όπως συνδέσεις σε Βάσεις Δεδομένων, API calls κλπ και θα πρέπει να αποφεύγεται να γράφεται μη-ντετερμινιστικός κώδικας.
Στο μπλοκ Β βρίσκεται η διεπαφή που ορίζει τις διεργασίες (Activities) και είναι τα δομικά στοιχεία ενός Workflow στο Temporal. Οι διεργασίες αυτές είναι ο πυρήνας της λογικής του Workflow και εδώ εκτελούνται όλες οι διεργασίες όπως υπολογισμοί, συνδέσεις με Βάσεις Δεδομένων κ.α. Στο συγκεκριμένο παράδειγμα η μόνη διεργασία είναι αυτή που κάνει κλήση στο API. Στο μπλοκ Γ υλοποιείται η διεπαφή του Workflow, γίνεται επίκληση των activities που εκτελούνται σε Workers. Τέλος, στο μπλοκ Δ υλοποιείται η διεπαφή των Activities και συνεπώς των μεθόδων των Activities, ενώ εκτελείται και ο κώδικας που περιέχεται σε κάθε Activity, που στην προκειμένη περίπτωση είναι η κλήση στο API.



```java
public class FetchStories {

    static final String TASK_QUEUE = "FetchStoriesTaskQueue";

    static final String WORKFLOW_ID = "FetchStoriesActivityWorkflow";

    @WorkflowInterface                                                    A
    public interface FetchStoriesWorkflow {

      @WorkflowMethod
      String getStories(String date) throws ProtocolException, IOException;
    }

    @ActivityInterface                                                    B
    public interface FetchStoriesActivities {

      String fetchStoriesFromApi(String date) throws ProtocolException, IOException;
    }

    public static class FetchStoriesImpl implements FetchStoriesWorkflow {   Γ

      private final FetchStoriesActivities activities =
          Workflow.newActivityStub(
                  FetchStoriesActivities.class,
              ActivityOptions.newBuilder().setStartToCloseTimeout(Duration.ofSeconds(2)).build());

      @Override
      public String getStories(String date) throws ProtocolException, IOException {
        return activities.fetchStoriesFromApi(date);
      }
    }

    static class FetchStoriesFromApi implements FetchStoriesActivities {     Δ

      @Override
      public String fetchStoriesFromApi(String date) throws IOException {

          StringBuilder result = new StringBuilder();
          URL url = new URL("https://pluggy.eu/api/v1/exhibitions?tags=" + date);
            HttpURLConnection conn = (HttpURLConnection) url.openConnection();
            conn.setRequestMethod("GET");
            conn.setRequestProperty("Content-Type", "application/json");
            conn.setDoOutput(true);


            try (BufferedReader reader = new BufferedReader(
                        new InputStreamReader(conn.getInputStream()))) {
                for (String line; (line = reader.readLine()) != null; ) {
                    result.append(line);
                }
            }

          return result.toString();
      }

    }
```

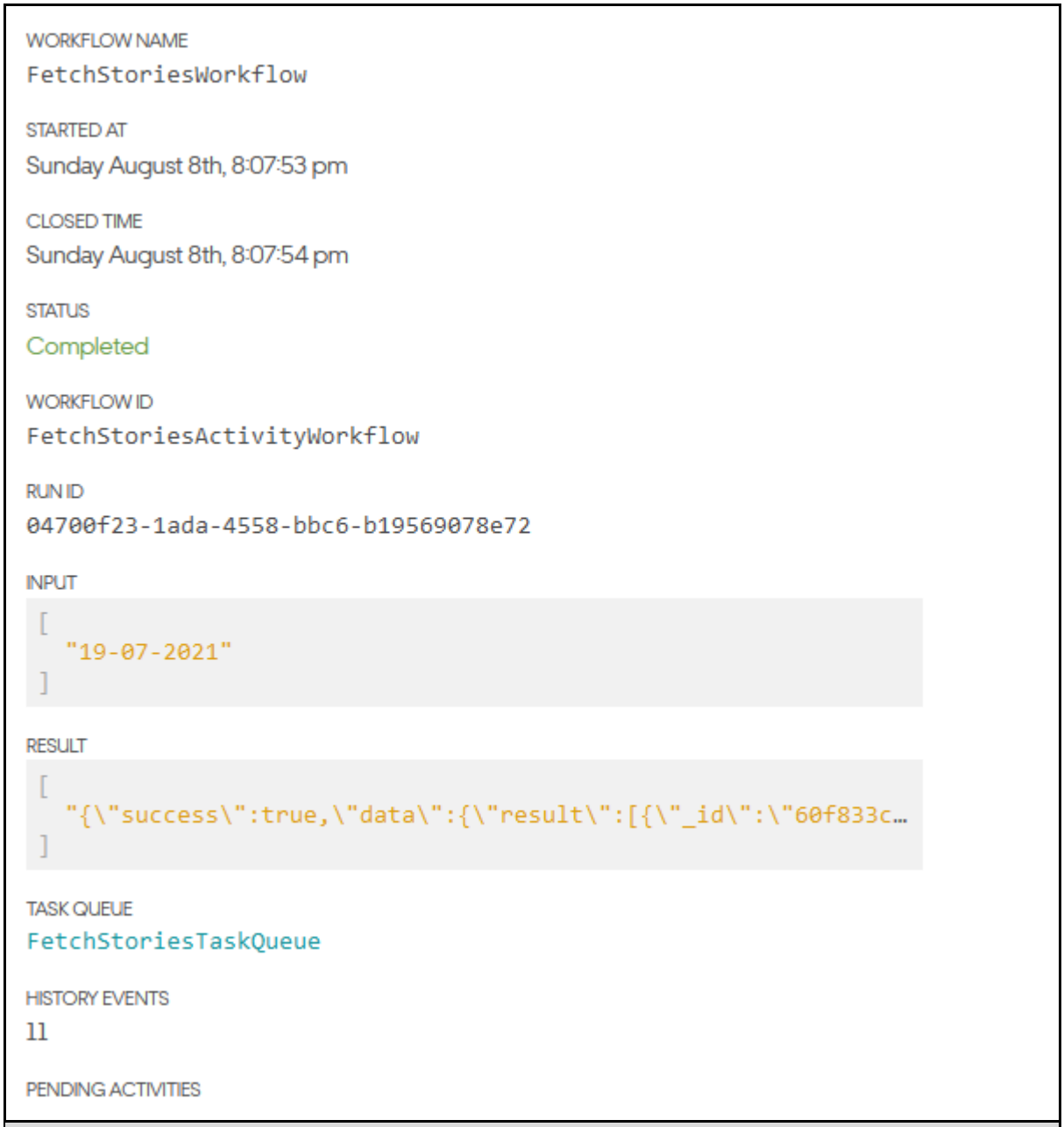
Εικόνα 28: Δομή ενός Workflow χρησιμοποιώντας το Java SDK του Temporal.

Κατά την εκτέλεση του Workflow (Εικόνα 29) δημιουργείται ένα στιγμιότυπο του Workflow service και συνδέεται σε αυτό ένας client, ο οποίος μπορεί να ξεκινήσει την εκτέλεση του Workflow. Στην συνέχεια δημιουργείται ένας Worker που επιβλέπει την ουρά διεργασιών του συγκεκριμένου Workflow και εγγράφονται σε αυτό το Workflow και οι Activities. Στην συνέχεια εκκινείται το Workflow και δίνεται σαν όρισμα η ημερομηνία για την μέθοδο του Workflow. Όταν η εκτέλεση του Workflow ολοκληρωθεί, εκτυπώνεται το αποτέλεσμα στην οθόνη.

```
public static void main(String[] args) throws ProtocolException, IOException {

    WorkflowServiceStubs service = WorkflowServiceStubs.newInstance();

    WorkflowClient client = WorkflowClient.newInstance(service);

    WorkerFactory factory = WorkerFactory.newInstance(client);

    Worker worker = factory.newWorker(TASK_QUEUE);

    worker.registerWorkflowImplementationTypes(FetchStoriesImpl.class);

    worker.registerActivitiesImplementations(new FetchStoriesFromApi());


    factory.start();

    FetchStoriesWorkflow workflow =
        client.newWorkflowStub(
                FetchStoriesWorkflow.class,
            WorkflowOptions.newBuilder()
                .setWorkflowId(WORKFLOW_ID)
                .setTaskQueue(TASK_QUEUE)
                .build());

    String fetchedStories = workflow.getStories("19-07-2021");

    System.out.println(fetchedStories);
    System.exit(0);
  }
```

Εικόνα 29: Εκτέλεση του Workflow και δημιουργία Worker.

Την διαδικασία εκτέλεσης του Workflow, καθώς και τα βήματα που ακολουθούνται κατά την εκτέλεση, μπορεί να τα παρακολουθήσει κανείς μέσω του Temporal UI, όπως φαίνεται και στην εικόνα 30.



Εικόνα 30: Αποτέλεσμα της εκτέλεσης του Workflow.

Κεφάλαιο 3: HYPERION Project

3.1 Εισαγωγή

Το περιεχόμενο αυτής της εισαγωγής έχει γραφτεί σύμφωνα με τις πληροφορίες που βρίσκονται στο [WP5].

Η κλιματική αλλαγή και διάφορα γεωφυσικά φαινόμενα που συμβαίνουν σε ιστορικά μέρη στα οποία βρίσκονται Μνημεία Πολιτιστικής Κληρονομιάς (Cultural Heritage Sites) επηρεάζουν σημαντικά την οικονομία της περιοχής, ενώ παράλληλα δημιουργούν αρνητικές επιπτώσεις στις κοινωνίες τους. Η διατήρηση των Μνημείων Πολιτιστικής Κληρονομιάς είναι μία δύσκολη πρόκληση, στην οποία σημαντικό ρόλο παίζουν τα υλικά και οι τεχνολογίες κατασκευής, οι στρατηγικές ανακατασκευής κ.α.[WP5]

Το Hyperion θα χρησιμοποιήσει **υπάρχοντα εργαλεία** και **υπηρεσίες** (όπως για παράδειγμα, μοντέλα ακραίων κλιματικών φαινομένων), **καινοτόμες τεχνολογίες** (όπως δορυφορική απεικόνιση για την επιθεώρηση μεγάλων περιοχών σε έκταση, μηχανική μάθηση κλπ) έτσι ώστε να προσφέρει μία ενσωματωμένη πλατφόρμα εκτίμησης της ανθεκτικότητας (integrated resilience assessment platform). Επιπλέον, θα βοηθάει τον τελικό χρήστη (ειδικοί του τομέα) να καταλάβει τις επιπτώσεις τον φαινομένων αυτών, να τον προετοιμάσει για μία καλύτερη, γρήγορη και αποτελεσματική αντιμετώπιση, καθώς και για μία βιώσιμη ανακατασκευή της περιοχής αυτής. [WP5]

Για την πραγματοποίηση των παραπάνω, το Hyperion συνυπολογίζει τα τοπικά οικοσυστήματα των Μνημείων Πολιτικής Κληρονομιάς και πραγματοποιώντας μία βιώσιμη προσέγγιση ανακατασκευής σε τεχνικό, κοινωνικό, περιβαλλοντολογικό, οικονομικό και θεσμικό επίπεδο, συμπεριλαμβάνοντας την **ενεργή συμμετοχή των κοινοτήτων** αλλά και επιχειρησιακών μοντέλων βασισμένα στο concept της "**load-balancing**" οικονομίας και προσφέροντας εργαλεία τύπου **financial risk-transfer tools**, με τα οποία μπορεί να υπολογίσει το κεφάλαιο που χρειάζεται για την γρήγορη ανακατασκευή τους. [WP5]

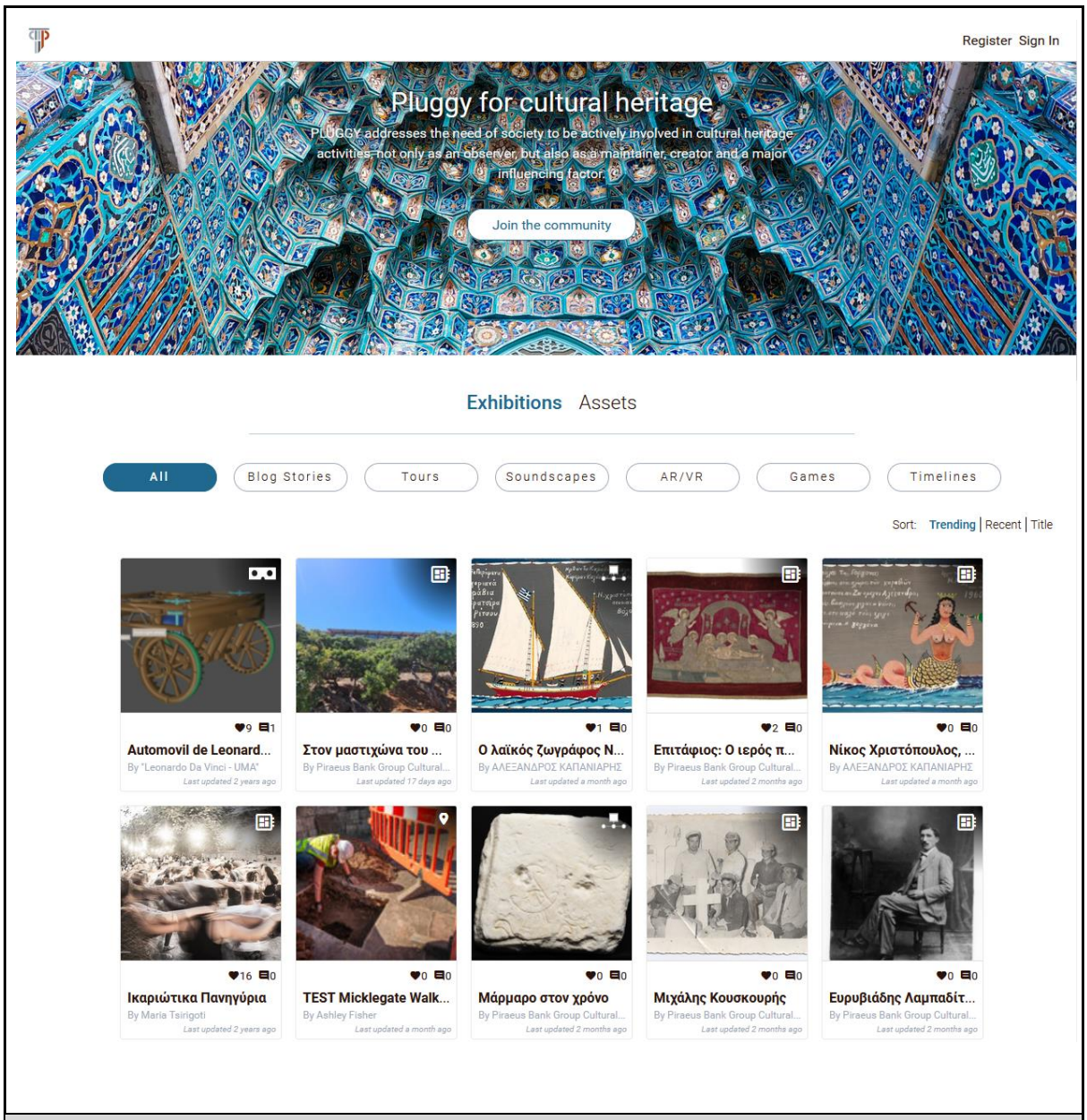
Στο Hyperion Project συνεργάζονται ένας πλήθος οργανισμών, εταιριών και φορέων για την ολοκλήρωσή του.[WP49] Το Hyperion θα πραγματοποιήσει δοκιμές σε τέσσερα σημεία της Ευρώπης. Τα σημεία αυτά είναι το νησί της Ρόδου στην Ελλάδα, η πόλη της Γρανάδα στην Ισπανία, το Τόνσμπεργκ στην Νορβηγία και η Βενετία στην Ιταλία.[WP50]

3.2 Communities' Engagement Tool - ICT Tool

Όπως προαναφέρθηκε και παραπάνω το Hyperion προτρέπει την κοινότητα να συμμετάσχει ενεργά στην αντιμετώπιση καταστροφών σε μέρη Παγκόσμιας Πολιτιστικής Κληρονομιάς. Συγκεκριμένα, κάτοικοι, αλλά και τοπικοί ιδιοκτήτες επιχειρήσεων μπορούν να χρησιμοποιήσουν το **Hyperion Communities' Engagement Tool** για να δημιουργήσουν και να δημοσιεύσουν stories, τα οποία αφορούν καταστροφές που έχουν συμβεί σε ιστορικά κτήρια και επιχειρήσεις που βρίσκονται σε περιοχές πολιτιστικής κληρονομιάς.

Εικόνα 31: Το λογότυπο του Hyperion Project, όπως αυτό φαίνεται στην επίσημη ιστοσελίδα του Project. [WP6]

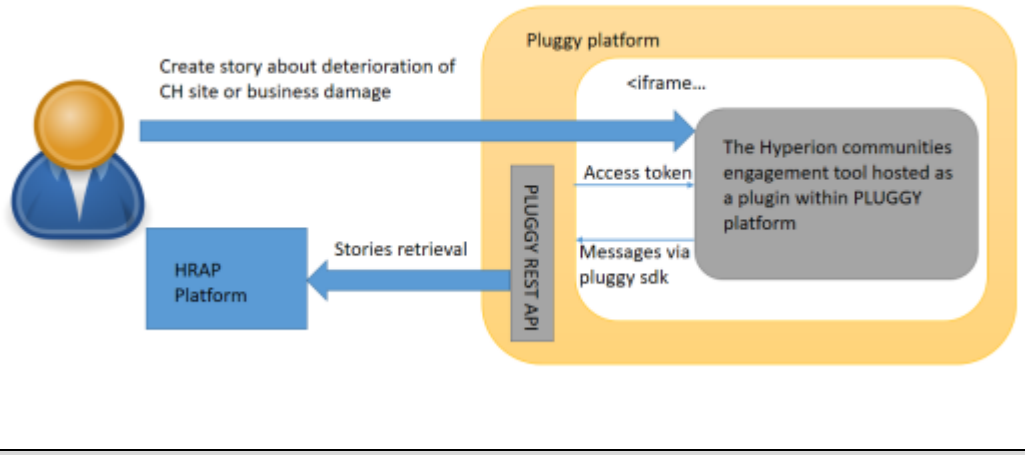
Το **Hyperion Communities' Engagement Tool** χρησιμοποιεί το API του **PLUGGY** [WP7] [WP8]. Το **PLUGGY** είναι ένα μέσο κοινωνικής δικτύωσης που έχει ως σκοπό την ανάδειξη των ευρωπαϊκών τοπίων πολιτιστικής κληρονομιάς (**European Cultural Heritage Landscape**) και προτρέπει τους Ευρωπαίους πολίτες να συμμετέχουν ενεργά σε δραστηριότητες πολιτιστικής κληρονομιάς, δημιουργώντας και δημοσιεύοντας οι ίδιοι τις δικές τους εκθέσεις (**Exhibitions**), των οποίων τα εκθέματα είναι κομμάτι της ευρωπαϊκής πολιτιστικής κληρονομιάς.



Εικόνα 32: Το PLUGGY. [WP8]

Οι λόγοι που το **Hyperion Communities' Engagement Tool** χρησιμοποιεί την κοινωνική πλατφόρμα PLUGGY είναι για να εκμεταλλευτεί τους ήδη υπάρχοντες χρήστες του PLUGGY, για να μπορεί να λάβει τα Stories από το PLUGGY χρησιμοποιώντας το API του, ενώ οι χρήστες δεν θα χρειάζονται να έχουν δύο διαφορετικούς λογαριασμούς (έναν για το PLUGGY και έναν για το HCET), αφού το HCET είναι ουσιαστικά ένα Plug-in του PLUGGY. Τέλος,

όλα τα περιεχόμενα του HCET (Citizen και Business Stories) θα μπορούν να προβληθούν μέσω του PLUGGY. [2]



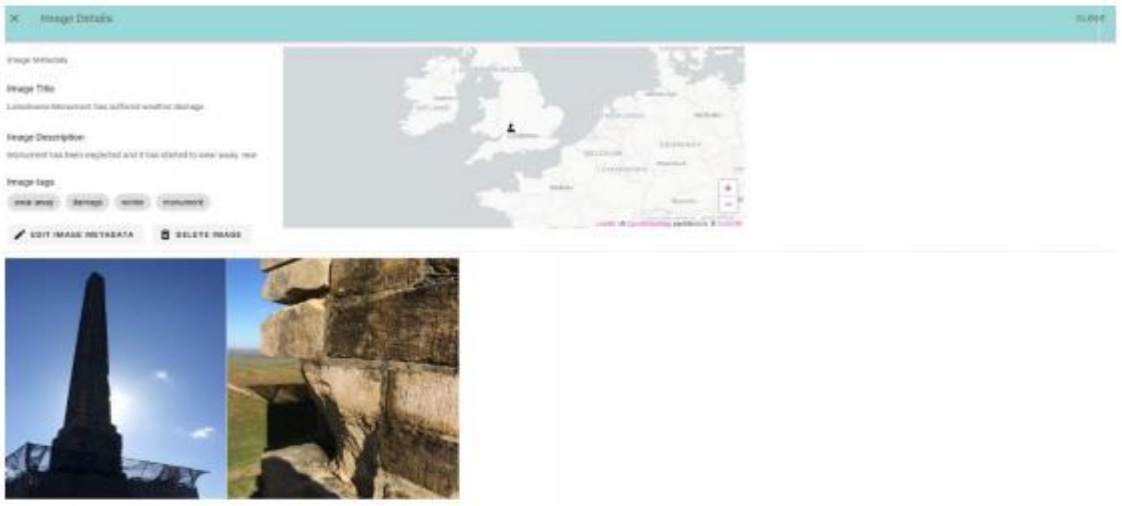
Εικόνα 33: Ενσωμάτωση του HCET στο PLUGGY και αλληλεπίδραση χρήστη, PLUGGY και HRAP. [2]

3.2.1 Hyperion Stories

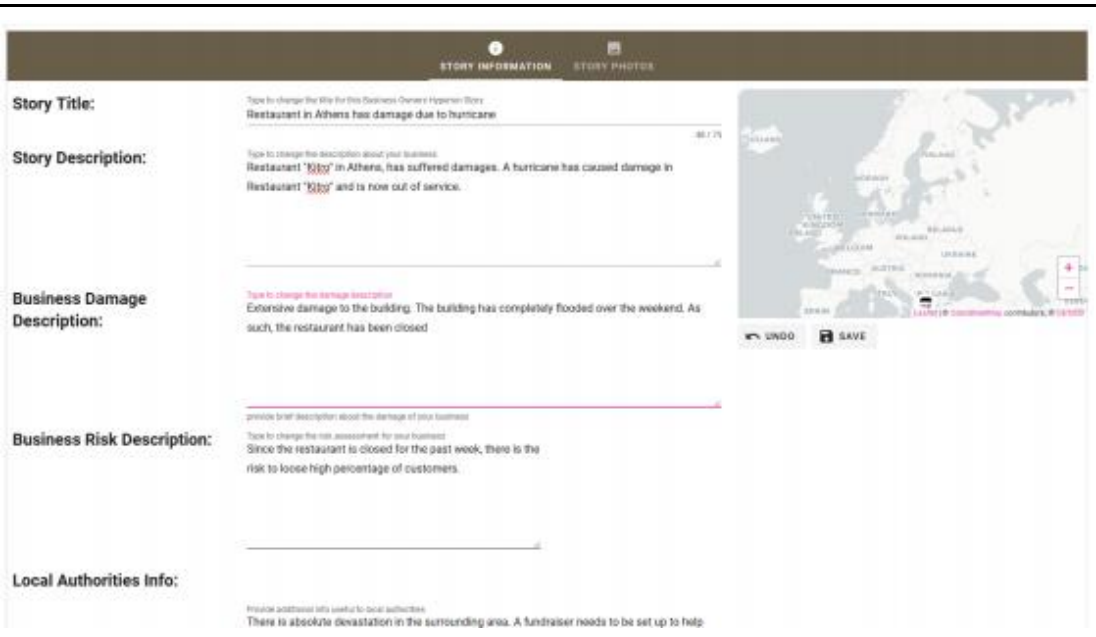
Όπως αναφέρθηκε προηγουμένως οι χρήστες θα είναι σε θέση να δημοσιεύσουν μέσω του **Hyperion Communities' Engagement Tool** Stories, τα οποία σύμφωνα με το [2] μπορούν να ανήκουν σε μία από τις ακόλουθες κατηγορίες:

1. **Hyperion Citizen Stories:** Stories, τα οποία δημοσιεύονται από έναν πολίτη και σκοπό έχουν να δημοσιεύσουν πιθανές καταστροφές που παρατηρήθηκαν σε ένα Μνημείο Παγκόσμιας Κληρονομιάς. Το Story αυτό θα περιέχει πληροφορίες όπως: Τίτλο, Περιγραφή της καταστροφής του μνημείου, σχετικές φωτογραφίες και ετικέτες (**tags**).
2. **Hyperion Business Owners Stories:** Τα συγκεκριμένα Stories αφορούν τους κατόχους επιχειρήσεων που βρίσκονται σε περιοχές που περιλαμβάνουν Μνημεία Παγκόσμιας Κληρονομιάς. Οι κάτοχοι τέτοιων επιχειρήσεων θα είναι σε θέση να αναφέρουν μέσω των Stories ζημιές που έπαθαν οι επιχειρήσεις τους και συγκεκριμένα επιπλέον πληροφορίες όπως: Περιγραφή ζημιάς επιχείρησης, εκτίμηση κινδύνου, σχετικές πληροφορίες, οι οποίες θα είναι χρήσιμες για τις τοπικές αρχές, ποσοστό

μείωσης πελατών λόγω της καταστροφής, προτεραιότητα και κρισιμότητα του προβλήματος, αλλά και σχετικές ετικέτες (**tags**).



Εικόνα 34: Διαδικασία δημιουργίας Story πολιτών. [2]

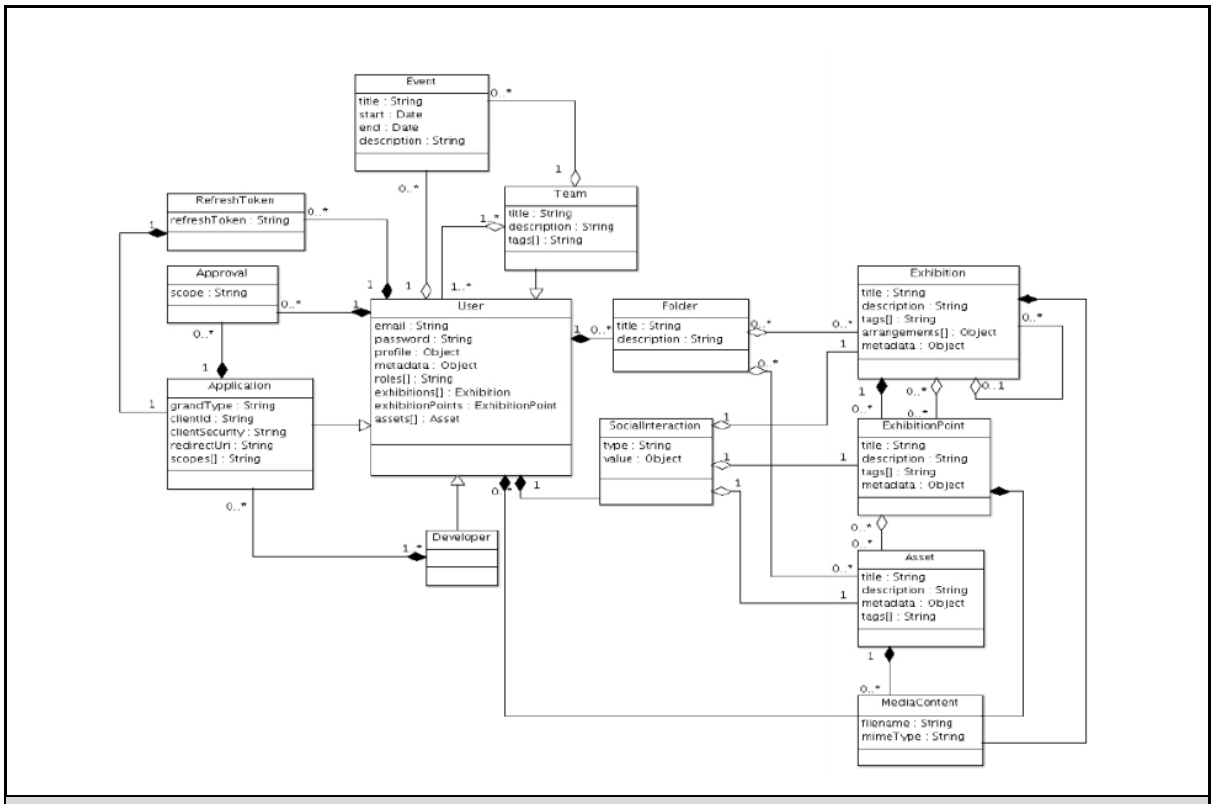


Εικόνα 35: Διαδικασία δημιουργίας Story για επιχειρήσεις. [2]

Όπως ειπώθηκε και προηγουμένως, το HCET είναι ένα plug-in του PLUGGY και αυτό σημαίνει ότι βασίζεται σε μεγάλο βαθμό σε αυτό. Συγκεκριμένα, το HCET αποθηκεύει τα

δεδομένα (Stories) στην Βάση Δεδομένων του PLUGGY, η οποία εκτίθεται από το PLUGGY μέσω του API του.

Όπως φαίνεται και στην εικόνα 34 και στην εικόνα 35, ο χρήστης θα μπορεί μέσω του HCET να δημιουργήσει ένα Story που αφορά ένα Μνημείο Πολιτιστικής Κληρονομιάς ή μία Επιχείρηση. Στην συνέχεια η πλατφόρμα HRAP θα το λαμβάνει μέσω του REST API του PLUGGY, όπως φαίνεται στην εικόνα 33.
Ο ευέλικτος και γενικός σχεδιασμός του PLUGGY το καθιστά ιδανικό για να το χρησιμοποιήσουν άλλες εφαρμογές αλλά και για να είναι εφικτό να φτιαχτούν σε αυτό plug-in, όπως αυτό του HCET.



Εικόνα 36: Το μοντέλο δεδομένων του PLUGGY. [2]

Από το παραπάνω μοντέλο, το HCET θα χρησιμοποιήσει με τον δικό του τρόπο τις οντότητες **Exhibition**, **ExhibitionPoint**, **Asset** και **MediaContent**, οι οποίες περιγράφονται στο [2] και βρίσκονται παρακάτω:

- **Exhibition:** Η οντότητα αυτή αντιστοιχεί σε ένα Story, είτε αυτό είναι Story πολίτη είτε Story επιχείρησης. Το κάθε Story με την σειρά του ίσως περιέχει ένα ή περισσότερα **ExhibitionPoints**, τα οποία μπορεί να περιέχουν επιπλέον πληροφορίες που αφορούν το story, ενώ επιπλέον ένα **Exhibition** παρέχει πληροφορίες όπως ημερομηνία δημιουργίας, τίτλο Ιστορίας, περιγραφή ιστορίας κ.α.
- **ExhibitionPoints:** Το ExhibitionPoint περιέχει επιπλέον πληροφορίες που αφορούν το story, ενώ στην περίπτωση του Hyperion Project και όταν το Story αφορά επιχειρήσεις, ο χρήστης μπορεί να κατεβάσει μέσω του ExhibitionPoint και ένα αρχείο JSON, το οποίο περιλαμβάνει επιπλέον πληροφορίες για το Story. Αυτές οι πληροφορίες είναι οι ακόλουθες:
    - **Business damage description:** Επιπλέον περιγραφή της καταστροφής που υπέστη η επιχείρηση.
    - **Business risk description:** Πιθανόν επιχειρησιακό ρίσκο που ίσως επέλθει στην επιχείρηση. Για παράδειγμα απώλεια πελατών επειδή η επιχείρηση θα παραμείνει κλειστή.
    - **Local authorities info:** Πληροφορίες που αφορούν την καταστροφή που έλαβε χώρα στη ευρύτερη περιοχή και οι οποίες έχουν δοθεί από κάποια τοπική αρχή (π.χ. Αστυνομία).
    - **Customers lost percentage:** Ποσοστό απώλειας πελατών (π.χ. -70%)
    - **Priority of the issue:** Ο βαθμός προτεραιότητας του προβλήματος (π.χ. High, Medium, Low, κ.α)
    - **Criticality of the issue:** Ο βαθμός κρισιμότητας του προβλήματος (π.χ. High, Medium, Low, κ.α)
    - **Relevant tags:** Ετικέτες που προστίθενται για να χαρακτηρίσουν επιπλέον το Story (π.χ. earthquake, flood, restaurant)
- **Asset:** Το Asset είναι μία οντότητα, η οποία μπορεί να περιλαμβάνει κάποια δεδομένα όπως για παράδειγμα εικόνες από την καταστροφή. Οι εικόνες αυτές συνοδεύονται με πληροφορίες όπως τίτλος, περιγραφή, ετικέτες, ημερομηνία δημιουργίας, πνευματικά δικαιώματα κ.α.

Όπως ειπώθηκε και προηγουμένως τα δεδομένα που αφορούν τα Hyperion Stories θα εκτίθενται μέσω κάποιων endpoints του PLUGGY API. Ένα Story θα είναι σε μορφή JSON και θα έχει την ακόλουθη μορφή:


```
{
    "__v": 1,
    "_id": "5fe321d5333a33a7f10e4e06",
    "arrangements": [],
    "createdAt": "2020-12-23T10:54:13.762Z",
    "creator": {
        "_id": "5c96c0a28235f62760f6ef31",
        "fullName": "Nikos Tousert",
        "id": "5c96c0a28235f62760f6ef31",
        "kind": "UserPerson",
        "mediaContent": [],
        "profile": {
            "firstName": "Nikos",
            "lastName": "Tousert"
        }
    },
    "description": "Restaurant \"Kitro\" in Athens, has suffered damages. A hurricane has caused damage in Restaurant \"Kitro\" and is now out of service.",
    "exhibitionPoints": [
        {
            "__v": 3,
            "_id": "5fe3233c333a33a7f10e4e08",
            "assets": [
                {
                    "__v": 4,
                    "_id": "5fe323ac333a33a7f10e4e0e",
                    "coverAbsoluteUrl": "/api/v1/assets/5fe323ac333a33a7f10e4e0e/media/602590ff10ed9eee408f7fda/facebook",
                    "createdAt": "2020-12-23T11:02:04.963Z",
```

```
                    "creator": "5c96c0a28235f62760f6ef31",
                    "description": "The kitchen of the restaurant
is flooded.",
                    "id": "5fe323ac333a33a7f10e4e0e",
                    "kind": "ContentAsset",
                    "legal": {
                        "isOwnWork": false
                    },
                    "location": {
                        "geo": {
                            "coordinates": [
                                23.76722283512782,
                                38.04771086637302
                            ],
                            "type": "Point",
                            "zoom": 14
                        }
                    },
                    "mediaContent": [
                        {
                            "_id": "602590ff10ed9eee408f7fda",
                            "aliases": null,
                            "chunkSize": 261120,
                            "contentType": "image/jpeg",
                            "filename":     "flooded-restaurant-
kitchen.jpg",
                            "length": 71621,
                            "md5":
"c82be02f087bd1f784a4def5fadc7b3d",
                            "metadata": null,
                            "uploadDate":              "2021-02-
11T20:18:07.662Z"
                        }
                    ],
                    "owner": "5c96c0a28235f62760f6ef31",
```

```
                    "public": true,
                    "social": {
                        "allViews": [],
                        "comments": [],
                        "lastMonthViews": [],
                        "likes": [],
                        "report": [],
                        "views": 23
                    },
                    "tags": [
                        "flooded",
                        "kitchen",
                        "restaurant",
                        "Hyperion Story",
                        "13-09-2021"
                    ],
                    "title": "The kitchen",
                    "type": "image",
                    "updatedAt": "2021-04-01T11:08:47.817Z"
                }
            ],
            "content": {
                "content": "602572b510ed9eee408f7fd6",
                "contentType": "json"
            },
            "coverAbsoluteUrl":
"/public/images/imagenotavailable_200.png",
            "createdAt": "2020-12-23T11:00:12.154Z",
            "creator": "5c96c0a28235f62760f6ef31",
            "id": "5fe3233c333a33a7f10e4e08",
            "kind": "ContentExhibitionPoint",
            "legal": {
                "isOwnWork": false
            },
            "location": {
```

```
            "geo": {
                "coordinates": [
                    0,
                    0
                ],
                "type": "Point",
                "zoom": 12
            }
        },
        "mediaContent": [],
        "owner": "5c96c0a28235f62760f6ef31",
        "previewMedia": {
            "_id": "602590ff10ed9eee408f7fda",
            "aliases": null,
            "chunkSize": 261120,
            "contentType": "image/jpeg",
            "filename": "flooded-restaurant-kitchen.jpg",
            "length": 71621,
            "md5": "c82be02f087bd1f784a4def5fadc7b3d",
            "metadata": null,
            "uploadDate": "2021-02-11T20:18:07.662Z"
        },
        "public": true,
        "social": {
            "allViews": [],
            "comments": [],
            "lastMonthViews": [],
            "likes": [],
            "report": [],
            "views": 0
        },
        "tags": [
            "restaurant",
            "damage",
            "flooded"
```

```
            ],
            "title": "Business Damage",
            "updatedAt": "2021-04-01T11:08:47.520Z"
        }
    ],
    "exhibitions": [],
    "kind": "ContentExhibition",
    "legal": {
        "isOwnWork": false
    },
    "location": {
        "geo": {
            "coordinates": [
                23.76728825448719,
                38.04769396850125
            ],
            "type": "Point",
            "zoom": 14
        }
    },
    "mediaContent": [],
    "owner": {
        "_id": "5c96c0a28235f62760f6ef31",
        "fullName": "Nikos Tousert",
        "id": "5c96c0a28235f62760f6ef31",
        "kind": "UserPerson",
        "mediaContent": [],
        "profile": {
            "firstName": "Nikos",
            "lastName": "Tousert"
        }
    },
    "previewMedia": {
        "_id": "602590ff10ed9eee408f7fda",
        "aliases": null,
```

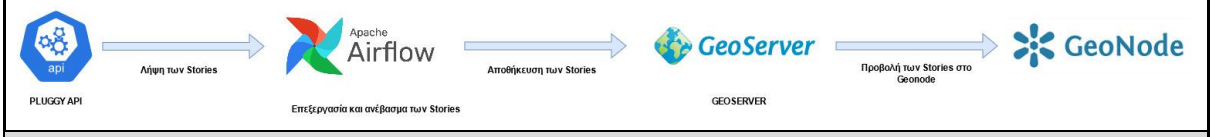
```
        "chunkSize": 261120,
        "contentType": "image/jpeg",
        "filename": "flooded-restaurant-kitchen.jpg",
        "length": 71621,
        "md5": "c82be02f087bd1f784a4def5fadc7b3d",
        "metadata": null,
        "uploadDate": "2021-02-11T20:18:07.662Z"
    },
    "public": true,
    "social": {
        "allViews": [],
        "comments": [],
        "lastMonthViews": [],
        "likes": [],
        "report": [],
        "views": 145
    },
    "tags": [],
    "title": "Restaurant in Athens has damage due to hurricane",
    "type": "hyperion_business_owners_story",
    "updatedAt": "2021-04-01T11:08:47.401Z",
    "weight": 1
}
```


Εικόνα 37: Ενα Hyperion Story έτσι όπως αυτό λαμβάνεται απο το API του PLUGGY και περιγράφεται από ένα αρχείο με δομή JSON.

3.2.2 Περιγραφή του προβλήματος

Στην αρχή της εργασίας αυτής αναφέρθηκαν οι στόχοι της. Ο πρώτος στόχος είναι η διερεύνηση του πεδίου της Μηχανικής Δεδομένων και συγκεκριμένα της κατασκευής σωληναγωγών δεδομένων (Data Pipelines). Ο δεύτερος στόχος είναι η εξέταση δημοφιλών Open-Source WfMS και ο τρίτος στόχος είναι η εφαρμογή του καταλληλότερου WfMS σε ένα πρόβλημα που αφορά το Hyperion Project.[WP6]

Στην προηγούμενη υποενότητα παρουσιάστηκε η μορφή με την οποία θα λαμβάνεται το Hyperion Story από το API του PLUGGY. Το middleware χρησιμοποιώντας το κατάλληλο Workflow Orchestrator θα λαμβάνει κάθε 24 ώρες τα Stories που κατέθεσαν οι χρήστες, τα οποία θα έχουν μορφή όπως της εικόνας 37. Αφού ληφθούν σε μορφή JSON, θα διατηρούνται μόνο οι απαραίτητες πληροφορίες, θα φτιάχνετε το περιεχόμενο της ιστορίας και θα δημιουργείται το τελικό αρχείο, το οποίο θα πρέπει να είναι συμβατό με τον Geoserver. Επιπλέον, το αρχείο θα ανεβαίνει στον τελικό προορισμό, ο οποίος είναι ο **GeoServer** [WP11]**.** Τέλος, τα Stories θα εισέρχονται και θα προβάλλονται μέσω του **Geonode** [WP9], αφού προηγουμένως το περιεχόμενο τους έχει εγκριθεί από διαχειριστές.



Εικόνα 38: Διάγραμμα περιγραφής της γενικής διαδικασίας.

GeoServer

Ο GeoServer[WP11] είναι ένας server, ο οποίος έχει αναπτυχθεί με την γλώσσα προγραμματισμού Java και επιτρέπει στους χρήστες να προβάλουν και να τροποποιούν γεωχωρικά δεδομένα. Χρησιμοποιεί προδιαγραφές σύμφωνα με το **Open Geospatial Consortium (OGC)** [WP12], ενώ επιτρέπει την δημιουργία χαρτών και τον διαμοιρασμό δεδομένων. Επιπλέον, ο GeoServer υποστηρίζει διάφορες μορφές δεδομένων, εκ των οποίων μερικές αναγράφονται στο διάγραμμα της Εικόνας 39.

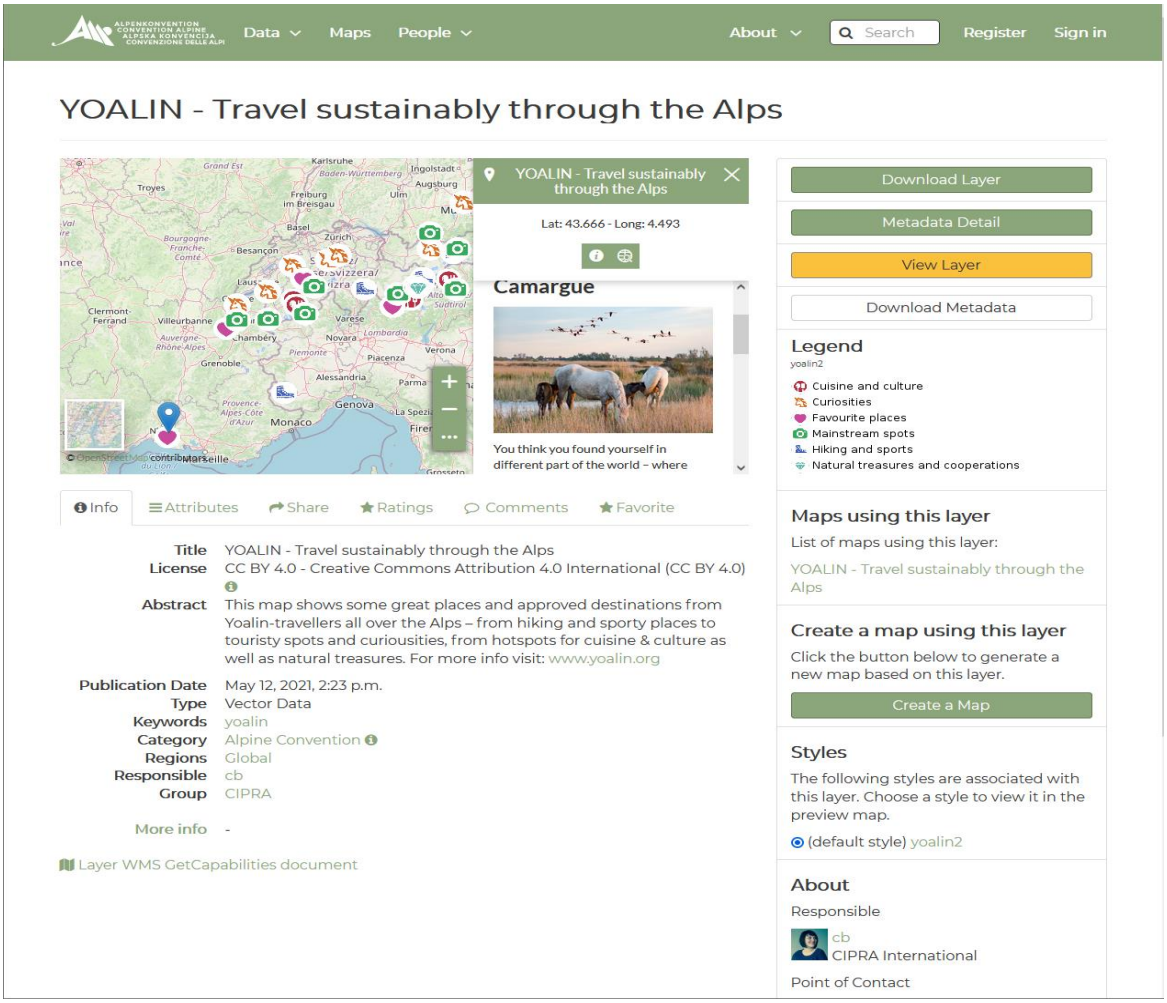
Εικόνα 39: Μερικές από τις δομές δεδομένων που υποστηρίζει ο GeoServer.

Geonode

Σύμφωνα με την επίσημη σελίδα του Geonode [WP9], *"το GeoNode είναι ένα σύστημα διαχείρισης γεωχωρικού περιεχομένου, μια πλατφόρμα για τη διαχείριση και δημοσίευση γεωχωρικών δεδομένων. Εμπεριέχει προγράμματα λογισμικού ανοιχτού κώδικα σε μια συνεπή και εύχρηστη διεπαφή που επιτρέπει σε μη εξειδικευμένους χρήστες να μοιράζονται δεδομένα και να δημιουργούν διαδραστικούς χάρτες."*

"Τα εργαλεία διαχείρισης δεδομένων που είναι ενσωματωμένα στο GeoNode επιτρέπουν την ολοκληρωμένη δημιουργία δεδομένων, μεταδεδομένων και οπτικοποίησης χάρτη. Κάθε σύνολο δεδομένων στο σύστημα μπορεί να κοινοποιηθεί δημόσια ή να περιοριστεί ώστε να επιτρέπεται η πρόσβαση σε συγκεκριμένους μόνο χρήστες. Τα κοινωνικά χαρακτηριστικά όπως τα προφίλ χρηστών, τα σχόλια και τα συστήματα αξιολόγησης επιτρέπουν την ανάπτυξη κοινοτήτων σε κάθε πλατφόρμα για τη διευκόλυνση της χρήσης, της διαχείρισης και του ποιοτικού ελέγχου των δεδομένων που περιέχει η παρουσία GeoNode."

"Έχει επίσης σχεδιαστεί για να είναι μια ευέλικτη πλατφόρμα την οποία οι προγραμματιστές λογισμικού μπορούν να επεκτείνουν, να τροποποιήσουν ή να ενσωματώσουν ώστε να πληρούν τις απαιτήσεις στις δικές τους εφαρμογές. "



Εικόνα 40: Παράδειγμα διαδραστικού χάρτη στο Geonode.[WP10]

3.2.3 Επιλογή του κατάλληλου Workflow Orchestrator

Για την επίλυση του προβλήματος θα πρέπει να χρησιμοποιηθεί ένα WfMS, με το οποίο θα διαχειρίζεται και θα επιβλέπεται η ροή των διεργασιών. Το επικρατέστερο WfMS ήταν το Apache Airflow. Οι λόγοι που το Apache Airflow επικράτησε είναι ιδιαίτερα σημαντικοί. Σε αντίθεση με το Luigi, το Apache Airflow περιλαμβάνει έναν Scheduler, ο οποίος είναι απαραίτητος για το USE CASE του HCET, αφού επιτρέπει την εκτέλεση αυτοματοποιημένων διαδικασιών σε τακτά χρονικά διαστήματα και την επανεκτέλεση των διεργασιών σε περίπτωση που χρειαστεί. Αντιθέτως, το Luigi δεν περιλαμβάνει κάποιον Scheduler και βασίζεται εξ ολοκλήρου σε cron jobs για να προγραμματίσει τις διεργασίες. Επιπλέον, στο Apache Airflow οι ροές διεργασιών (DAGs of tasks) ορίζονται στην γλώσσα python, η οποία

είναι μια γλώσσα υψηλού επιπέδου, προσφέροντας έτσι ένα ευέλικτο μέσο περιγραφής και συγγραφής των ροών διεργασιών, καθώς παρέχει σημαντικά εργαλεία, τα οποία μπορεί να χρησιμοποιήσει κανείς για να επικοινωνήσει π.χ. με APIs και Δομές Δεδομένων. Επιπρόσθετα, το Apache Airflow είναι το μόνο από τα εξεταζόμενα WfMS, το οποίο περιλαμβάνει την λειτουργία Backfilling. Το Backfilling είναι ένα πολύ ισχυρό χαρακτηριστικό του Apache Airflow το οποίο δίνει την δυνατότητα στον χρήστη να εκτελέσει παρελθοντικές διεργασίες, οι οποίες είχαν αποτύχει να εκτελεστούν στο παρελθόν. Μια ακόμη διαφορά του Airflow με το Luigi και το Temporal είναι η χρήση των Operators του πρώτου. Οι Operators καθιστούν πολύ εύκολο το να συγγράψει κανείς DAGs, ενώ το Airflow δίνει στον χρήστη την επιλογή να γράψει τους δικούς του Operators, χαρακτηριστικό που οδήγησε το Community του Apache Airflow στο να γράψει Operators που υποστηρίζουν δεκάδες εργαλεία και υπηρεσίες. Ένας ακόμη λόγος για την επιλογή του Apache Airflow ήταν το ιδιαίτερο φιλικό προς τον χρήστη User Interface, το οποίο δίνει την δυνατότητα στον χρήστη να επιβλέπει την εκτέλεση των DAGs, καθώς και άλλες δυνατότητες όπως να σταματήσει την εκτέλεση κάποιου DAG, να εξετάσει στο log για ποιον λόγο ένα Task απέτυχε κ.α. Σε σύγκριση με το User Interface του Luigi και του Temporal που είναι αρκετά απλοϊκά, το User Interface του Airflow παρέχει πολλά εργαλεία για να επιβλέπει ο χρήστης τις τρέχοντες διεργασίες καθώς και την ανταλλαγή των μεταξύ τους δεδομένων.

Τέλος, ίσως ο σημαντικότερος παράγοντας που έπαιξε καθοριστικό ρόλο στο να επιλέξω το Apache Airflow, ήταν το τεράστιο Community που το υποστηρίζει σε σχέση με το Luigi και το Temporal. Εκτός αυτού, το υλικό που υπάρχει διαθέσιμο, εκτός από τα επίσημα documentation των Temporal και Luigi, είναι αρκετά περιορισμένο. Σε αντίθεση με αυτό του Apache Airflow που μπορεί κανείς να βρει βιβλία και αρκετές πηγές στο διαδίκτυο.

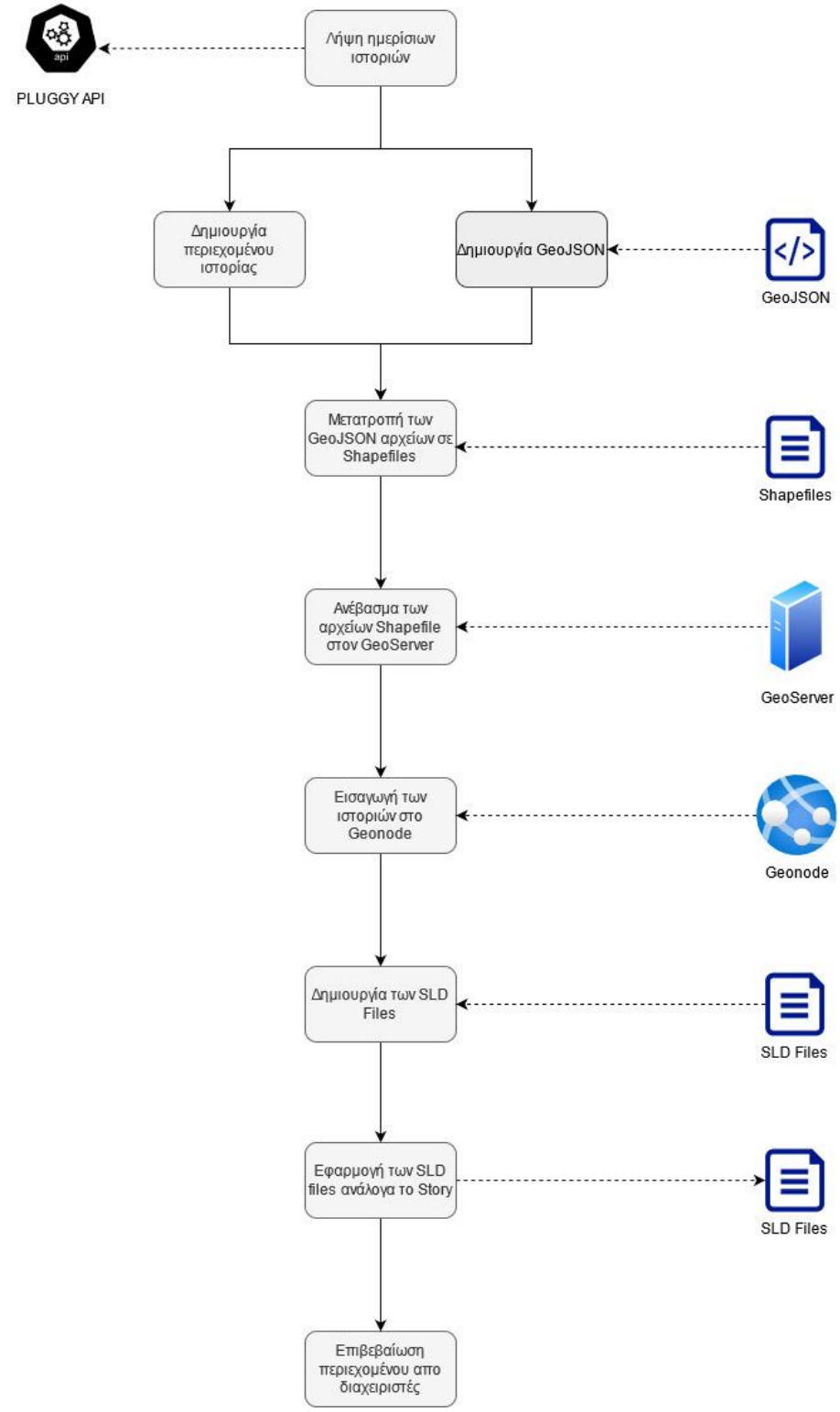
3.2.4 Προσέγγιση του προβλήματος

Για τον σχεδιασμό του σωληναγωγού δεδομένων θα πρέπει να διευκρινιστούν κάποια ιδιαίτερα χαρακτηριστικά του προβλήματος. Η **πηγή** των δεδομένων είναι το PLUGGY API από το οποίο θα εξάγονται τα Stories της ημέρας σε μορφή **JSON** μέσω εκτέλεσης της κατάλληλης http request εντολής. Ο **προορισμός** είναι ο GeoServer, ο οποίος δέχεται, εκτός των άλλων, και τα αρχεία της εικόνας 39. Τα ShapeFiles που θα αντιστοιχούν σε κάθε ιστορία

και θα πρέπει να βρίσκονται κάπου αποθηκευμένα για να μπορεί έπειτα ο GeoServer να έχει πρόσβαση σε αυτά.

Αρχικά, υπήρχαν σκέψεις τα αρχεία αυτά να αποθηκεύονται σε μία Βάση Δεδομένων, η οποία θα είναι εγκατεστημένη στον μηχάνημα που φιλοξενεί και τον GeoServer, ωστόσο αντ' αυτού αποφασίστηκε να αποθηκευτούν στο File System του ίδιου μηχανήματος. Εφόσον δεν υπάρχουν διαφορετικά είδη αρχείων με περίπλοκες συσχετίσεις μεταξύ τους ή ανάγκη να δημιουργηθούν αυστηρά δομημένοι πίνακες με πολλαπλά πεδία, θεωρήθηκε ότι το να αποθηκευτούν τα αρχεία ShapeFiles σε μία βάση θα προσέθετε επιπλέον πολυπλοκότητα στο υπάρχον σύστημα αλλά και χρονική καθυστέρηση. Για αυτό τον λόγο αποφασίστηκε τα αρχεία Shapefiles να αποθηκεύονται στο File System του μηχανήματος που φιλοξενεί τον GeoServer. Συνεπώς, η εκτέλεση του συνόλου των διεργασιών του σωληναγωγού δεδομένων θα πρέπει να είναι σε θέση να πραγματοποιήσει τα ακόλουθα:

1. Να εξάγει τα ημερήσια Stories από το API του PLUGGY
2. Να διακρίνει ποια ιστορία αφορά ιστορία ενός πολίτη και ποια αφορά ιστορία μιας επιχείρησης
3. Να κατασκευάζει το περιεχόμενο της ιστορίας ανάλογα με τον τύπο της ιστορίας (Citizen Story/Business Story)
4. Να δημιουργεί την δομή της ιστορίας βασιζόμενη σε κάποιον από τους συμβατούς τύπους αρχείων που υποστηρίζει ο GeoServer (Εικόνα 39)
5. Να διαθέτει τα αρχεία των ιστοριών στον GeoServer
6. Να ανεβάζει το αντίστοιχο style (αρχείο SLD) για τον κάθε τύπο στον GeoServer
7. Να εφαρμόζει το κατάλληλο SLD σε μία ιστορία ανάλογα με τον τύπο της ιστορίας
8. Να δημοσιεύει τις ιστορίες στο Geonode
9. Να θέτει τα κατάλληλα δικαιώματα προβολής της ιστορίας ώστε οι ιστορίες να μπορούν να προβληθούν μόνο από διαχειριστές, έτσι ώστε οι διαχειριστές να είναι σε θέση να εγκρίνουν το περιεχόμενο της ιστορίας προτού αυτές δημοσιευθούν στο ευρύ κοινό



Εικόνα 41: Πρόχειρο διάγραμμα ροής εργασίας με τις διεργασίες του σωληναγωγού.

Όπως διαπιστώνεται από την περιγραφή του προβλήματος, μία προσέγγιση για την επίλυση του προβλήματος θα ήταν να σχεδιαστεί ένας σωληναγωγός (Pipeline), ο οποίος θα χρησιμοποιεί το μοτίβο ETL και θα περιέχει όλες τις απαραίτητες διεργασίες, οι οποίες θα περιγράφονται σε ένα DAG και θα εκτελούνται ανά συγκεκριμένο χρονικό διάστημα από το Apache Airflow. Παρακάτω περιγράφονται οι διεργασίες, οι οποίες θα εκτελούνται και θα περιέχονται στο DAG:

Διεργασία 1

Η διεργασία 1 είναι υπεύθυνη για να λάβει τις νέες ιστορίες από το API του PLUGGY και την υλοποιεί ένας SimpleHttpOperator, ο οποίος εκτελεί την μέθοδο αίτησης HTTP GET με ορίσματα τις παραμέτρους Hyperion Story και την τρέχον ημερομηνία. Πρότεινα στην ομάδα του ICCS [WP21] που είναι υπεύθυνη για τον Wizard δημιουργίας ιστοριών, όταν δημιουργείται μία νέα ιστορία από τον χρήστη να επισυνάπτονται αυτόματα στην ιστορία τα tags Hyperion Story και η ημερομηνία δημιουργίας της ιστορίας, έτσι ώστε να διαχωρίζονται εύκολα τα Hyperion Story, από όλα τα άλλα exhibitions του PLUGGY. Το αποτέλεσμα της μεθόδου GET, το οποίο είναι σε μορφή JSON αποθηκεύεται σε μία XCOM, η οποία είναι μία δομή που μοιάζει με hashmap (JAVA) ή dictionary (Python) και είναι μία από τις λύσεις που παρέχει το Airflow για την ανταλλαγή δεδομένων μεταξύ των στιγμιότυπων των διεργασιών σε ένα DAG run.

Διεργασία 2

Την διεργασία 2 υλοποιεί ένας PythonOperator, ο οποίος καλεί μία μέθοδο που φιλτράρει από κάθε ληφθείσα ιστορία σημαντικά δεδομένα όπως τις συντεταγμένες της ιστορίας, τον τύπο της ιστορίας (Citizen/Business Story) και ένα id (μοναδικό αναγνωριστικό ιστορίας) και τα οποία είναι απαραίτητα για να κατασκευάσει ένα αρχείο GeoJSON, το οποίο περιέχει κάποια χαρακτηριστικά όπως συντεταγμένες και τον τύπο γεωμετρίας, ο οποίος στην περίπτωση των Hyperion Stories είναι απαραίτητο να είναι Point, επειδή οι ιστορίες θα σημειώνονται με σημεία πάνω στον χάρτη. Επέλεξα να κατασκευάσω ένα GeoJSON αρχείο, διότι είναι μία κατανοητή δομή που μπορεί να κατασκευαστεί εύκολα και να μετατραπεί σε ShapeFile (.SHP), δηλαδή ένα από τα αρχεία που είναι συμβατά με τον Geoserver.

Διεργασία 3

Η διεργασία 3 εκτελείται παράλληλα με την διεργασία 2 επειδή δεν υπάρχουν αλληλεξαρτήσεις μεταξύ τους. Σκοπός αυτής της διεργασίας είναι να κατασκευάσει το περιεχόμενο της ιστορίας. Πιο συγκεκριμένα όταν ο χρήστης πατάει την πινέζα της ιστορίας στον χάρτη θα αναδύεται ένα παράθυρο, το οποίο θα περιέχει το περιεχόμενο της ιστορίας.(Βλέπε 3.2.1)

Για την κατασκευή κάθε ιστορίας, λαμβάνεται κάθε ιστορία που έχει ληφθεί μέσω της διεργασίας 1 και κρατούνται από αυτήν οι απαραίτητες πληροφορίες, όπως τίτλος ιστορίας, εικόνες, ετικέτες και άλλα, καθώς και επιπλέον πληροφορίες (βλέπε 3.2.1) αν η ιστορία αφορά μια επιχείρηση (Business Story), όπως και εικόνες που αφορούν την καταστροφή. Τέλος, όλη η ιστορία κωδικοποιείται με URL Encoding.

Διεργασία 4

Η διεργασία 4 υλοποιείται μέσω ενός BashOperator και είναι υπεύθυνη για την μετατροπή των αρχείων από GeoJSON σε αρχεία ShapeFiles (.SHP). Οι ιστορίες προς το παρόν είναι αποθηκευμένες σε μορφή GeoJSON και είναι χωρισμένες σε φακέλους ανάλογα με την ημερομηνία λήψης και το id τους. Η διεργασία 4 εκτελεί ένα Bash Script και μετατρέπει με χρήση της βιβλιοθήκης της GDAL [WP13] κάθε αρχείο GeoJSON σε αρχείο ShapeFile, το οποίο αποτελείται από 4 αρχεία, τα οποία έχουν τις καταλήξεις .shp (αρχείο που περιέχει τις γεωμετρίες της ιστορίας), .dbf (αρχείο που περιέχει τα attributes της ιστορίας), .shx (αρχείο δείκτης προς την γεωμετρία της ιστορίας) και την κατάληξη .prj (αρχείο που περιέχει πληροφορίες για τον μορφότυπο της προβολής, όπως για παράδειγμα το σύστημα συντεταγμένων)

Διεργασία 5 & διεργασία 6

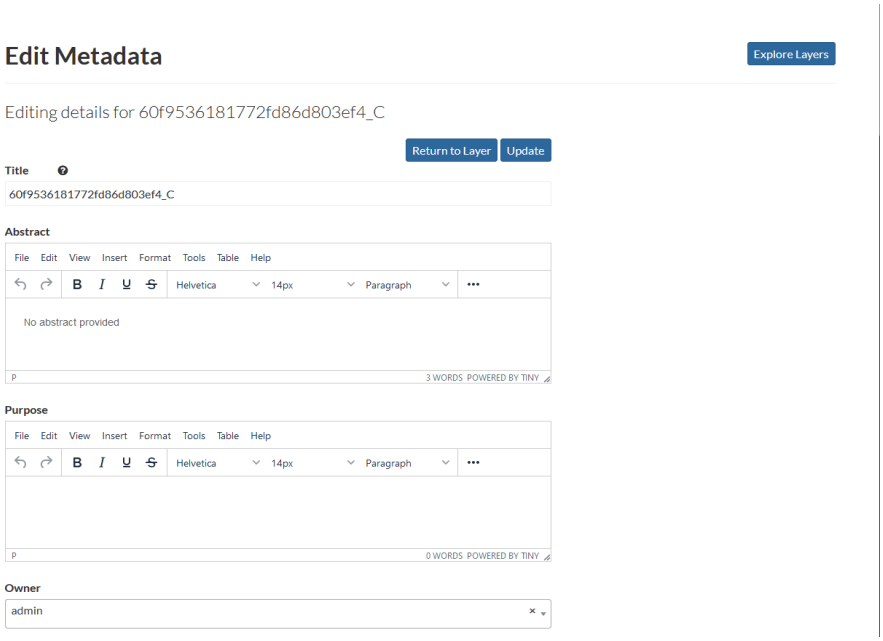
Πλέον τα αρχεία ShapeFiles είναι διαθέσιμα και η διεργασία 5 καλείται να τα ανεβάσει στον GeoServer Server. Για να το πετύχω αυτό κατασκεύασα έναν Bash Operator, οποίος τρέχει την εντολή rsync σε συνδυασμό με την εντολή ssh για πιστοποίηση του χρήστη με χρήση δημοσίου κλειδιού και ανεβάζει στον GeoServer τα Shapefiles, αφού πρώτα τα συμπιέσει. Η διεργασία 6 αποσυμπιέζει τις ιστορίες στον GeoServer και καθαρίζει το local directory (Airflow) και το remote directory (GeoServer) από περιττά αρχεία και φακέλους.

Διεργασία 7

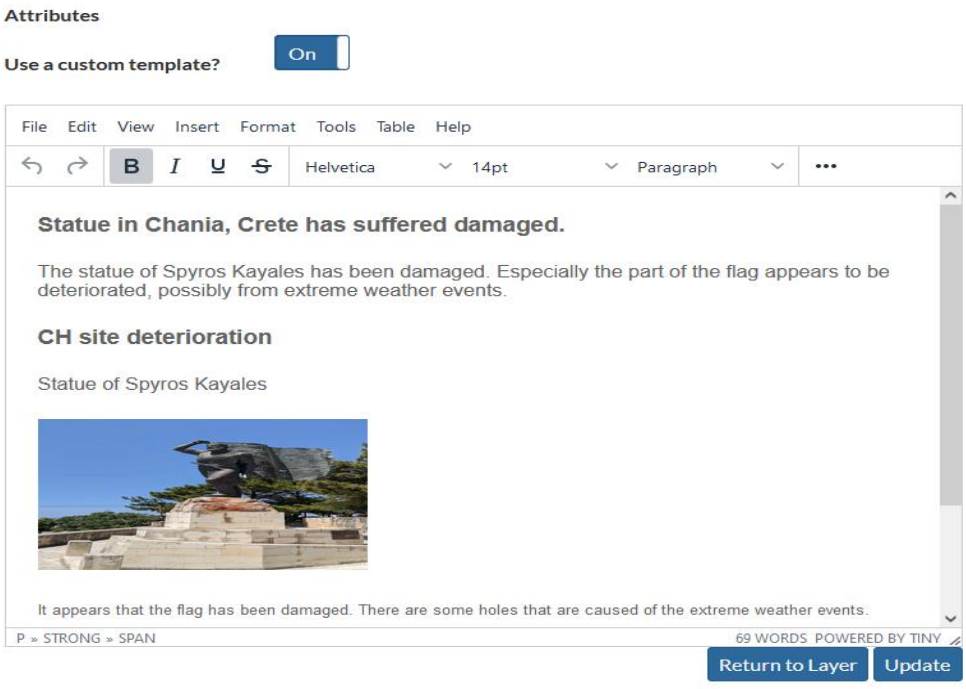
Ο GeoServer παρέχει μία δομή που ονομάζεται Workspace και ουσιαστικά χρησιμοποιείται για να ομαδοποιήσει layers (Stories στην περίπτωση του Hyperion Project). Επίσης, ο GeoServer χρησιμοποιεί ένα mapping file που ονομάζεται Datastore, το οποίο δείχνει σε μία πηγή δεδομένων. Η διεργασία 7 δημιουργεί μέσω του API του GeoServer ένα Workspace με το όνομα hyperion και ένα Datastore, το οποίο είναι τύπου Directory of Spatial Files και δείχνει στον φάκελο που βρίσκονται τα ShapeFiles στον GeoServer. Τόσο ο GeoServer τόσο και το Geonode τρέχουν σε δύο ξεχωριστά Docker [WP53] containers. Για να δημοσιευτούν οι ιστορίες από τον GeoServer στο Geonode εκτελεί η διεργασία 7 στο container του Geonode την εντολή update layers, η οποία δημοσιεύει τα νέα layers (Stories) που ανέβηκαν στον GeoServer στο Geonode.

Διεργασία 8

Η διεργασία 8 είναι μία από τις πιο σημαντικές διεργασίες. Στην διεργασία 3 ανέφερα πως κατασκευάζεται το περιεχόμενο της ιστορίας που βλέπει ο χρήστης. Κανονικά, για να γράψει κάποιος το περιεχόμενο της ιστορίας θα πρέπει να το κάνει μέσω του Geonode και ενός html editor που βρίσκεται στην σελίδα advanced metadata του layer. Δυστυχώς, στο Geonode API δεν βρήκα κάποιο end-point με το οποίο μπορεί κανείς να ενημερώσει τα περιεχόμενα της σελίδας metadata advanced.



Εικόνα 42: Κάποια από τα πεδία της σελίδας advanced metadata.



Εικόνα 43: Ο html editor στον οποίο συντάσσεται το περιεχόμενο της ιστορίας.

Για αυτόν τον λόγο φίλτραρα το http πακέτο που μεταδίδεται όταν γίνεται ενημέρωση των advanced metadata μέσω του εργαλείου network inspector που είναι εγκατεστημένο στον Mozilla Firefox [WP54] και το κατασκεύασα σε κώδικα Python.

Εικόνα 44: Το πακέτο που ενημερώνει τα metadata της ιστορίας μέσω της μεθόδου POST.

Συνεπώς, η διεργασία 8 στην αρχή της εκτέλεσής της πραγματοποιεί μια σύνδεση με το Geonode χρησιμοποιώντας το csrftoken και το sessionid και έπειτα για κάθε ιστορία κατασκευάζει το περιεχόμενο της μεθόδου POST που θα σταλεί για να γίνει ενημέρωση των metadata της ιστορίας. Αφού κατασκευαστεί το περιεχόμενο του POST, εκτελείται μία μέθοδος POST, η οποία φέρει τα δεδομένα που κατασκευάστηκαν και ενημερώνει τα metadata της ιστορίας.

Εικόνα 45: Ένα κομμάτι από το περιεχόμενο του POST request και συγκεκριμένα το περιεχόμενο του Custom HTML Editor.

Στην συνέχεια, ανάλογα με τον τύπο της ιστορίας, αν δηλαδή η ιστορία αφορά ιστορία πολίτη ή ιστορία επιχείρησης, φορτώνεται στον Geoserver το αντίστοιχο SLD αρχείο και εφαρμόζεται στην ιστορία.

Εικόνα 46: Η πινέζα που αφορά τα Citizen Stories.

Εικόνα 47: Η πινέζα που αφορα τα Business Stories.

Τέλος, όταν μία ιστορία δημοσιεύεται στο Geonode πρέπει πρώτα το περιεχόμενό της να ελέγχεται και στην συνέχεια να εγκρίνεται ή να απορρίπτεται από τους διαχειριστές. Για αυτό τον λόγο στο τέλος της διεργασίας 8 εκτελείται μία μέθοδος POST που ενημερώνει τα δικαιώματα της ιστορίας, έτσι ώστε να εμφανίζεται αρχικά μόνο σε διαχειριστές.

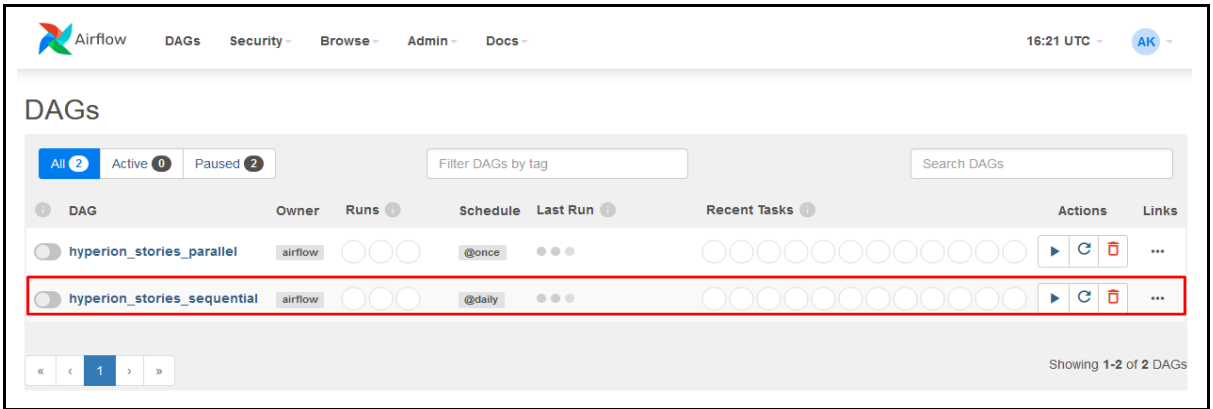
3.2.5 Εκτέλεση του DAG στο Airflow

Όπως αναφέρθηκε στο κεφάλαιο 2.1.3 το Airflow μπορεί να χρησιμοποιήσει διάφορους Executors για να εκτελέσει τις διεργασίες που βρίσκονται στην ουρά. Αρχικά, θα χρησιμοποιήσω τον Sequential Executor, ο οποίος είναι ο προεπιλεγμένος executor κατά την εγκατάσταση του Apache Airflow.
Κατα την εκκίνηση της διαδικασίας θα πρέπει να ξεκινήσει ο webserver του Apache Airflow και ο scheduler.

```
airflow webserver
airflow scheduler
```

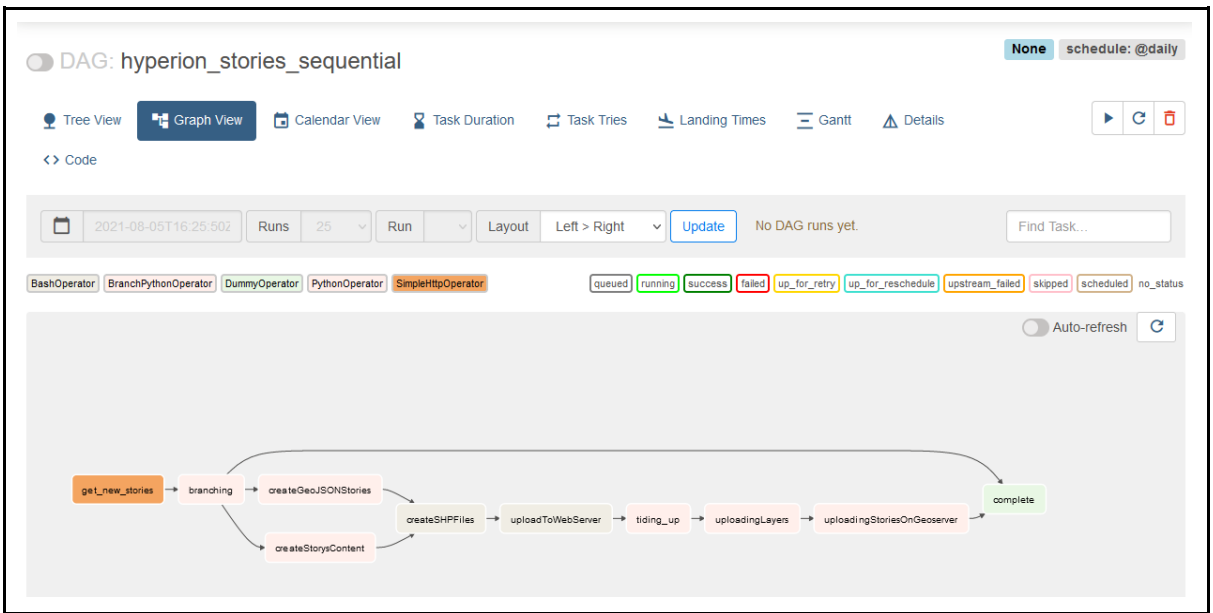
Αρχικά, το DAG εμφανίζεται στην λίστα με τα DAGs στο dashboard του πίνακα σαν απενεργοποιημένο. Έχω προγραμματίσει το DAG να τρέξει τρεις φορές, για τις ημερομηνίες

18-7-2021, 19-7-2021 και 20-7-2021, γνωρίζοντας ότι στις 18-7-2021 δεν ανέβηκε καμία ιστορία, στις 19-7-2021 ανέβηκε μια ιστορία και στις 20-7-2021 ανέβηκαν 2 ιστορίες.



Εικόνα 48: Το DAG φορτώθηκε κατά την εκκίνηση του scheduler.

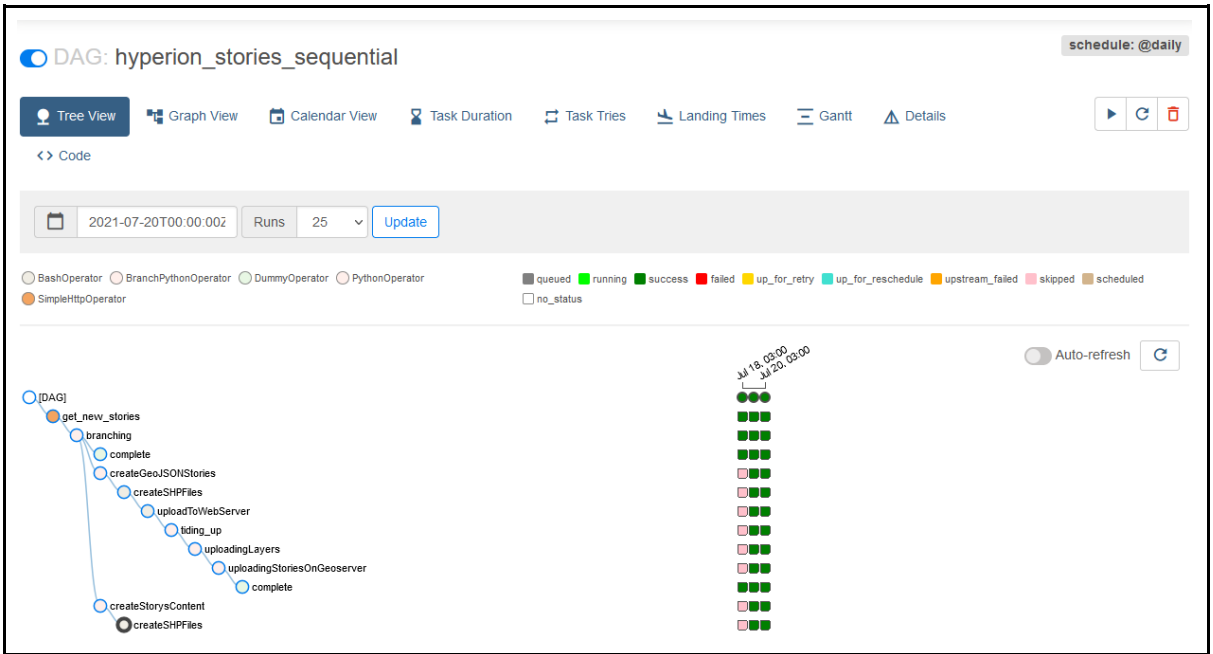
Το διάγραμμα ροής διεργασιών με το σύνολο των διεργασιών του και τις αλληλοεξαρτήσεις του εμφανίζεται με την μορφή ενός κατευθυνόμενου άκυκλου γράφου στην εικόνα 49.



Εικόνα 49: Το DAG που πρόκειται να εκτελεστεί.

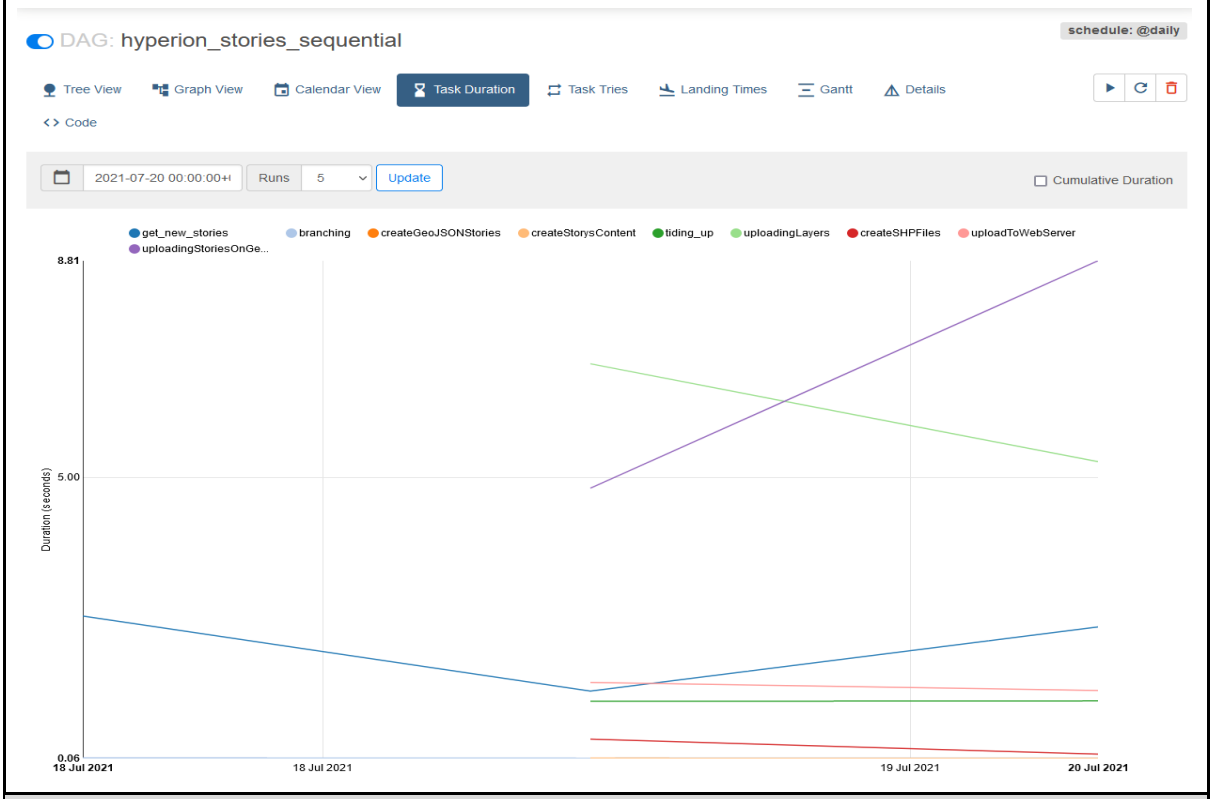
Τα αποτελέσματα για το τρέξιμο του DAG για κάθε μια μέρα φαίνεται στην εικόνα 50. Σε αυτήν διακρίνεται αριστερά το σύνολο των διεργασιών, το οποίο παρουσιάζεται σε δενδρική

μορφή. Δεξιά κάθε στήλη αντιπροσωπεύει ένα DAG και τα κουτάκια αντιστοιχούν σε κάθε μία διεργασία του DAG. Την πρώτη μέρα εμφανίζονται κάποια ροζ κουτάκια επειδή αυτές οι διεργασίες παραλήφθηκαν λόγω του ότι δεν υπάρχουν διαθέσιμες ιστορίες την ημέρα αυτή και συνεπώς κάποιες διεργασίες δεν εκτελέστηκαν. Τις υπόλοιπες δυο μέρες βρέθηκαν νέες ιστορίες, οπότε εκτελέστηκαν όλες οι διεργασίες του DAG.

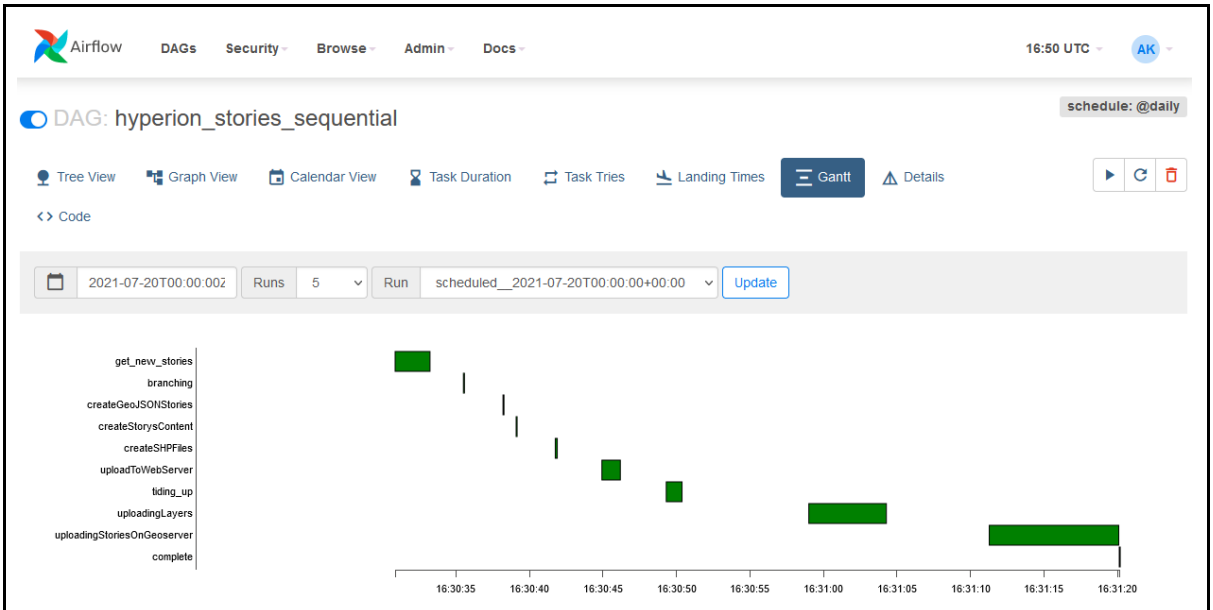


Εικόνα 50: Τα αποτελέσματα του DAG που έτρεξε για κάθε μία από τις τρεις μέρες.

Ένας από τους σημαντικούς λόγους όπως προανέφερα που διάλεξα το Apache Airflow για την υλοποίηση αυτού του Use Case, ήταν το γεγονός ότι παρέχει πολλά εργαλεία στον σχεδιαστή του σωληναγωγού για να παρακολουθήσει την πορεία των διεργασιών. Τέτοια εργαλεία φαίνονται στις εικόνες 51 και 52. Στην εικόνα 51 φαίνεται ο χρόνος εκτέλεσης κάθε διεργασίας, ενώ στην εικόνα 52 φαίνεται το διάγραμμα GANTT που δείχνει τις εκτελέσεις κάθε διεργασίας, αν εκτελούνται παράλληλα ή ακολουθιακά και σε ποιο διάστημα χρόνου.



Εικόνα 51: Η διάρκεια εκτέλεσης κάθε διεργασίας.



Εικόνα 52: Διάγραμμα GANTT που δείχνει την εκτέλεση κάθε διεργασίας.

3.2.6 Διαχείριση αποτυχημένης εκτέλεσης

Το Apache Airflow εκτός από το να εκτελεί και να επιβλέπει τις διεργασίες που εκτελούνται προσφέρει στον χρήστη και άλλα σημαντικά εργαλεία όπως για παράδειγμα έναν μηχανισμό που ενημερώνει τον Μηχανικό Δεδομένων μέσω email σε περίπτωση που κάποια διεργασία αποτύχει να εκτελεστεί. Για να χρησιμοποιήσει κανείς την λειτουργία αυτή θα πρέπει να συμπληρωθούν στο αρχείο airflow.cfg οι απαραίτητες ρυθμίσεις αλλά και να προστεθεί ένα καινούργιο όρισμα στον ορισμό του DAG, το οποίο θα δίνει οδηγία να σταλεί ένα ενημερωτικό email σε περίπτωση που το DAG αποτύχει να εκτελεστεί.

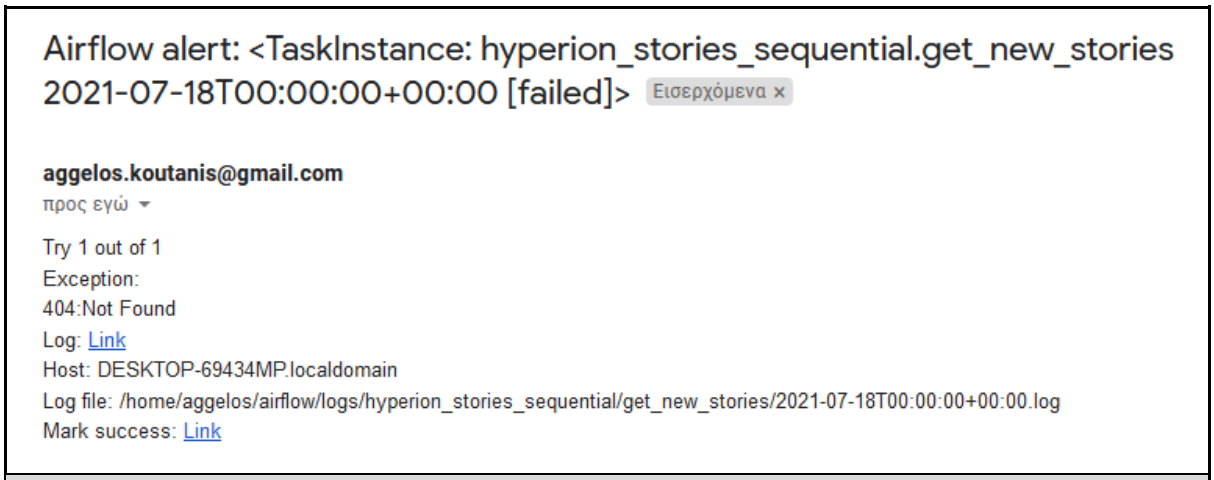
```
args = {
    'email': ['aggelos.koutanis@gmail.com'],
    'email_on_failure': True}
dag = DAG(
    dag_id="hyperion_stories_sequential",
    start_date= datetime(2021, 7, 18),
    # schedule_interval='*/1 * * * *',
    schedule_interval='@daily',
    catchup=True,
    end_date=datetime(2021, 7, 18),
    default_args=args)
```

Εικόνα 53: Ορισμός της λειτουργίας ενημέρωσης του ΜΔ σε περίπτωση αποτυχίας εκτέλεσης.

```
smtp_host =  smtp.gmail.com
smtp_starttls = True
smtp_ssl = False
smtp_user = aggelos.koutanis@gmail.com
smtp_password = **********
smtp_port = 587
smtp_mail_from =  aggelos.koutanis@gmail.com
smtp_timeout = 30
smtp_retry_limit = 5
```

Εικόνα 54: Απαραίτητες ρυθμίσεις στο Airflow.cfg.

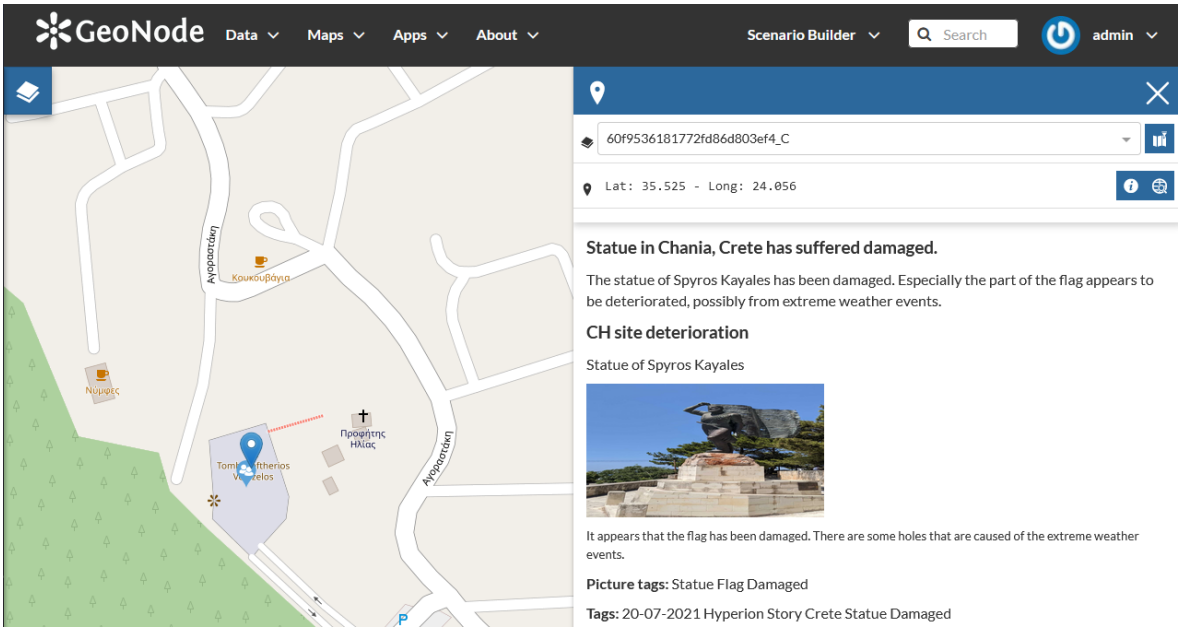
Για να ελέγξω την λειτουργία ενημέρωσης μέσω ηλ. ταχυδρομείου τροποποίησα ελαφρώς την πρώτη διεργασία του DAG αλλάζοντας την διεύθυνση του API από το οποίο λαμβάνω τις ιστορίες, έτσι ώστε το DAG να αποτυγχάνει κατά την εκτέλεση της πρώτης διεργασίας. Όπως φαίνεται και στην εικόνα 55, το περιεχόμενο του email ενημερώνει σε ποια προσπάθεια εκτέλεσης παρουσιάστηκε το σφάλμα, το είδος του σφάλματος, το οποίο στην προκειμένη περίπτωση ήταν 404: Not Found. Επιπλέον, παρέχει στον ΜΔ έναν σύνδεσμο που οδηγεί στο log της διεργασίας που βρίσκεται στον web server του Airflow, το όνομα του host, το file path στο οποίο βρίσκεται το Log file και τέλος, έναν σύνδεσμο μέσω του οποίου μπορεί να αλλαχθεί το status της διεργασίας από failed σε success.



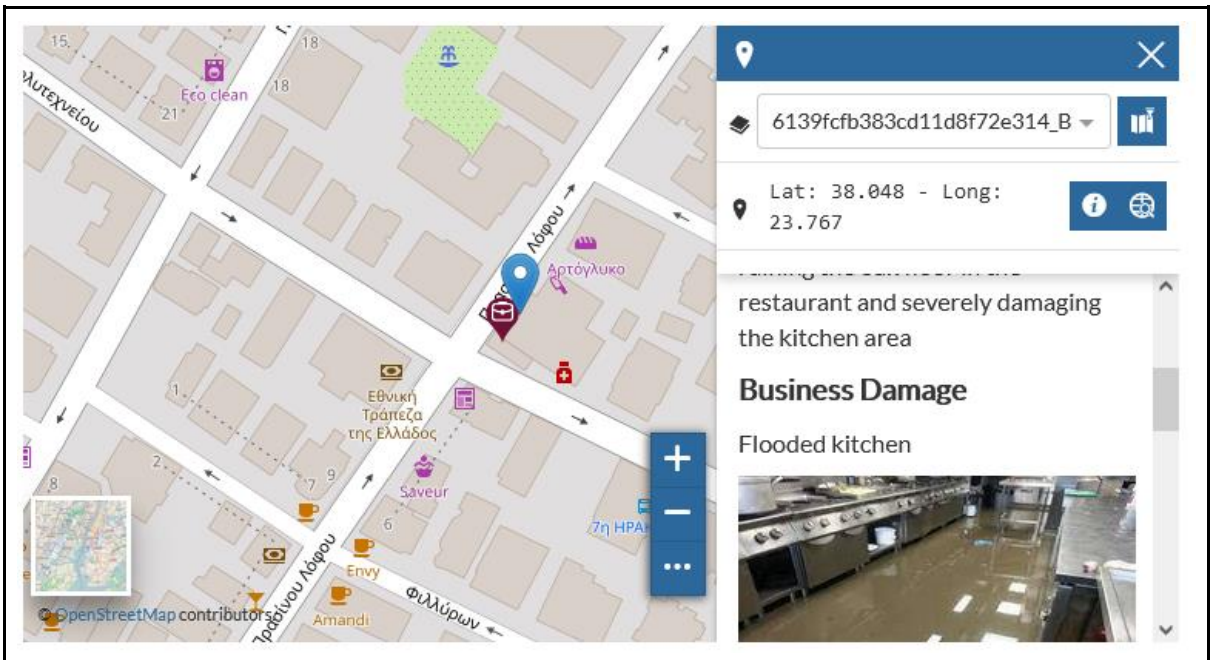
Εικόνα 55: Το email που ενημερώνει τον ΜΔ για την αποτυχία της διεργασίας.

3.2.7 Αποτελέσματα

Το αποτέλεσμα μιας επιτυχημένης εκτέλεσης του DAG είναι να δημοσιευτούν οι ιστορίες για τις συγκεκριμένες ημέρες στο Geonode. Η ιστορία προς το παρόν είναι διαθέσιμη μόνο στους διαχειριστές μέχρι να εγκρίνουν το περιεχόμενό της. Για να δει κανείς τις ιστορίες πρέπει να επισκεφτεί την σελίδα που φιλοξενεί το Geonode και να επιλέξει να δει τα ανεβασμένα Layers. Η κάθε ιστορία (Layer) ανεβαίνει μεμονωμένα στην λίστα με τα Layers και προβάλλονται στον χρήστη όπως φαίνεται στις εικόνες 56 και 57.



Εικόνα 56: Δημοσιευμένη ιστορία πολίτη.

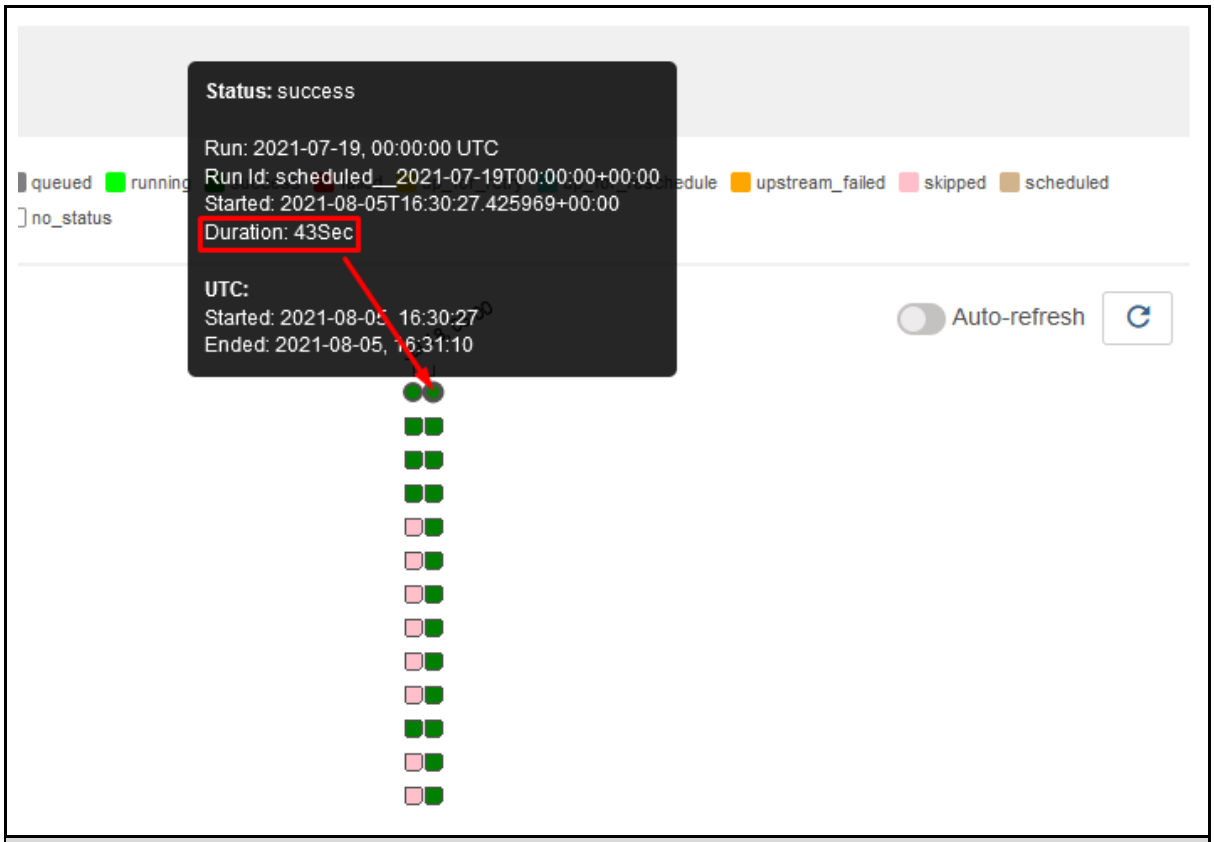


Εικόνα 57: Δημοσιευμένη ιστορία επιχείρησης.

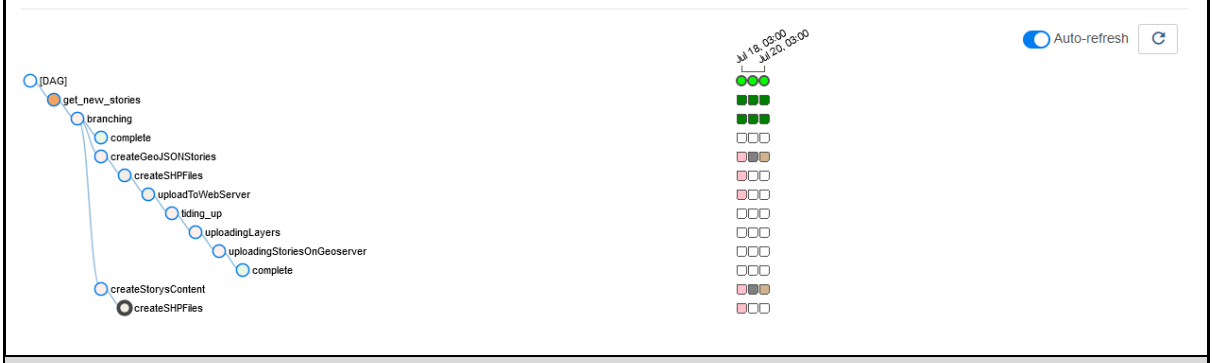
3.2.8 Πιθανές βελτιώσεις

3.2.8.1 Αλλαγή Executor

Όπως φαίνεται στην εικόνα 58, η εκτέλεση του DAG την 19-07-2021 χρειάστηκε 43 δευτερόλεπτα για να εκτελεστεί. Κάθε DAG δεν εκτελείται παράλληλα για κάθε ημερομηνία, αλλά ακολουθιακά (Εικόνα 59) και το ίδιο ισχύει για τις διεργασίες μέσα στο DAG δηλαδή εκτελούνται και αυτές ακολουθιακά, παρόλο που στο DAG φαίνεται να εκτελούνται παράλληλα. (Εικόνα 60).



Εικόνα 58: Διάρκεια εκτέλεσης του DAG την 19-07-2021.

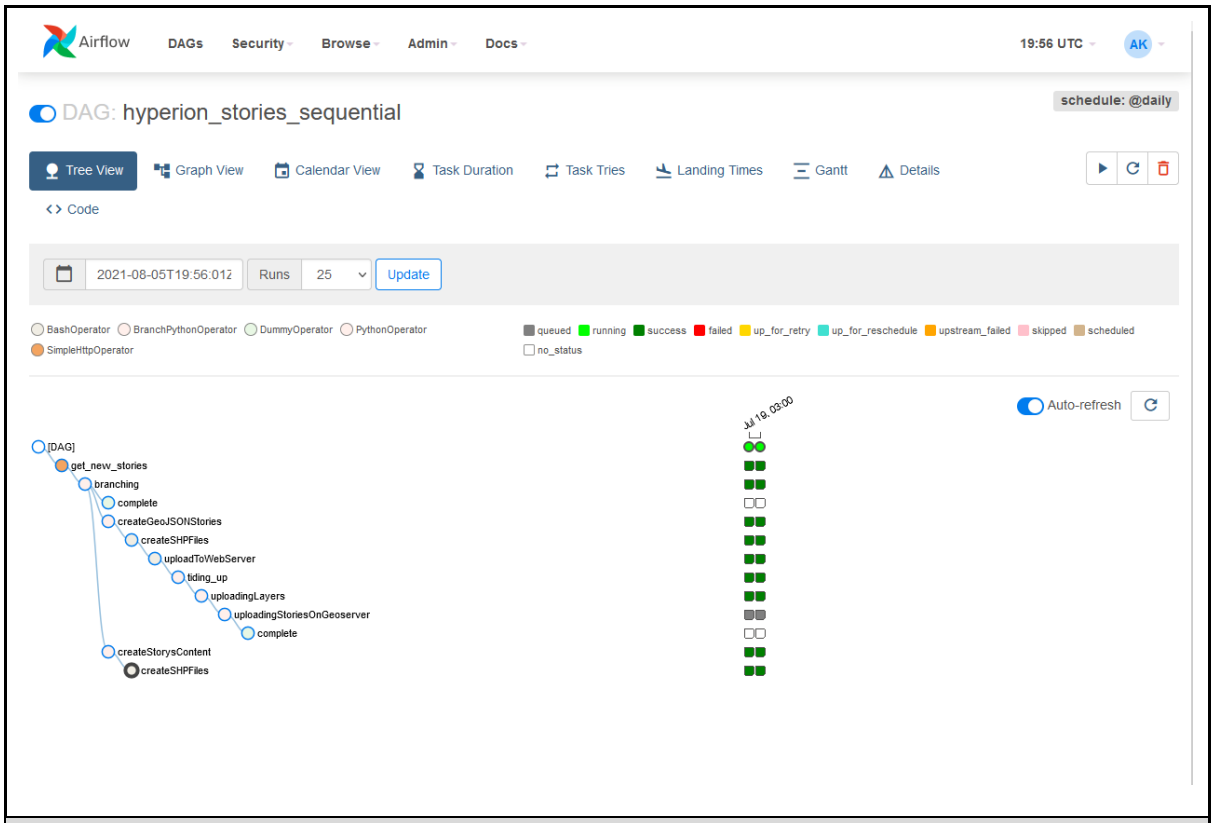


Εικόνα 59: Τα DAGs εκτελούνται ακολουθιακά και όχι παράλληλα.

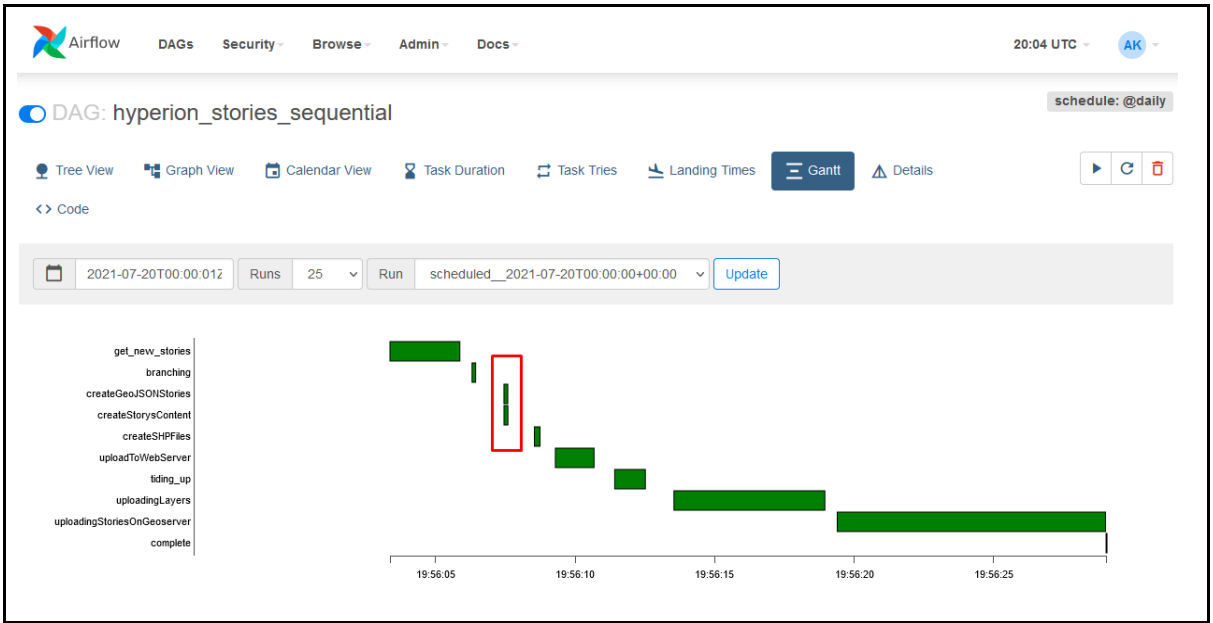
Εικόνα 60: Οι διεργασίες παρόλο που εμφανίζονται παράλληλες στο DAG, δεν εκτελούνται ταυτόχρονα.

Οπότε μία πιθανή βελτίωση θα ήταν αντί να χρησιμοποιήσω τον Sequential Executor, ο οποίος σύμφωνα με το επίσημο documentation του Apache Airflow [WP17] πρέπει να χρησιμοποιείται μόνο για testing, να χρησιμοποιήσω τον Local Executor. Έτσι, θα μπορέσω να τρέξω παράλληλα διεργασίες και να εξοικονομήσω χρόνο. Ο Sequential Executor χρησιμοποιεί την SQLite [WP16], η οποία μπορεί να έχει μόνο μία σύνδεση ανοιχτή την φορά. Για αυτό τον λόγο με την εγκατάσταση του Local Executor θα πρέπει να δημιουργηθεί και μία βάση PostgreSQL[WP39] ή MySQL [WP35], η οποία μπορεί να διαχειριστεί περισσότερες από μία συνδέσεις. Αφού δημιούργησα μία νέα βάση στη PostgreSQL για τον LocalExecutor

και τρέχοντας ξανά τον scheduler με τις νέες ρυθμίσεις χρησιμοποιώντας τον Local Executor παρατηρείται ότι τα δυο DAG runs τρέχουν παράλληλα (Εικόνα 61), όπως και οι διεργασίες που έχουν ορισθεί παράλληλα στο DAG. (Εικόνα 62)

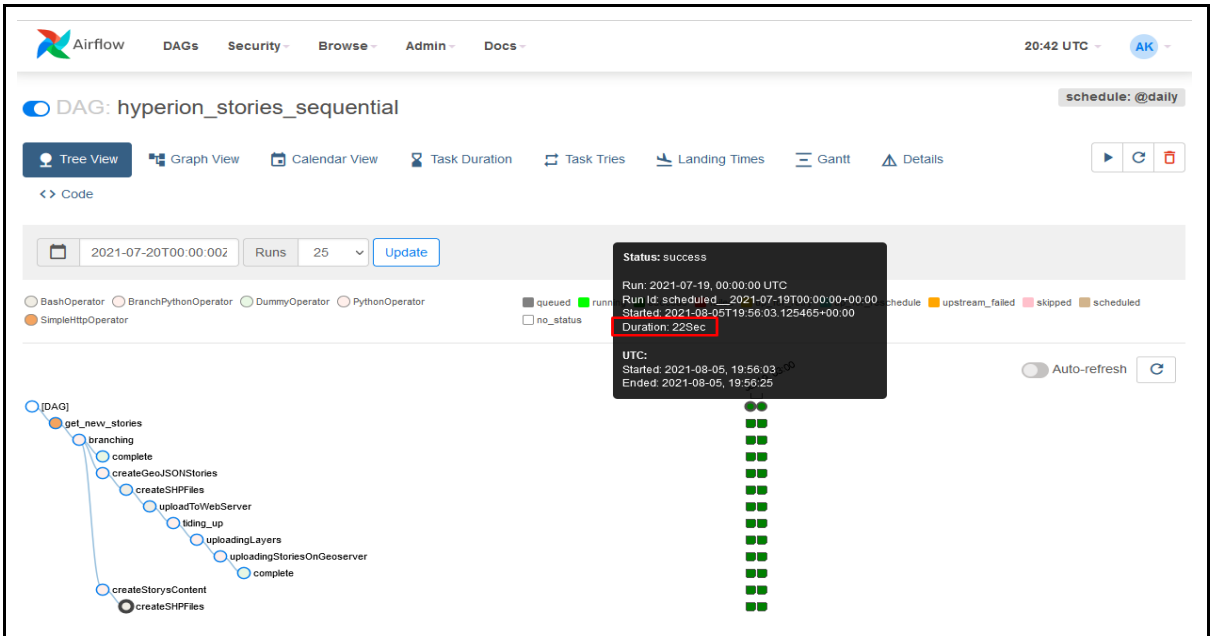


Εικόνα 61: Τα DAGs τρέχουν παράλληλα μεταξύ τους χρησιμοποιώντας τον LocalExecutor.



Εικόνα 62: Οι διεργασίες 2 και 3 τρέχουν παράλληλα.

Χρησιμοποιώντας τον Local executor, το DAG που εκτελέστηκε στις 19-7-2021 έτρεξε σχεδόν 50% πιο γρήγορα (Εικόνα 63) απ' ότι με Sequential executor (Εικόνα 58).



Εικόνα 63: Ο χρόνος εκτέλεσης του DAG την ημερομηνία 19-7-2021 με χρήση Local Executor.

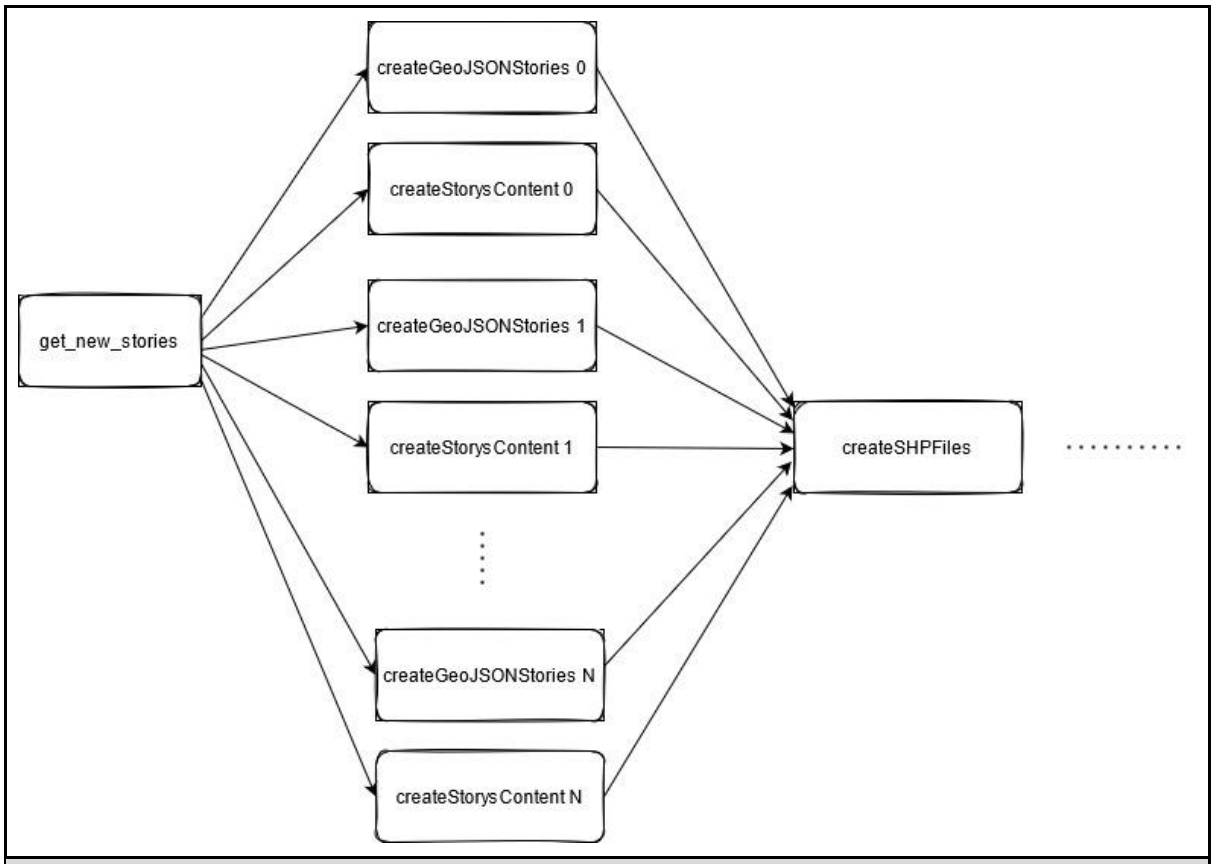
3.2.8.2 Δυναμική δημιουργία διεργασιών

Παρόλο που χρησιμοποιώντας τον LocalExecutor πλέον μπορούν να εκτελεστούν παράλληλα πολλές διεργασίες, η κάθε διεργασία εσωτερικά εκτελεί ένα script ακολουθιακά για κάθε ιστορία που έλαβε την συγκεκριμένη ημερομηνία. Στο παρακάτω κομμάτι κώδικα φαίνεται ότι στην μέθοδο createStoryContent υπάρχει μία for loop που εκτελεί ένα συγκεκριμένο κομμάτι κώδικα για κάθε ιστορία που λαμβάνεται την δεδομένη ημερομηνία.

```
def createStorysContent(**context):
    """Δημιουργεί το περιεχόμενο της ιστορίας """
    story_format = {}
    stories=json.loads(
context['task_instance'].xcom_pull(task_ids='get_new_stories'))

    for story in stories["data"]["result"]:

        storyType = story['type']
        storyID = story['_id']

        a_story = StoryStructure(story)

        if "citizen" in storyType:
            newFolder = str(storyID) + "_C"
        elif "business" in storyType:
            newFolder = str(storyID) + "_B"

        story_format[newFolder] = a_story.extractInfo()

        logging.info(story_format)

    return story_format
```

Συνεπώς, θα μπορούσε να αναρωτηθεί κανείς αν γίνεται στο Apache Airflow να κατασκευάσει κανείς δυναμικά διεργασίες ανάλογα με το αποτέλεσμα που λαμβάνει από μία προγενέστερη διεργασία. Στην περίπτωση του HCET η δημιουργία των δυναμικά παράλληλων διεργασιών θα γινόταν σε όποια διεργασία τρέχει κώδικας μέσα σε for loop για κάθε story, όπως για παράδειγμα στις διεργασίες 1 και 2 που δημιουργούν τα GeoJSON αρχεία και το περιεχόμενο

της ιστορίας αντίστοιχα. Έτσι, οι διεργασίες 1 και 2 δεν θα εκτελούσαν ακολουθιακά το περιεχόμενό τους, αλλά παράλληλα.



Εικόνα 64: Δυναμική δημιουργία διεργασιών.

Για να ελέγξω αν δουλεύει κάτι τέτοιο στο Airflow μελέτησα τον τρόπο με τον οποίο το Airflow διαβάζει το αρχείο DAG. Σύμφωνα με το επίσημο documentation του Airflow [WP19], ο scheduler διαβάζει τον φάκελο με τα DAGs ανά τακτά χρονικά διαστήματα (1 δευτερόλεπτο από προεπιλογή) και ανανεώνει τα DAGs αν υπάρχουν αλλαγές. Ένα από τα πλεονεκτήματα του Airflow είναι ότι μπορεί κανείς να δηλώσει τα DAGs γράφοντας κώδικα σε Python. Αυτό δίνει μεγάλη ευελιξία στον χρήστη, οπότε θα μπορούσε κανείς να δηλώσει το DAG με τον εξής τρόπο:

```
    # Προηγείται κώδικας
    for index in range(int(dynamicValue)):
        dynamicCreateGeoJSON = PythonOperator(
            task_id='createGeoJSONForStory' + str(index),
```

```
            dag=dag,
            provide_context=True,
            python_callable=createStoriesFromJSONtoGeoJSON,
            op_args=[index])
        dynamicCreateStoryContent = PythonOperator(
            task_id='createStoryContent' + str(index),
            dag=dag,
            provide_context=True,
            python_callable=createStoryContent,
            op_args=[index])

    # Ακολουθεί κώδικας
```

Όπου η μεταβλητή dynamicValue είναι το αποτέλεσμα της προηγούμενης διεργασίας, δηλαδή το πλήθος των ημερήσιων ιστοριών. Η for loop θα κατασκευάσει δυναμικά τόσες παράλληλες διεργασίες όσες υποδεικνύει η τιμή της dynamicValues. Ωστόσο, αν ο scheduler δεν προλάβει να ανανεώσει το DAG, οι δυναμικές διεργασίες δεν θα προλάβουν να δημιουργηθούν και αυτό καθιστά το όλο εγχείρημα των δυναμικών διεργασιών ασταθές.

Παρόλο που υπάρχει τρόπος να δημιουργήσει κανείς δυναμικές διεργασίες, αυτό καθιστά το DAG ασταθές και μία τέτοια πρακτική δεν συνιστάται. Το αποτέλεσμα μιας τέτοιας πρακτικής δεν είναι βέβαιο ότι θα πετύχει, διότι αυτό εξαρτάται από το αν ενημερωθεί το DAG από τον Scheduler στον σωστό χρόνο αλλά και από άλλους παράγοντες. Σύμφωνα με το επίσημο documentation του Apache Airflow [WP56], η χρήση των δυναμικών DAG προτείνεται όταν χρειάζεται να φορτωθούν δυναμικά configuration options ή για να αλλαχθούν τα option σε κάποιον Operator, ενώ συνίσταται η τοπολογία του DAG να παραμένει σχετικά σταθερή.

3.2.9 Συμπέρασμα

Η Μηχανική Δεδομένων είναι μία επιστήμη που επικεντρώνεται σε πρακτικές εφαρμογές και καλείται να συλλέξει δεδομένα, να τα επεξεργαστεί και να τα επικυρώσει, έτσι ώστε αυτά να χρησιμοποιηθούν σαν δεδομένα εισόδου σε ένα άλλο σύστημα. Στην εργασία αυτή εξετάστηκαν κάποιες μέθοδοι τις οποίες μπορεί κανείς να χρησιμοποιήσει για να επεξεργαστεί τα δεδομένα αυτά. Το σύνολο το διεργασιών που απαιτείται να εκτελεστεί για να ολοκληρώσει τις παραπάνω ενέργειες αποτελεί την δομή ενός σωληναγωγού. Για την διαχείριση, την

επίβλεψη και την ανάπτυξη τέτοιων σωληναγωγών δεδομένων έχουν αναπτυχθεί πλατφόρμες ενορχήστρωσης ροών εργασιών ή αλλιώς συστήματα διαχείρισης ροών διεργασιών. Όλες οι πλατφόρμες που εξετάστηκαν σε αυτήν την εργασία αποτελούν εξαιρετικά εργαλεία για την σύνταξη και την ανάπτυξη σωληναγωγών δεδομένων. Για κάθε μία από αυτές περιγράφηκε το αρχιτεκτονικό τους μοντέλο, τα βασικά δομικά τους στοιχεία αλλά και παραδείγματα σύνταξης και εκτέλεσης DAG, καθώς και τα γραφικά περιβάλλοντα που τις συνοδεύουν μέσω των οποίων ο χρήστης μπορεί να παρακολουθήσει το τρέξιμο των διεργασιών. Οι τρεις πλατφόρμες ανοιχτού κώδικα που παρουσιάστηκαν χρησιμοποιούν γλώσσες υψηλού επιπέδου, όπως Python και Java για να συντάξουν τις διεργασίες των διαγραμμάτων ροών εργασίας. Αυτό καθιστά την συγγραφή των διεργασιών δυναμική, αλλά και πιο ξεκάθαρη σε σχέση με άλλες πλατφόρμες που χρησιμοποιούν XML, JSON ή YAML.

Από τις τρεις ανοιχτού κώδικα πλατφόρμες που παρουσιάστηκαν, αυτή που επιλέχθηκε να χρησιμοποιηθεί για την υλοποίηση ενός σωληναγωγού δεδομένων που θα λαμβάνει τις ανεβασμένες ιστορίες των χρηστών από το PLUGGY και θα τις ανεβάζει μετά από ένα σύνολο επεξεργασιών στο Geonode, ήταν η Apache Airflow. Κατά την άποψή μου είναι η πιο ανεπτυγμένη και εξελιγμένη πλατφόρμα σε σύγκριση με τις άλλες δύο που παρουσιάστηκαν και προσφέρει εργαλεία, όπως ο Scheduler και μία μεγάλη λίστα από Operators, τα οποία ήταν απαραίτητα κατά την σχεδίαση και ανάπτυξη των διεργασιών του σωληναγωγού δεδομένων. Η εγκατάσταση του Apache Airflow σε τοπικό δίκτυο ήταν ανώδυνη, ενώ αρκετά εύκολη ήταν και η παραμετροποίηση του (Αλλαγή executor, επιλογή πλήθους Workers κλπ). Το αποτέλεσμα ήταν το θεμιτό, αφού τήρησε όλες τις προαπαιτούμενες προδιαγραφές. Επιπλέον, επιχειρήθηκε να βελτιωθεί η λογική με την οποία λειτουργεί ο σωληναγωγός δεδομένων, έτσι ώστε να επιταχυνθεί η συνολική διαδικασία, παρόλα αυτά όπως διαπιστώθηκε η υλοποίηση δυναμικών διεργασιών στο Airflow είναι ασταθής και πρέπει να αποφεύγεται.

Ωστόσο, η δουλειά μου στο Hyperion Community Engagement Tool δεν έχει ακόμα τελειώσει. Το Airflow προς το παρόν τρέχει σε τοπικό δίκτυο και δεν έχει μπει ακόμα σε παραγωγή. Οπότε ένα από τα επόμενα πράγματα που πρέπει να γίνουν, μεταξύ άλλων, είναι να γίνει το deployment του Apache Airflow. Ανάλογα με το αν αποφασιστεί η εγκατάσταση του Apache Airflow να γίνει σε ένα μηχάνημα ή αν γίνει κάποια εγκατάσταση σε Cluster, θα πρέπει να

χρησιμοποιηθεί ο αντίστοιχος Executor. Το Apache Airflow Documentation προτείνει τον LocalExecutor όταν η εγκατάσταση γίνεται σε ένα μηχάνημα, ενώ σε μια εγκατάσταση πολλαπλών κόμβων σε Cluster συνιστώνται οι Executors KubernetesExecutor ή CeleryExecutor. Τέλος, θα πρέπει να δοκιμαστεί η ορθή λειτουργία του σωληναγωγού δεδομένων με real-life δεδομένα.

Citations

[1] Densmore, James. *Data Pipelines Pocket Reference: Moving and Processing Data for Analytics*. 1st ed., O'Reilly Media, 2021.

[2] D7.7 Communities' Engagement ICT Tool (First Version)

[3] El-Sappagh, Shaker H. Ali, et al. "A Proposed Model for Data Warehouse ETL Processes." *Journal of King Saud University - Computer and Information Sciences*, vol. 23, no. 2, 2011, pp. 91–104. *Crossref*, doi:10.1016/j.jksuci.2011.05.005.

[4] Trujillo J., Luján-Mora S. (2003) A UML Based Approach for Modeling ETL Processes in Data Warehouses. In: Song IY., Liddle S.W., Ling TW., Scheuermann P. (eds) Conceptual Modeling - ER 2003. ER 2003. Lecture Notes in Computer Science, vol 2813. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-39648-2_25

[5] Katragadda, R., Tirumala, S.S., & Nandigam, D. (2015). ETL tools for Data Warehousing: An empirical study of Open Source Talend Studio versus Microsoft SSIS. IEEE (Ed.), CWISCE'2015 International Conference on Web Information System and Computing Education, The 2nd World Congress on Computer Applications and Information Systems (WCCAIS'2015)

[6] Inmon, William H. "What is a data warehouse." *Prism Tech Topic* 1.1 (1995): 1-5.

[7] Miloslavskaya, Natalia, and Alexander Tolstoy. "Big data, fast data and data lake concepts." *Procedia Computer Science* 88 (2016): 300-305.

[8] Marín-Ortega, Pablo Michel, et al. "ELTA: New approach in designing business intelligence solutions in era of big data." *Procedia technology* 16 (2014): 667-674.

[9] Arjona, Aitor, et al. "Triggerflow: Trigger-based orchestration of serverless workflows." *Future Generation Computer Systems* (2021).

Ιστότοποι

[WP1] What is a Data Warehouse? https://cloud.google.com/learn/what-is-a-data-warehouse

[WP2] Smallcombe, Mark. “ETL vs ELT: 5 Critical Differences.” *Xplenty*, March 16, 2021 , www.xplenty.com/blog/etl-vs-elt.

[WP3] “Azkaban.” *Azkaban*, azkaban.github.io. Accessed 22 May 2021.

[WP4] “The Official YAML Web Site.” YAML, yaml.org. Accessed 22 May 2021.

[WP5] Mvazas. “HYPERION’s Vision.” *Hyperion Project*, 9 Feb. 2021, www.hyperion-project.eu/hyperions-vision.

[WP6] Mvazas. “Home.” *Hyperion Project*, 27 Oct. 2020, www.hyperion-project.eu.

[WP7] “PLUGGY EU Project.” *PLUGGY Project*, www.pluggy-project.eu. Accessed 21 June 2021.

[WP8] “PLUGGY for Cultural Heritage.” *PLUGGY*, pluggy.eu. Accessed 21 June 2021.

[WP9] “What Is GeoNode — GeoNode 3.2.0 Documentation.” *Geonode Docs*, docs.geonode.org/en/master/about/index.html. Accessed 27 June 2021.

[WP10] “AC Atlas.” *AlpConv Atlas*, www.atlas.alpconv.org. Accessed 27 June 2021.

[WP11] “GeoServer.” *GeoServer*, geoserver.org. Accessed 27 June 2021.

[WP12] “The Home of Location Technology Innovation and Collaboration | OGC.” *Open Geospatial Consortium (OGC)*, www.ogc.org. Accessed 27 June 2021.

[WP13] “GDAL — GDAL Documentation.” *GDAL*, gdal.org/index.html. Accessed 24 July 2021.

[WP14] *Home*. (n.d.). Apache Airflow. Retrieved July 26, 2021, from https://airflow.apache.org

[WP15] *Getting Started — Luigi 2.8.13 documentation*. (n.d.). Luigi. Retrieved July 27, 2021, from https://luigi.readthedocs.io/en/stable/

[WP16] "SQLite Home Page." *SQLite*, www.sqlite.org/index.html. Accessed 5 Aug. 2021.

[WP17] "Sequential Executor — Airflow Documentation." *Apache Airflow*, airflow.apache.org/docs/apache-airflow/stable/executor/sequential.html. Accessed 5 Aug. 2021.

[WP19] "Best Practices — Airflow Documentation." *Airflow Documentation*, airflow.apache.org/docs/apache-airflow/stable/best-practices.html. Accessed 6 Aug. 2021.

[WP20] "Temporal.Io: Build Invincible Apps." *Temporal*, temporal.io. Accessed 8 Aug. 2021.

[WP21] "ICCS – Institute of Communications and Computer Systems." *Institute of Communication and Computer Systems*, www.iccs.gr/en/?noredirect=en_US. Accessed 9 Aug. 2021.

[WP22] "Luigi.Interface Module — Luigi 2.8.13 Documentation." *Luigi Interface Module*, luigi.readthedocs.io/en/stable/api/luigi.interface.html?highlight=workers#luigi.interface.core.workers. Accessed 13 Aug. 2021.

[WP23] "Apache Kafka." *Apache Kafka*, kafka.apache.org.

[WP24] "BigQuery: Cloud Data Warehouse." *Google Cloud*, cloud.google.com/bigquery. Accessed 15 Aug. 2021.

[WP25] "Amazon Redshift - Cloud Data Warehouse - Amazon Web Services." *Amazon Web Services, Inc.*, aws.amazon.com/redshift. Accessed 15 Aug. 2021.

[WP26] "Snowflake on Microsoft Azure | Snowflake Cloud Partners." *Snowflake*, www.snowflake.com/technology-partners/microsoft.

[WP27] Thallam, Rajesh. "An Overview of BigQuery's Architecture and How to Quickly Get Started." *Google Cloud Blog*, 2 Sept. 2020, cloud.google.com/blog/products/data-analytics/new-blog-series-bigquery-explained-overview.

[WP28] "Columnar Storage - Amazon Redshift." *Columnar Storage - Amazon Redshift*, docs.aws.amazon.com/redshift/latest/dg/c_columnar_storage_disk_mem_mgmnt.html. Accessed 15 Aug. 2021.

[WP29] "Key Concepts & Architecture — Snowflake Documentation." *Key Concepts & Architecture — Snowflake Documentation*, docs.snowflake.com/en/user-guide/intro-key-concepts.html. Accessed 15 Aug. 2021.

[WP30] "Architecture Overview — Airflow Documentation." *Architecture Overview — Airflow Documentation*, airflow.apache.org/docs/apache-airflow/stable/concepts/overview.html. Accessed 18 Aug. 2021.

[WP31] "Concepts — Airflow Documentation." *Concepts — Airflow Documentation*, airflow.apache.org/docs/apache-airflow/stable/concepts/index.html. Accessed 18 Aug. 2021.

[WP32] "Executor — Airflow Documentation." *Airflow Documentation*, airflow.apache.org/docs/apache-airflow/stable/executor/index.html. Accessed 18 Aug. 2021.

[WP33] "Pods." *Kubernetes*, kubernetes.io/docs/concepts/workloads/pods. Accessed 18 Aug. 2021.

[WP34] "Introduction to Amazon S3 (4:31)." *Amazon Web Services, Inc.*, aws.amazon.com/s3. Accessed 18 Aug. 2021.

[WP35] "MySQL." *MySQL*, www.mysql.com. Accessed 18 Aug. 2021.

[WP36] "Using the Central Scheduler." *Luigi 2.8.13 Documentation*, luigi.readthedocs.io/en/stable/central_scheduler.html. Accessed 18 Aug. 2021.

[WP37] "Execution Model." *Luigi 2.8.13 Documentation*, luigi.readthedocs.io/en/stable/execution_model.html. Accessed 18 Aug. 2021.

[WP38] "Apache Cassandra | Apache Cassandra Documentation." *Apache Cassandra*, cassandra.apache.org/_/index.html. Accessed 19 Aug. 2021.

[WP39] "PostgreSQL" https://www.postgresql.org/

[WP40] Guindon, Christopher. “Eclipse Desktop & Web IDEs | The Eclipse Foundation.” *The Eclipse Foundation*, www.eclipse.org/ide. Accessed 19 Aug. 2021.

[WP41] “What Is a Data Lake?” *Amazon Web Services, Inc.*, aws.amazon.com/big-data/datalakes-and-analytics/what-is-a-data-lake. Accessed 21 Aug. 2021.

[WP42] https://www.coursera.org/articles/what-does-a-data-engineer-do-and-how-do-i-become-one

[WP43] “YAML Ain’t Markup Language (YAML™) Version 1.2.” *YAML*, yaml.org/spec/1.2/spec.html. Accessed 23 Aug. 2021.

[WP44] “Introduction - Conductor.” *Netflix’s Conductor*, netflix.github.io/conductor. Accessed 23 Aug. 2021.

[WP45] “Welcome To.” *Python.Org*, 18 Aug. 2021, www.python.org.

[WP46] “Extensible Markup Language (XML).” *www.w3.org*, www.w3.org/XML Accessed 23 Aug. 2021.

[WP47] “JSON.” *JSON*, www.json.org/json-en.html. Accessed 23 Aug. 2021.

[WP48] https://www.java.com/en/

[WP49] “Partners.” *Hyperion Project*, 22 Apr. 2020, www.hyperion-project.eu/partners.

[WP50] “Test Sites.” *Hyperion Project*, 22 Apr. 2020, www.hyperion-project.eu/test-sites.

[WP51] Singer. “Singer | Open Source ETL.” *Singer*, www.singer.io. Accessed 15 May 2021.

[WP52] “Temporal Documentation | Temporal Documentation.” *Temporal Docs*, docs.temporal.io. Accessed 26 Sept. 2021.

[WP53] “Empowering App Development for Developers.” *Docker*, www.docker.com. Accessed 28 Sept. 2021.

[WP54] Mozilla. “Internet for People, Not Profit.” *Mozilla*, www.mozilla.org/en-US. Accessed 28 Sept. 2021.

[WP55] “Diagrams.Net - Free Flowchart Maker and Diagrams Online.” *Diagrams.Net*, app.diagrams.net. Accessed 3 Oct. 2021.

[WP56] “DAGs — Airflow Documentation.” *DAGs - Airflow Documentation*, airflow.apache.org/docs/apache-airflow/stable/concepts/dags.html#dynamic-dags. Accessed 3 Oct. 2021.

[WP57] *https://App.Diagrams.Net/*, app.diagrams.net. Accessed 14 Oct. 2021.