ΧΑΡΟΚΟΠΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ

ΣΧΟΛΗ ΨΗΦΙΑΚΗΣ ΤΕΧΝΟΛΟΓΙΑΣ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΜΑΤΙΚΗΣ

Υλοποίηση CI/CD pipeline για τη βιβλιοθήκη Python JGraphT

Πτυχιακή εργασία

Τσάμης Φώτιος

Αθήνα, 2021

ΧΑΡΟΚΟΠΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ

ΣΧΟΛΗ ΨΗΦΙΑΚΗΣ ΤΕΧΝΟΛΟΓΙΑΣ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΜΑΤΙΚΗΣ

Τριμελής Εξεταστική Επιτροπή

Μιχαήλ Δημήτριος

Επίκουρος Καθηγητής,

Τμήμα Πληροφορικής και Τηλεματικής,

Χαροκόπειο Πανεπιστήμιο

Νικολαΐδη Μαρία

Καθηγήτρια,

Τμήμα Πληροφορικής και Τηλεματικής,

Χαροκόπειο Πανεπιστήμιο

Τσερπές Κωνσταντίνος

Επίκουρος Καθηγητής,

Τμήμα Πληροφορικής και Τηλεματικής,

Χαροκόπειο Πανεπιστήμιο

Ο Φώτης Τσάμης δηλώνω υπεύθυνα ότι:

1) Είμαι ο κάτοχος των πνευματικών δικαιωμάτων της πρωτότυπης αυτής εργασίας και από όσο γνωρίζω η εργασία μου δε συκοφαντεί πρόσωπα, ούτε προσβάλει τα πνευματικά δικαιώματα τρίτων.

2) Αποδέχομαι ότι η ΒΚΠ μπορεί, χωρίς να αλλάξει το περιεχόμενο της εργασίας μου, να τη διαθέσει σε ηλεκτρονική μορφή μέσα από τη ψηφιακή Βιβλιοθήκη της, να την αντιγράψει σε οποιοδήποτε μέσο ή/και σε οποιοδήποτε μορφότυπο καθώς και να κρατά περισσότερα από ένα αντίγραφα για λόγους συντήρησης και ασφάλειας.

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

Περίληψη

Οι μεθοδολογίες Continuous Integration, Delivery & Deployment αποτελούν πλέον βασικό ζητούμενο σε πολλά έργα προγραμματισμού που απαιτείται γρήγορη και αξιόπιστη ανάπτυξη του κώδικα και εξ' ίσου γρήγορη ροή νέων εκδόσεων. Σε αυτή την εργασία, πέραν της θεωρητικής περιγραφής των εννοιών αυτών, των απαιτήσεών τους αλλά και των πλεονεκτημάτων τους, υλοποιείται ένα πλήρες CI/CD pipeline για το build, το test και το release της βιβλιοθήκης γράφων ανοιχτού κώδικα python-jgrapht. Βασικός στόχος της εργασίας αυτής είναι η διευκόλυνση των προγραμματιστών της βιβλιοθήκης ως προς αυτές τις διαδικασίες και η αυτοματοποιημένη δημιουργία και δημοσίευση πακέτων Python wheel συμβατά με όλες τις τελευταίες εκδόσεις της CPython στα τρία πιο δημοφιλή λειτουργικά συστήματα (Linux, MacOS, Windows). Ως αποτέλεσμα, μετά από κάθε αλλαγή που πραγματοποιείται στον κώδικα, δημιουργούνται αυτόματα 9 διακριτά πακέτα της βιβλιοθήκης εντός μερικών μόνο λεπτών, ενώ δίνεται η δυνατότητα αυτόματης δημοσίευσής τους στο Python Package Index.

Λέξεις κλειδιά: CI/CD, Docker, Python, JGraphT

Abstract

Continuous Integration, Delivery & Deployment methodologies are now a key aspect of many programming projects that require fast and reliable code development and an equally fast flow of new releases. In this work, in addition to the theoretical description of these concepts, their requirements and their advantages, a complete CI / CD pipeline is implemented for the build, test and release of the python-jgrapht open source graph library. The main goal of this work is to facilitate the library developers in these processes and to automate the creation and publication of Python wheel packages compatible with all the latest versions of CPython on the three most popular operating systems (Linux, MacOS, Windows). As a result, after each change in the code, 9 separate packages of the library are automatically created in just a few minutes, with the possibility of their automated publication in the Python Package Index.

Keywords: CI/CD, Docker, Python, JGraphT

Κατάλογος πινάκων

Κατάλογος σχημάτων

Συντομογραφίες

CI	Continuous Integration
CD	Continuous Delivery / Continuous Deployment
VCS	Version Control System
PyPI	Python Package Index
PEP	Python Enhancement Proposal
CLI	Command Line Interface
ELF	Executable and Linkable Format
MSVC	Microsoft Visual C++ compiler
XP	Extreme Programming

1. Εισαγωγή

Παραδοσιακά, η ανάπτυξη λογισμικού γινόταν συνήθως με τους προγραμματιστές να δουλεύουν ανεξάρτητα ο ένας από τον άλλο και με μεγάλες περιόδους να μεσολαβούν μεταξύ της κάθε έκδοσης λογισμικού. Κάθε σύνολο αλλαγών από τους προγραμματιστές έπρεπε να ενοποιηθεί με τον κυρίως κώδικα αφού είχε ολοκληρωθεί, στην πορεία ο κώδικας να περάσει στην ομάδα διασφάλισης ποιότητας και αφού καταφέρουν να πραγματοποιήσουν επιτυχώς το build του κώδικα, να ελέγξουν με end-to-end tests για τυχόν bugs σε αυτόν, στην οποία - κοινή - περίπτωση ο κώδικας θα έπρεπε να επανεξεταστεί από τους προγραμματιστές για να επιλυθεί το πρόβλημα. Ως αποτέλεσμα της μακράς αυτής διαδικασίας, η δημιουργία νέων εκδόσεων καθυστερούσε ακόμη περισσότερο, ενώ η διασφάλιση της ποιότητας του κώδικα ήταν ένα πολύ δύσκολο εγχείρημα για την αρμόδια ομάδα.

Με την αυξημένη πολυπλοκότητα στην ανάπτυξη λογισμικού προκύπτει η ανάγκη αυτοματοποίησης επαναλαμβανόμενων διαδικασιών όπως το build του πηγαίου κώδικα, η δοκιμή (testing) του και η διανομή του στους τελικούς αποδέκτες. Για να καλυφθούν αποτελεσματικά αυτές οι ανάγκες έχει δημιουργηθεί ένα σύνολο πρακτικών που ονομάζεται Continuous Integration, Continuous Delivery και Continuous Deployment (CI/CD). Στόχος αυτών των πρακτικών είναι να υπάρχει μια όσο το δυνατόν πιο συνεχής ροή νέου κώδικα στο κυρίως αποθετήριο από τους προγραμματιστές, ο οποίος αυτόματα γίνεται build, ελέγχεται ως προς κάποια βασικά κριτήρια ποιότητας και λειτουργικότητας και αποθηκεύεται σε μια μορφή που μπορεί εύκολα να διανεμηθεί σε άλλους και να αποτελέσει σημείο αναφοράς. Τέλος, στην ιδανική περίπτωση που όλες αυτές οι διαδικασίες έχουν εξελιχθεί αρκετά στα πλαίσια ενός project, το Continuous Deployment μπορεί να διασφαλίσει ότι όλες οι αλλαγές που πληρούν τις προϋποθέσεις του αυτοματοποιημένου ελέγχου ποιότητας μπορούν να γίνουν deploy σε ένα παραγωγικό σύστημα χωρίς καμία ανθρώπινη παρέμβαση πέραν της καταχώρισης του κώδικα.

Σε αυτή την εργασία εξετάζεται μια προσέγγιση υλοποίησης ενός CI/CD pipeline για το build, το test και την διανομή πακέτων της βιβλιοθήκης python-jgrapht [1] συμβατά με τα τρία πιο δημοφιλή λειτουργικά συστήματα, δηλαδή Linux, MacOS και Windows και για όλες τις τελευταίες εκδόσεις της γλώσσας προγραμματισμού Python: 3.6, 3.7, 3.8.

2. Θεωρητικό υπόβαθρο

2.1. CI/CD

Τα CI και CD είναι δύο ακρωνύμια που εμφανίζονται συχνά τα τελευταία χρόνια στην ανάπτυξη λογισμικού. Το CI είναι τα αρχικά για το Continuous Integration, ενώ το CD είναι τα αρχικά είτε για το Continuous Delivery ή για το Continuous Deployment - δύο έννοιες με διαφορετική σημασία.

Ο όρος Continuous Integration εμφανίστηκε για πρώτη φορά το 1991 από τον Grady Booch στο βιβλίο Object-Oriented Analysis and Design: with Applications [2] χωρίς όμως να περιγράφει σε αυτό την ιδέα της εφαρμογής του CI στη διαδικασία του build ολόκληρης της εφαρμογής. Με την πρόοδο των μεθοδολογιών και των τεχνολογιών στην ανάπτυξη λογισμικού απέκτησε ιδιαίτερη σημασία η αυτοματοποίηση ολόκληρης της διαδικασίας του build μιας εφαρμογής με κάθε νέα αλλαγή που πραγματοποιούνταν στον κώδικα [3]. Η μεθοδολογία προγραμματισμού Extreme Programming (XP) υιοθέτησε την πρακτική του CI και την ενέταξε ως αναπόσπαστο μέρος της σε συνδυασμό με την εκτέλεση των unit tests μετά από κάθε αλλαγή στον κώδικα [3][4]. Το CI χρησιμοποιείται επίσης στο μοντέλο ανάπτυξης λογισμικού agile [5] καθώς και στο μοντέλο DevOps [6] - μοντέλα που αμφότερα εστιάζουν στην επιτάχυνση της διαδικασίας της ανάπτυξης λογισμικού με ταυτόχρονη διασφάλιση υψηλής ποιότητας.

Ενώ η πρακτική του Continuous Integration είναι ικανή να καλύψει τις ανάγκες για ταχεία ανάπτυξη αξιόπιστου λογισμικού που απαιτούν οι μεθοδολογίες agile και XP, σταματάει στο merge του κώδικα στο κυρίως αποθετήριο, χωρίς να χειρίζεται καθόλου τον τομέα της έκδοσης (release) και της εγκατάστασης (deployment) του λογισμικού. Δεδομένου ότι η αυτοματοποίηση του build και του testing αποδείχθηκε εξαιρετικά χρήσιμη για τις ομάδες ανάπτυξης λογισμικού, δεν άργησε να δημιουργηθεί και η ανάγκη για την αυτοματοποίηση της δημιουργίας νέας έκδοσης λογισμικού. Για αυτό το λόγο εισήχθη η έννοια του Continuous Delivery, πρακτική η οποία αναλαμβάνει ακριβώς το κομμάτι της συνεχούς δημιουργίας νέων εκδόσεων ενός λογισμικού και της διάθεσής τους με οργανωμένο τρόπο σε κάποιο σύστημα για άμεσο deployment, μετά από ανθρώπινη παρέμβαση.

Με την υπόθεση ότι σε μία ομάδα ανάπτυξης λογισμικού υπάρχει ανεπτυγμένη η κουλτούρα του Test Driven Development (TDD) - η οποία επιτάσσει την δημιουργία καλογραμμένων tests πριν τη συγγραφή ενός νέου feature στον κυρίως κώδικα και τη δοκιμή του με αυτό - και μια αξιόπιστη και πλήρης σουίτα από tests που εκτελούνται στη φάση του CI, σε ορισμένες περιπτώσεις είχε νόημα η περαιτέρω επέκταση της αυτοματοποίησης των διαδικασιών ώστε οι νέες εκδόσεις του λογισμικού να μην γίνονται deploy στα παραγωγικά συστήματα με ανθρώπινη παρέμβαση, αλλά αυτόματα. Με αυτό τον τρόπο, που ονομάζεται Continuous Deployment, ολοκληρώνεται η φιλοσοφία του Continuous Integration, Delivery, Deployment, ή αλλιώς: CI/CD.

Μία αυτοματοποιημένη διαδικασία, στα πλαίσια της πρακτικής του CI/CD, ονομάζεται job. Ένα σύνολο από jobs ονομάζεται stage και αποτελεί μια λογική και σειριακή ομαδοποίηση των jobs. Τα τυπικά stages είναι τα: build για το χτίσιμο του κώδικα, test για τη δοκιμή του κώδικα και deploy για την εγκατάσταση του κώδικα σε κάποιο σύστημα, ενώ μπορούν να οριστούν περισσότερα ή διαφορετικά stages. Στο σύνολό τους, τα stages αποτελούν αυτό που ονομάζεται CI/CD pipeline.

Στην ενότητα αυτή αναλύονται περαιτέρω αυτές οι τρεις έννοιες, παρατίθενται ορισμένα από τα πλεονεκτήματα που μπορούν να προσφέρουν, καθώς και οι απαιτήσεις που έχει η κάθε μία από αυτές για να τεθεί σωστά σε εφαρμογή.

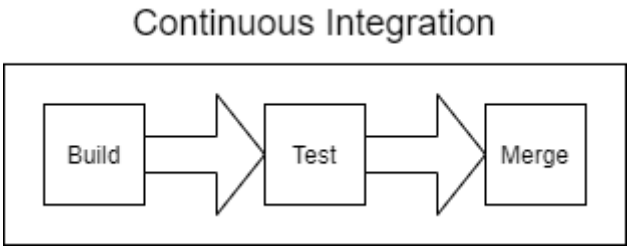
2.2. Continuous Integration

Continuous Integration (CI) ονομάζεται η πρακτική της αυτοματοποίησης της ενσωμάτωσης αλλαγών στον κώδικα ενός λογισμικού από διαφορετικούς προγραμματιστές σε ένα κυρίως αποθετήριο σε συνεχή βάση. Με κάθε αλλαγή στο αποθετήριο εκτελούνται προκαθορισμένες διεργασίες για το χτίσιμο (build) του κώδικα και τη δοκιμή (test) του με κάποια σενάρια δοκιμών. Με τον τρόπο αυτό, διασφαλίζεται ένα επίπεδο ποιότητας του κώδικα και διευκολύνεται η ταυτόχρονη εργασία πολλών προγραμματιστών, μειώνοντας την πιθανότητα κάποια αλλαγή ενός προγραμματιστή να προκαλέσει πρόβλημα σε κάποιον άλλον.

Ένα Version Control System (VCS) αποτελεί απαραίτητη προϋπόθεση για την εφαρμογή της πρακτικής του CI, ενώ απαιτείται και ένας build server που είναι επιφορτισμένος με το να εκτελεί όλες τις απαιτούμενες ενέργειες. Εκτός από το build και τη δοκιμή του κώδικα, επιπλέον αυτοματοποιήσεις μπορούν να εισαχθούν ως μέρος ενός CI pipeline όπως ενδεικτικά έλεγχοι για: την ποιότητα του κώδικα, ορθής σύνταξης, πληρότητας τεκμηρίωσης κ.α.

Μερικά από τα πλεονεκτήματα της εφαρμογής του CI είναι:

- λιγότερα bugs καταλήγουν στο κυρίως αποθετήριο καθώς κάθε commit δοκιμάζεται ξεχωριστά από το CI pipeline
- η διαδικασία του build γίνεται αυτόματα χωρίς να χρειάζεται να δαπανηθεί χρόνος
- οι ομάδες που εξασφαλίζουν την ποιότητα του κώδικα (QA) δεν αφιερώνουν χρόνο σε βασικά, επαναλαμβανόμενα tests, αντ' αυτού επικεντρώνονται σε πιο πολύπλοκα tests που ενδεχομένως να μην μπορούν ή να μην είναι εύκολο να αυτοματοποιηθούν.



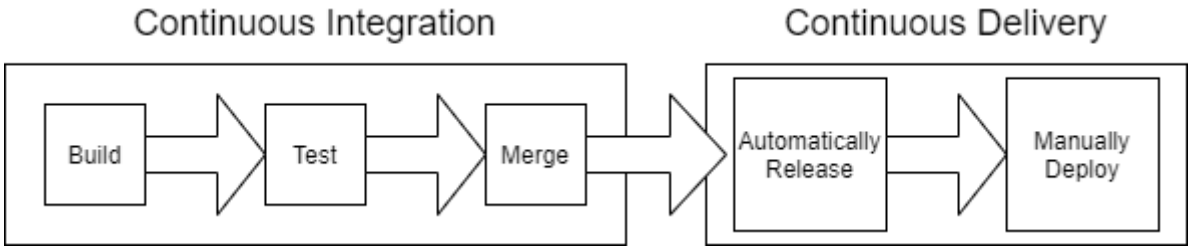
Σχ. 1: Κύριες φάσεις ενός CI pipeline

2.3. Continuous Delivery

Η έννοια Continuous Delivery ξεκινάει αμέσως μετά τη διαδικασία του Continuous Integration και περιλαμβάνει την αυτοματοποιημένη καταχώριση των builds που πραγματοποιούνται στο Continuous Integration σε κάποιο αποθετήριο, καθώς και τη δυνατότητα για άμεση εκτέλεση ενός deployment σε κάποιο περιβάλλον, είτε αυτό είναι δοκιμαστικό είτε παραγωγικό.

Για να υλοποιηθεί το Continuous Delivery σωστά προϋποθέτει την ύπαρξη ενός λειτουργικού CI pipeline με ικανοποιητική κάλυψη του κώδικα ως προς τα αυτοματοποιημένα tests για να εξασφαλιστεί ότι οι αυτοματοποιημένες εκδόσεις του λογισμικού θα είναι και λειτουργικές.

Η χρήση ενός Continuous Delivery pipeline έχει ως συνέπεια εκτός από αυτοματοποίηση των νέων εκδόσεων ενός λογισμικού, την εύκολη πρόσβαση σε όλες τις παλαιότερες εκδόσεις του, ενώ προσφέρει τη δυνατότητα για δημιουργία νέων εκδόσεων σε πολύ σύντομο χρονικό διάστημα μεταξύ τους.



Σχ. 2: Κύριες φάσεις ενός CI/CD pipeline

2.4. Continuous Deployment

Το Continuous Deployment προχωράει την αυτοματοποίηση ένα βήμα μπροστά από το Continuous Delivery και δίνει τη δυνατότητα για την πραγματοποίηση deployments ενός λογισμικού χωρίς καμία ανθρώπινη παρέμβαση, με μόνη προϋπόθεση να επιτυγχάνουν τα tests στη φάση του Continuous Integration. Με αυτόν τον τρόπο, οι προγραμματιστές μπορούν να βλέπουν τις αλλαγές τους να περνάνε στο παραγωγικό περιβάλλον πολύ πιο γρήγορα σε σχέση με τις παραδοσιακές μεθόδους ανάπτυξης λογισμικού και οι χρήστες μπορούν να επωφεληθούν από διορθώσεις σε bugs ή νέα χαρακτηριστικά γρηγορότερα.

Για να μπορέσει να λειτουργήσει στην πράξη η μεθοδολογία του Continuous Deployment απαιτείται η βέλτιστη δυνατή κάλυψη από καλογραμμένα tests αφού λόγω της πρακτικά ανύπαρκτης ανθρώπινης παρέμβασης (πέραν του προγραμματιστή) κάθε αλλαγή δύναται να ενσωματωθεί στο παραγωγικό σύστημα. Έτσι, αν ένα test αποτύχει στο CI pipeline, η νέα έκδοση δεν θα προχωρήσει και επομένως το bug που έκανε το test να αποτύχει δεν θα εισαχθεί στο παραγωγικό σύστημα.

2.5. Εργαλεία και υπηρεσίες για CI/CD

Ένα από τα πρώτα δημοφιλή εργαλεία για CI/CD ήταν το CruiseControl που εκδόθηκε για πρώτη φορά το Μάρτιο του 2001 από την αμερικανική εταιρεία ThoughtWorks και ήταν γραμμένο σε Java. Το CruiseControl λειτουργούσε με απλό τρόπο: παρακολουθούσε για αλλαγές στον κώδικα μιας εφαρμογής, και δημιουργούσε ένα νέο build το οποίο πέρναγε από ένα σύνολο από tests για να επαληθευτεί πως λειτουργεί σωστά.

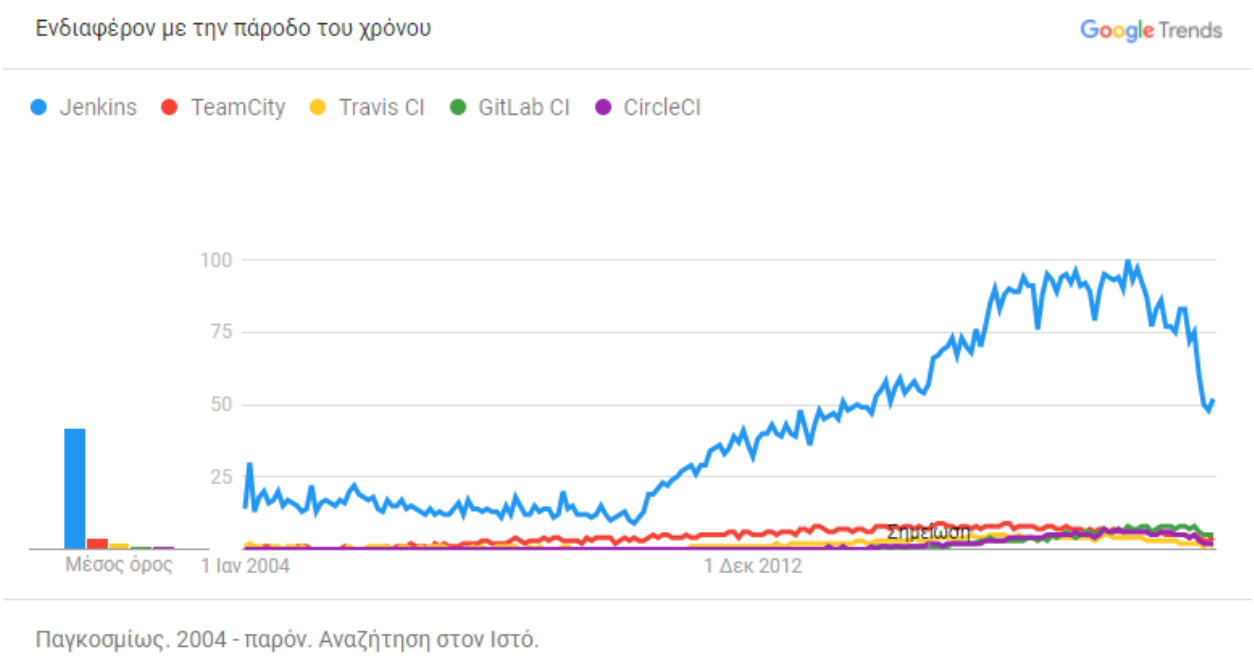
Το 2005 εμφανίστηκε ένα ακόμη εργαλείο για CI, γραμμένο σε Java: το Hudson. Το Hudson γράφτηκε από τον Kohsuke Kawaguchi, που εργαζόταν για την Sun Microsystems και από το 2008 και έπειτα άρχισε σταδιακά να παίρνει μεγαλύτερο μερίδιο χρηστών από το CruiseControl. Όταν το 2010 η Oracle αγόρασε την Sun Microsystems, η κοινότητα του Hudson αποχώρησε από την ανάπτυξη και υποστήριξή του και μετέφερε τις προσπάθειές της σε ένα νέο έργο υπό το όνομα Jenkins. Το Jenkins από τότε παραμένει και το πιο δημοφιλές σύστημα για CI. [7]

Στο διάγραμμα που ακολουθεί φαίνεται το ενδιαφέρον του κόσμου για το CruiseControl και το Jenkins από το 2004 μέχρι και το 2020 όπως αποτυπώνεται από το Google Trends. Είναι φανερό πως μόλις το Jenkins έκανε την εμφάνισή του, στην πράξη αντικατέστησε το CruiseControl.

Σχ. 3: Διάγραμμα ενδιαφέροντος για το Jenkins και το CruiseControl

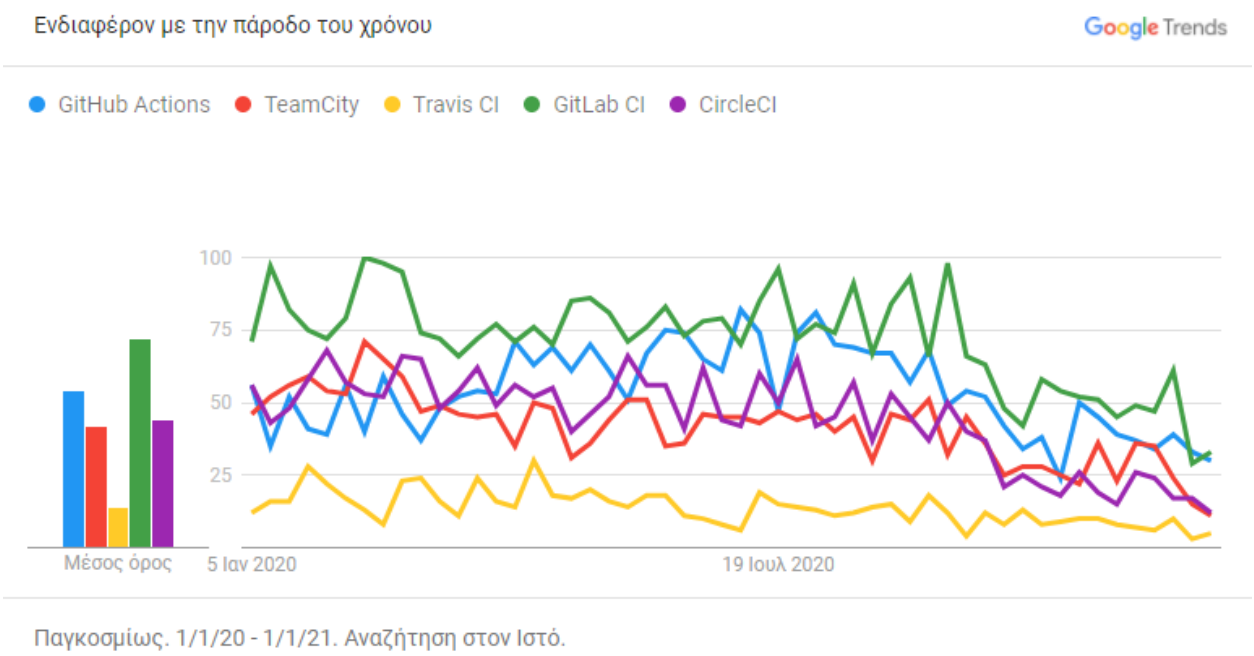
Όπως ήταν φυσικό, εκτός από το Jenkins και το CruiseControl, τα τελευταία χρόνια έχουν εμφανιστεί δεκάδες ακόμη υλοποιήσεις συστημάτων CI, με πολλά από αυτά να προσανατολίζονται στο μοντέλο Software as a Service. Οι πιο δημοφιλείς υλοποιήσεις από αυτές τις νέες πλατφόρμες είναι το TeamCity, το Travis CI, GitLab CI και το CircleCI. Αξίζει όμως να σημειωθεί πως το Jenkins παραμένει η πιο δημοφιλής επιλογή για πλατφόρμα CI.

Στο παρακάτω διάγραμμα συγκρίνεται η δημοφιλία των πιο πρόσφατων υπηρεσιών και λογισμικών για CI με αυτή του Jenkins όπως διαμορφώνεται με την πάροδο του χρόνου.



Σχ. 4: Διάγραμμα ενδιαφέροντος για το Jenkins και πιο πρόσφατα συστήματα CI

Τον τελευταίο χρόνο έκανε την εμφάνισή του και το GitHub Actions. Μια προσέγγιση που προσφέρει λειτουργικότητα CI εντός της πλατφόρμας του GitHub. Στο παρακάτω διάγραμμα συγκρίνεται ως προς τη δημοφιλία του με τις υπόλοιπες δημοφιλείς πλατφόρμες για CI.



Σχ. 5: Διάγραμμα ενδιαφέροντος για το GitHub Actions και άλλα δημοφιλή συστήματα CI

3. Περιγραφή τεχνολογιών

3.1. JGraphT

Το JGraphT είναι μια βιβλιοθήκη γραμμένη σε Java που υλοποιεί δομές και αλγορίθμους γράφων με στόχο την αυξημένη αποδοτικότητα και τη σταθερότητα. [1]

3.2. Travis CI

Το Travis CI είναι μια υπηρεσία για εκτέλεση CI/CD pipelines και build λογισμικού σε διάφορα λειτουργικά συστήματα και αρχιτεκτονικές. Παρέχει τη δυνατότητα να ενημερώνεται αυτόματα από το GitHub για αλλαγές σε ένα git repository και να εκτελεί ένα pipeline που ορίζεται μέσω της γλώσσας YAML σε ένα αρχείο με όνομα .travis.yml που τοποθετείται στον αρχικό κατάλογο του project. Σε αυτό το αρχείο ορίζονται τα επιθυμητά βήματα για την εκτέλεση του pipeline, η γλώσσα στην οποία είναι γραμμένο το project, τα επιθυμητά λειτουργικά συστήματα και οι αρχιτεκτονικές για την εκτέλεση του build καθώς και τα dependencies του.

Το Travis CI παρέχει ορισμένες από τις υπηρεσίες του δωρεάν σε projects ανοιχτού κώδικα.

3.3. GraalVM - Native Image

Το GraalVM είναι ένα JVM και JDK. Υποστηρίζει ahead-of-time compilation Java εφαρμογών μέσω του Native Image, μια διαδικασία μέσω της οποίας μπορούν να παραχθούν native binaries από bytecode Java (αρχεία Java class). Το εκτελέσιμο που παράγεται ως αποτέλεσμα του Native Image περιλαμβάνει όλες τις κλάσεις του αρχικού κώδικα Java, τις βιβλιοθήκες που χρησιμοποιούνται από εκείνον καθώς και όλα τα απαραίτητα στοιχεία που παρέχει ένα JVM και απαιτούνται για την εκτέλεσή του κώδικα Java, όπως ενδεικτικά τους μηχανισμούς διαχείρισης μνήμης, τους μηχανισμούς thread scheduling κ.α. Οι μηχανισμοί αυτοί ενσωματώνονται στο εκτελέσιμο από ένα υποκατάστατο του JVM που ονομάζεται Substrate VM και αναπτύσσεται ως μέρος του Native Image project. Με αυτόν τον τρόπο επιτυγχάνεται η εξάλειψη της ανάγκης για JVM ως μέσο για την εκτέλεση του αρχικού κώδικα, με το native εκτελέσιμο να χρειάζεται λιγότερο χρόνο για την εκκίνηση καθώς και λιγότερη μνήμη RAM. [8]

3.4. Docker

Το docker είναι μια πλατφόρμα για virtualization σε επίπεδο λειτουργικού συστήματος με σκοπό τη δυνατότητα διανομής λογισμικού σε ανεξάρτητα πακέτα που ονομάζονται containers. Τα containers εκτελούνται εντός του πυρήνα του λειτουργικού συστήματος, καθιστώντας τα πιο ελαφριά σε απαιτήσεις σε σχέση με μια εικονική μηχανή. Το Docker λειτουργεί σε Linux, MacOS και Windows συστήματα.

Στο Linux, το Docker βασίζεται σε χαρακτηριστικά του πυρήνα όπως τα cgroups και τα kernel namespaces για να επιτύχει την ανεξάρτητη και απομονωμένη εκτέλεση πολλαπλών containers εντός του ίδιου συστήματος. Τα cgroups χρησιμοποιούνται για τον περιορισμό των διαθέσιμων πόρων συστήματος όπως ο επεξεργαστής και η μνήμη RAM σε κάθε container, ενώ τα

namespaces δίνουν στα containers μια περιορισμένη όψη των ενεργών διεργασιών, των χρηστών, του δικτύου κ.α. ώστε να επιτευχθεί μια μορφή isolation.

Στο MacOS το Docker εκτελείται χρησιμοποιώντας ένα Linux virtual machine, ενώ υποστηρίζει συγκεκριμένες εκδόσεις Windows.

3.5. Wheels

Τα wheels είναι ο ενδεδειγμένος τρόπος της γλώσσας Python για διανομή βιβλιοθηκών έτοιμων προς εγκατάσταση ως ένα αρχείο που περιέχει είτε κώδικα στην ίδια τη γλώσσα, είτε binary ή τον συνδυασμό τους. Τα αρχεία wheel έχουν τυπική κατάληξη .whl και στην πράξη αποτελούν συμπιεσμένα αρχεία ZIP με συγκεκριμένη δομή των εσωτερικών αρχείων καθώς και συγκεκριμένο πρότυπο ονόματος που ορίζονται στο PEP 427. [9]

3.6. Manylinux

Το manylinux είναι ένα πρότυπο αλλά και σύνολο εργαλείων που διευκολύνει τη δημιουργία wheels συμβατά με όσο το δυνατόν περισσότερες διανομές και εκδόσεις διανομών Linux. Για να επιτευχθεί αυτό, το project προσφέρει έτοιμα Docker images βασισμένα σε όσο το δυνατόν παλαιότερες εκδόσεις της διανομής CentOS Linux μέσα στα οποία μπορεί κανείς να κάνει build τον κώδικά του. Με αυτόν τον τρόπο, η σύνδεση (linking) με βιβλιοθήκες του συστήματος όπως π.χ. η libc γίνεται με την εκάστοτε έκδοση της βιβλιοθήκης, πράγμα που διασφαλίζει την συμβατότητα του ABI με τις μελλοντικές εκδόσεις της βιβλιοθήκης λόγω backwards compatibility που τηρείται σε αυτές τις περιπτώσεις

Έτσι, αντί να χρειαστεί η εγκατάσταση ενός CentOS 6 για την πραγματοποίηση του build, αρκεί η λήψη και η εκτέλεση του manylinux docker image.

Το πρόβλημα έγκειται στο γεγονός ότι κάνοντας build ένα binary σε πρόσφατη έκδοση διανομής, το binary αυτό δε θα μπορεί να εκτελεστεί σε παλαιότερες εκδόσεις ακόμη και της ίδιας διανομής Linux λόγω ασυμβατότητας του ABI των βιβλιοθηκών συστήματος

Με το PEP 571 ορίστηκε το manylinux2010 που βασίζεται σε CentOS 6 καθώς το manylinux1 βασίζεται στο CentOS 5.11 για το οποίο σταμάτησε η υποστήριξη στις 31 Μαρτίου 2017. [10]

3.7. PyPI

Το PyPI είναι το Python Package Index. Αποτελεί ένα αποθετήριο και ευρετήριο πακέτων Python μέσω του οποίου χρήστες μπορούν να ανεβάσουν ή να κάνουν λήψη βιβλιοθηκών και projects σε Python. Στο PyPI καταχωρούνται πακέτα που ακολουθούν το πρότυπο wheel ή/και αρχεία που περιλαμβάνουν τον πηγαίο κώδικα (source distribution archives / sdist).

3.8. pytest

Το pytest είναι ένα framework για εκτέλεση unit tests στην γλώσσα Python. Βοηθάει στην εφαρμογή του Test Driven Development (TDD) ενώ είναι ένας τρόπος να υλοποιηθεί το κομμάτι του testing στο CI. Οι προγραμματιστές γράφουν εκτός από τον κώδικα μιας συνάρτησης, μεθόδου ή κλάσης και ένα σύνολο από use cases και τα αποτελέσματα που αναμένονται από αυτά τα use cases. Έτσι δημιουργείται ένας δομημένος τρόπος για τη διασφάλιση της ορθής λειτουργίας του κώδικα σε κάθε αλλαγή του από τους προγραμματιστές.

3.9. Auditwheel

Το auditwheel είναι ένα εργαλείο για τον έλεγχο της συμβατότητας ενός wheel με τα πρότυπα manylinux1, manylinux2010 και manylinux2014. Αυτό επιτυγχάνεται μέσω ελέγχων που πραγματοποιούνται για τις εξωτερικές βιβλιοθήκες από τις οποίες ένα wheel εξαρτάται πέραν εκείνων που ορίζονται στο πρότυπο manylinux.

Εκτός αυτού, το auditwheel παρέχει τη δυνατότητα επιδιόρθωσης ενός wheel που εξαρτάται από εξωτερικές βιβλιοθήκες ώστε να συμορφώνεται με το πρότυπο manylinux, τοποθετώντας τις επιπλέον βιβλιοθήκες εντός του πακέτου wheel και τροποποιώντας τα κατάλληλα RPATH ώστε να φορτώνονται οι ενσωματωμένες βιβλιοθήκες κατά την εκτέλεση. Αυτό επιτυγχάνει ένα αποτέλεσμα παρόμοιο με το να πραγματοποιούνταν static linking μεταξύ των δύο βιβλιοθηκών χωρίς όμως να γίνεται κάτι τέτοιο. [11]

3.10. Twine

Το Twine είναι ένα εργαλείο για τη δημοσίευση πακέτων wheel και sdist (source distribution) στο Python Package Index (PyPI) που λειτουργεί ανεξάρτητα από το setup.py script.

3.11. pyenv

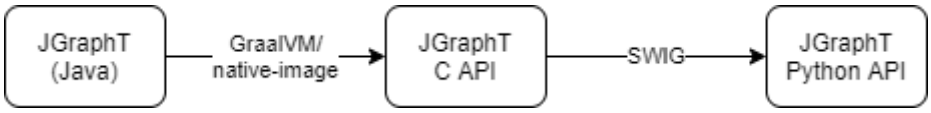
Τα pyenv και pyenv-win είναι δύο εργαλεία για MacOS και Windows αντίστοιχα που διευκολύνουν την εγκατάσταση και την εκτέλεση διαφορετικών εκδόσεων του CPython interpreter στο ίδιο σύστημα, χωρίς τη χρήση virtual environments.

4. Υλοποίηση

Σε αυτό το κεφάλαιο περιγράφεται η υλοποίηση του CI pipeline για τη βιβλιοθήκη python-jgrapht με τη χρήση των τεχνολογιών που περιγράφονται στην ενότητα 3.

Για να καταστεί δυνατή η χρήση στη γλώσσα Python της βιβλιοθήκης JGraphT που είναι γραμμένη σε Java, χρειάζεται να χρησιμοποιηθεί το ενδιάμεσο interface jgrapht-capi το οποίο είναι μια διεπαφή του JGraphT στη γλώσσα C. Το jgrapht-capi παράγεται με τη χρήση του GraalVM και του Native Image και αποτελείται από ένα native shared library καθώς και τα header files που ορίζουν τις διαθέσιμες δομές και συναρτήσεις της βιβλιοθήκης JGraphT. Στη

συνέχεια, με τη χρήση του SWIG δημιουργείται η διεπαφή (bindings) μεταξύ του jgrapht-capi shared library και της γλώσσας Python. Αυτή η διεπαφή στην πράξη είναι ένα επιπλέον shared library το οποίο μπορεί να χρησιμοποιηθεί απευθείας από την Python ως importable βιβλιοθήκη.



Σχ. 6: Βήματα για το build του python-jgrapht

Δεδομένου ότι αμφότερα το Native Image και το SWIG παράγουν C κώδικα και native binaries, θα απαιτούνταν από τον τελικό χρήστη της βιβλιοθήκης python-jgrapht να κάνει compile στον υπολογιστή του και το jgrapht-capi αλλά και το python-jgrapht shared library. Αυτή η διαδικασία είναι χρονοβόρα και με πολλές πιθανότητες για σφάλματα, ενώ θα έπρεπε να επαναλαμβάνεται για κάθε λειτουργικό σύστημα και για κάθε αρχιτεκτονική που ο προγραμματιστής θέλει να υποστηρίξει. Δεδομένου ότι η Python είναι μία cross-platform γλώσσα, είναι επιθυμητό η Python εντολή `import jgrapht` να λειτουργεί σε οποιαδήποτε πλατφόρμα λειτουργεί και ο ίδιος ο CPython interpreter.

Ο στόχος αυτής της εργασίας είναι να αυτοματοποιήσει τη διαδικασία παραγωγής όλων των ενδιάμεσων αρχείων και της τελικής βιβλιοθήκης python-jgrapht για περιβάλλοντα Linux, Windows, MacOS, την εκτέλεση unit tests με τη χρήση του pytest και να δώσει τη δυνατότητα για την έκδοση της τελικής βιβλιοθήκης στο PyPI.

Για την υλοποίηση του CI pipeline χρησιμοποιείται το Travis CI ως πλατφόρμα CI, ενώ το GitHub ως πλατφόρμα για τη φιλοξενία του git repository.

Η υλοποίηση χωρίζεται στα εξής στάδια:

1. Δημιουργία του setup.py
2. Linux/manylinux CI job
3. MacOS X CI job
4. Windows 10 CI job
5. pytest unit testing
6. Δημοσίευση έκδοσης στο PyPI

4.1 setup.py

Το setup.py είναι ένα βοηθητικό Python script που παρέχεται σε πολλά Python projects με σκοπό το build και την εγκατάσταση του project σε ένα σύστημα ενώ μπορεί να διευκολύνει και άλλες διαδικασίες, όπως ενδεικτικά η ανάπτυξη του κώδικα χωρίς να απαιτείται η εγκατάστασή του μετά από κάθε αλλαγή, το build της τεκμηρίωσης του project και η εκτέλεση των unit tests. Συνήθως, για την υλοποίηση αυτού του script χρησιμοποιούνται οι βιβλιοθήκες distutils και setuptools οι οποίες δημιουργούν ένα CLI με εντολές που μπορούν να εκτελεστούν στο project. [12]

Οι τυπικές εντολές είναι

build	Build όλων των απαραίτητων αρχείων για την εγκατάσταση
build_py	Αντιγραφή των Python αρχείων στον κατάλογο build/
build_ext	Build των C/C++ extension
build_clib	Build των βιβλιοθηκών C/C++ που χρησιμοποιούνται από τα C/C++ extensions
build_scripts	Αντιγραφή των βοηθητικών Python scripts στον κατάλογο build/
clean	Καθαρισμός προσωρινών αρχείων που δημιουργήθηκαν από την εντολή build
install	Εγκατάσταση των περιεχομένων του καταλόγου build/ στο σύστημα
install_lib	Εγκατάσταση των Python modules (αρχεία .py και extensions)
install_headers	Εγκατάσταση των C/C++ header files στο σύστημα
install_scripts	Εγκατάσταση των βοηθητικών script στο σύστημα
install_data	Εγκατάσταση αρχείων που δεν αποτελούν μέρος του κώδικα στο σύστημα
sdist	Δημιουργία ενός αρχείου (tar, zip, κ.α.) με τον κώδικα του project (source distribution)
register	Καταχώρηση του project στο PyPI
bdist	Δημιουργία ενός binary αρχείου του project που έχει γίνει build

Πίν. 1: Τυπικές εντολές ενός Python setup.py script

4.1.1 Ιδιαιτερότητες του python-jgrapht setup.py

Στο setup.py του python-jgrapht χρειάστηκε να γίνουν override ορισμένες εντολές, να δημιουργηθούν κάποιες νέες καθώς και να τοποθετηθούν κάποιες ειδικές περιπτώσεις για τη σωστή λειτουργία σε όλα τα επιθυμητά λειτουργικά συστήματα.

Συγκεκριμένα, ορίστηκε επιπλέον η εντολή build_capi με την οποία δημιουργείται το JGraphT C API που αποτελεί προαπαιτούμενο για το build του Python API.

Η setuptools εντολή build έγινε override καθώς εκτελούσε την υπο-εντολή build_py πριν από την υπο-εντολή build_ext, με αποτέλεσμα τα παραγόμενα Python αρχεία από το SWIG να μην συμπεριλαμβάνονται στο build/ directory και επομένως ούτε στο τελικό distribution του python-jgrapht.

Η εντολή build_ext έγινε override ώστε να εκτελεί την build_capi ως υπο-εντολή πριν δημιουργήσει το SWIG Python extension της βιβλιοθήκης.

Βασική προϋπόθεση του setup.py για τη βιβλιοθήκη python-jgrapht είναι το cross-platform - η δυνατότητα δηλαδή να λειτουργεί και να παράγει σωστό αποτέλεσμα σε Linux, MacOS και Windows.

4.2. CI/CD Pipeline

Για την υλοποίηση του CI/CD pipeline επιλέχθηκε η πλατφόρμα Travis CI η οποία παρέχει δωρεάν εκτέλεση CI pipelines σε έργα ανοιχτού λογισμικού όπως είναι το python-jgrapht. Η περιγραφή του Travis CI/CD pipeline γίνεται σε ένα αρχείο με όνομα `.travis.yml` το οποίο βρίσκεται στον αρχικό κατάλογο του git repository του project και είναι γραμμένο στη γλώσσα YAML. Η υπηρεσία Travis CI διαβάζει αυτόματα το αρχείο αυτό και εκτελεί τις περιγραφόμενες σε αυτό ενέργειες. Το σύνολο των ενεργειών αυτών αποτελεί ένα Travis build.

Ένα Travis build χωρίζεται σε stages ενώ κάθε stage αποτελείται από ένα σύνολο jobs. Το κάθε job εκτελείται παράλληλα με όλα τα υπόλοιπα jobs που ανήκουν στο ίδιο stage. Τα Travis jobs χωρίζονται σε 12 συνολικά phases, εκ των οποίων ορισμένα είναι προαιρετικά και η χρήση τους εξαρτάται από τις ανάγκες του project και τους στόχους του CI/CD pipeline. [13]

Το CI/CD pipeline που δημιουργήθηκε για το python-jgrapht αποτελείται από ένα μόνο stage (το προεπιλεγμένο) το οποίο περιλαμβάνει 3 διακριτά jobs - το καθένα εκ των οποίων αφορά ένα λειτουργικό σύστημα. Τα 3 αυτά jobs εκτελούνται παράλληλα στους servers του Travis και ανεξάρτητα το ένα από το άλλο.

Στο παρακάτω απόσπασμα YAML φαίνεται ο σκελετός των τριών Travis jobs για τα λειτουργικά συστήματα Linux, MacOS X και Windows

```
jobs:
  include:
    - os: linux
      ...
    - os: osx
      ...
    - os: windows
      ...
```

Στο καθένα από αυτά τα jobs ορίζεται ένας συνδυασμός από τα ακόλουθα keywords

os	Το λειτουργικό σύστημα στο οποίο θα εκτελεστεί το job
language	Η γλώσσα προγραμματισμού που είναι γραμμένο το λογισμικό
services	Επιπλέον υπηρεσίες που χρειάζεται να ενεργοποιηθούν για την εκτέλεση του job

env	Μεταβλητές περιβάλλοντος που θα είναι διαθέσιμες κατά την εκτέλεση του job
install	Προετοιμασία και εγκατάσταση προαπαιτούμενων πακέτων
script	Οι κύριες εντολές για την εκτέλεση του job
deploy	Διαδικασία για τη δημοσίευση του αποτελέσματος του CI pipeline

Πίν. 2: Βασικά keywords του Travis CI

4.2.1. Linux

Το build του python-jgrapht για Linux πραγματοποιείται εντός του manylinux2010 Docker image. Για να επιτευχθεί αυτό, απαιτείται η ενεργοποίηση του docker service στο .travis.yml ως εξής:

```
- os: linux
  services:
    - docker
```

Στη συνέχεια, ορίζεται το επιθυμητό Docker image, εν προκειμένω το manylinux2010_x86_64 ως μεταβλητή περιβάλλοντος

```
env: DOCKER_IMAGE=quay.io/pypa/manylinux2010_x86_64
```

Στο install phase γίνεται λήψη του docker image, ενώ στην κυρίως φάση, δηλαδή στο script phase εκτελείται το travis/build-manylinux2010_x86_64.sh script. Τέλος, εκτελείται η εντολή ls -l dist/ για να φανούν τα παραγόμενα αρχεία wheel στα αρχεία καταγραφής του job.

```
install:
  - docker pull $DOCKER_IMAGE
script:
  - docker run --rm -v `pwd`:/io $DOCKER_IMAGE
/io/travis/build-manylinux2010_x86_64.sh
  - ls -l dist/
```

Το script travis/build-manylinux2010_x86_64.sh αρχικά εξασφαλίζει ότι εντός του manylinux docker container θα εγκατασταθούν τα εξής dependencies

CMake	Για το build του JGraphT C API
GraalVM / native-image	Για τη δημιουργία του JGraphT C API
Maven	Για το build του JGraphT
SWIG	Για τη δημιουργία των Python bindings του JGraphT C API

Πίν. 3: Dependencies για το build του python-jgrapht σε περιβάλλον manylinux2010

Στην πορεία, το script εκτελεί:

- την εντολή bdist_wheel του setup.py για τη δημιουργία των wheel για όλες τις υποστηριζόμενες εκδόσεις Python που βρίσκονται εντός του manylinux Docker image
- την εντολή auditwheel show για κάθε παραγόμενο wheel ώστε να τυπωθεί στα CI logs εάν τα wheels συμβαδίζουν με τις προδιαγραφές του manylinux
- την εντολή sdist του setup.py για τη δημιουργία του source distribution του project, το οποίο στην πορεία θα δημοσιευτεί στο PyPI μαζί με τα wheels

Τέλος, αφού εγκατασταθεί για κάθε έκδοση Python το αντίστοιχο wheel της βιβλιοθήκης με τη χρήση του pip, εκτελούνται τα unit tests για όλες τις διαθέσιμες εκδόσεις Python μέσω του pytest.

```
4450 ================== 419 passed, 2 skipped, 3 warnings in 2.08s ==================
4451 The command "docker run --rm -v `pwd`:/io $DOCKER_IMAGE /io/travis/build-manylinux2010_x86_64.sh" exited with 0.
4452 $ ls -l dist/
4453 total 30484
4454 -rw-r--r-- 1 root root 10259286 Jan 18 10:07 jgrapht-1.5.0.4.dev0-cp36-cp36m-manylinux2010_x86_64.whl
4455 -rw-r--r-- 1 root root 10260159 Jan 18 10:07 jgrapht-1.5.0.4.dev0-cp37-cp37m-manylinux2010_x86_64.whl
4456 -rw-r--r-- 1 root root 10300054 Jan 18 10:08 jgrapht-1.5.0.4.dev0-cp38-cp38-manylinux2010_x86_64.whl
4457 -rw-r--r-- 1 root root   391453 Jan 18 10:08 jgrapht-1.5.0.4.dev0.tar.gz
4458 The command "ls -l dist/" exited with 0.
4459
4460 Skipping a deployment with the pypi provider because this is not a tagged commit
4461
4462 Done. Your build exited with 0.
```

Εικ. 1: Έξοδος του Travis CI pipeline για το Linux job του python-jgrapht

Καθότι το python-jgrapht αποτελείται από δύο chained shared libraries: το C API με όνομα αρχείου libjgrapht_capi.so και το Python extension με όνομα αρχείου _backend.cpython-[xx]-[arch]-linux-gnu.so, ιδιαίτερη προσοχή χρειάστηκε κατά τη δημιουργία του Python extension να οριστεί το RPATH του στο σωστό directory στο οποίο τοποθετείται το libjgrapht_capi.so. Σε διαφορετική περίπτωση, το Python extension δε θα ήταν δυνατό να φορτώσει το C API shared object. Για αυτό το λόγο, κατά τη δήλωση του extension στο setup.py ήταν απαραίτητη η δήλωση του τελικού path στο οποίο τοποθετούνται τα shared objects μέσω της παραμέτρου runtime_library_dirs στην οποία δίνεται ως μοναδικό path το $ORIGIN. Το $ORIGIN ως τιμή του RPATH στο _backend.so υποδυκνύει στο shared object ότι μπορεί να εντοπίσει και να φορτώσει την libjgrapht_capi.so - από την οποία εξαρτάται - στο ίδιο directory στο οποίο βρίσκεται και η ίδια. Τα εργαλεία readelf και ldd αποδεικνύονται ιδιαίτερα χρήσιμα για την προβολή των εξαρτήσεων μιας βιβλιοθήκης καθώς και για την τιμή που παίρνει το RPATH και τις υπόλοιπες ιδιότητες ενός ELF.

```
_backend_extension = Extension('jgrapht._backend',
['jgrapht/backend.i','jgrapht/backend.c'],
                               include_dirs=['jgrapht/',
'vendor/build/jgrapht-capi/', 'vendor/build/jgrapht-capi/src/main/native'],
                               library_dirs=['vendor/build/jgrapht-capi/'],
                               libraries=['jgrapht_capi'],
                               runtime_library_dirs=['$ORIGIN'],
                               extra_link_args=extra_link_args,
                               )
```

4.2.2. MacOS X

Σε αυτή την ενότητα αναλύεται η διαδικασία που ακολουθείται για το build του python-jgrapht σε περιβάλλον MacOS X.

Στο MacOS X job, ορίστηκαν το install phase και το script phase. Στο install phase εγκαθίστανται μέσω του Homebrew το GraalVM, το Native Image και το SWIG καθώς και οι υποστηριζόμενες εκδόσεις Python ταυτόχρονα στο ίδιο σύστημα με τη χρήση του pyenv. Στο script phase καλείται το travis/build-macos.sh script το οποίο χρησιμοποιώντας το pyenv δημιουργεί και εγκαθιστά το python-jgrapht wheel για κάθε επιθυμητή έκδοση Python. Στην πορεία εγκαθιστά τα wheel και εκτελεί τα unit tests.

```
    - os: osx
      language: shell
      sudo: required
      install:
        - brew cask install graalvm/tap/graalvm-ce-java11
        - export PATH=/Library/Java/JavaVirtualMachines/graalvm-ce-
java11-20.1.0/Contents/Home/bin:"$PATH"
        - gu install native-image
        - brew install swig
        - pip3 install twine
        - pyenv install 3.6.9
        - pyenv install 3.7.5
        - pyenv install 3.8.0
      script:
        - travis/build-macos.sh
        - ls -l dist/
```

Αντίστοιχα με το Linux job, χρειάστηκε ιδιαίτερη προσοχή στο setup.py καθώς η παράμετρος runtime_library_dirs δεν είχε το επιθυμητό αποτέλεσμα σε περιβάλλον Darwin. Για να αντιμετωπιστεί αυτό το πρόβλημα, χρειάστηκε να προστεθεί η οδηγία για το RPATH απευθείας στον linker. Αυτό επιτεύχθηκε με τη χρήση της παραμέτρου extra_link_args στο Extension object του setup.py και συγκεκριμένα δίνοντάς της την τιμή `['-Wl,-rpath,@loader_path']`.

Προϋπόθεση για να λειτουργήσει αυτό ήταν η χρήση του install_name_tool κατά το build του C API shared library ώστε να οριστεί το DYLIB ID του παραγόμενου Mach-O αρχείου στο σωστό path. Αυτό πραγματοποιήθηκε στο στάδιο του CMake του jgrapht-capi project. Για να εξασφαλιστεί η εκτέλεση του install_name_tool μόνο σε περιβάλλον Darwin η εντολή προστέθηκε εντός ενός if(APPLE) clause.

```
if(APPLE)
  add_custom_command(
    OUTPUT ${JGRAPHT_LIBRARY} APPEND
    COMMAND install_name_tool -id "@rpath/${JGRAPHT_LIBRARY}"
${CMAKE_BINARY_DIR}/${JGRAPHT_LIBRARY}
  )
endif(APPLE)
```

4.2.3. Windows 10

Το Travis CI, στο Windows περιβάλλον του παρέχει το Git Bash ως shell για την εκτέλεση εντολών, το Visual Studio 2017 ως toolchain για compile και το Chocolatey ως package manager. [14]

Το Git Bash είναι μέρος του Git for Windows project και αποτελεί μια προσομοίωση του Bash shell σε περιβάλλον Windows.

Το Chocolatey είναι ένα λογισμικό διαχείρισης πακέτων και ταυτόχρονα ένα αποθετήριο πακέτων για Windows. Τα διαθέσιμα πακέτα δημιουργούνται και συντηρούνται από την κοινότητα του Chocolatey ενώ είναι πλήρως encapsulated και η διαδικασία εγκατάστασής τους αυτοματοποιημένη.

Για το build του python-jgrapht σε Windows 10 περιβάλλον χρειάστηκε η εγκατάσταση του pyenv-win του Maven, του GraalVM και του SWIG μέσω του Chocolatey στο install phase του αντίστοιχου CI job.

Στο script phase, για να μπορέσουν το setup.py του python-jgrapht και το CMake του jgrapht-capi να εντοπίσουν τον MSVC compiler του Microsoft Visual Studio ήταν αναγκαία η κλήση του vcvars64.bat script. Το vcvars64.bat script είναι μέρος του Visual Studio και ο ρόλος του είναι ο ορισμός των απαραίτητων μεταβλητών περιβάλλοντος για να λειτουργήσει σωστά ο MSVC compiler. Επιπλέον, για την σωστή λειτουργία του CMake του jgrapht-capi χρειάστηκε να οριστεί η μεταβλητή περιβάλλοντος CMAKE_GENERATOR με τιμή “Visual Studio 15 2017 Win64” καθώς και η επιθυμητή αρχιτεκτονική των παραγόμενων shared libraries με τη μεταβλητή περιβάλλοντος CMAKE_GENERATOR_TOOLSET και τιμή “host=x64”.

Στην πορεία, αντίστοιχα με τα βήματα στα περιβάλλοντα Linux και Darwin και με τη χρήση του pyenv-win γίνονται build τα wheels του python-jgrapht για όλες τις υποστηριζόμενες εκδόσεις Python και εκτελούνται τα unit tests με τη βοήθεια του pytest.

Μια σημαντική παράμετρος της διαδικασίας σε περιβάλλον Windows ήταν ότι η κλήση του vcvars64.bat script έπρεπε να γίνει με τις μεταβλητές που θα όριζε να παραμένουν διαθέσιμες στο περιβάλλον για τις επόμενες εντολές. Αυτό σε Bash επιτυγχάνεται με την χρήση της εντολής source. Καθότι όμως το Git Bash είναι μια προσομοίωση του Bash δεν παρέχει τη δυνατότητα να εκτελεστεί η εντολή source με παράμετρο ένα .bat αρχείο, ενώ επιτρέπει την απευθείας εκτέλεση ενός .bat αρχείου. Επομένως, ολόκληρη η διαδικασία για το build του python-jgrapht

έπρεπε να περιγραφεί σε ένα νέο .bat script (travis/build.bat) στο οποίο η πρώτη εντολή είναι η “call vcvars64.bat” που αποτελεί την αντίστοιχη της “source” στο Bash.

4.2.4. Release

Στο τέλος του CI/CD pipeline στο αρχείο .travis.yml ορίζεται η ενότητα deploy. Σε αυτή την ενότητα περιγράφεται η διαδικασία για τη δημοσίευση όλων των παραγόμενων στοιχείων από τη διαδικασία build του python-jgrapht για όλα τα περιβάλλοντα και για όλες τις υποστηριζόμενες εκδόσεις Python.

Το Travis CI υποστηρίζει ως deploy provider το PyPI ορίζοντας την ιδιότητα “deploy.provider” στην τιμή “pypi”. Με αυτόν τον τρόπο, και έχοντας διαθέσιμο ένα λογαριασμό στο PyPI, το ίδιο το Travis CI μπορεί να δημοσιεύσει σε αυτό τα παραγόμενα wheels καθώς και το source distribution αυτόματα. Αντί για τη δημοσίευση πακέτων με προσωπικό λογαριασμό χρήστη, το PyPI προσφέρει τη δυνατότητα δημιουργίας ενός token με write access για χρήση από scripts ή CI/CD πλατφόρμες. Έτσι, ορίζοντας την τιμή “__token__” στην παράμετρο deploy.user και το environment variable PYPI_PASSWORD εντός του web interface του Travis CI στην τιμή του token από το PyPI δίνεται η δυνατότητα στο Travis CI να έχει πρόσβαση δημοσίευσης νέων πακέτων στο PyPI εκ μέρους ενός οποιουδήποτε λογαριασμού.

Καθώς το pipeline που περιγράφεται στο .travis.yml εκτελείται για κάθε commit που πραγματοποιείται στο git repository του python-jgrapht project και επειδή δεν είναι επιθυμητή η δημοσίευση νέας έκδοσης για κάθε νέο commit, δηλώνεται - με τη χρήση του “on.tags: true” - στο deploy phase ο περιορισμός της δημοσίευσης των παραγόμενων artifacts στο PyPI μόνο εφόσον στο commit έχει εφαρμοστεί ένα git tag. Με αυτό τον τρόπο, αναμένεται ότι για κάθε νέο release του project θα δημιουργείται και ένα αντίστοιχο tag με το όνομα της νέας έκδοσης. Σε αυτή και μόνο σε αυτή την περίπτωση θα δημοσιεύονται και τα παραγόμενα αρχεία αυτόματα από το Travis CI στο PyPI.

```
deploy:
  provider: pypi
  distributions: "sdist bdist_wheel"
  # Do not delete generated build files (i.e. wheels)
  skip_cleanup: true
  # Do not attempt to upload to PyPI an already uploaded wheel
  skip_existing: true
  user: __token__
  # PyPI password should be stored on Travis CI as a secret env
variable named PYPI_PASSWORD
  on:
    tags: true
```

5. Συμπεράσματα

Σε αυτή την εργασία αφού δόθηκε μια σύντομη θεωρητική βάση για τις έννοιες CI/CD και πραγματοποιήθηκε μια ανασκόπηση των διαθέσιμων εργαλείων και υπηρεσιών για την υλοποίηση ενός CI/CD pipeline, εξετάστηκε η υλοποίηση ενός cross-platform CI/CD pipeline για τη βιβλιοθήκη γράφων python-jgrapht με τη χρήση της πλατφόρμας Travis CI, καθώς και τα προβλήματα που χρειάστηκε να αντιμετωπιστούν για την εκτέλεση του pipeline σε τρία διαφορετικά λειτουργικά συστήματα (Linux, MacOS X, Windows) και σε τρεις διαφορετικές εκδόσεις της CPython. Ως αποτέλεσμα αυτής της εργασίας, κάθε αλλαγή στον κώδικα που πραγματοποιείται στο επίσημο αποθετήριο του python-jgrapht ενεργοποιεί το CI/CD pipeline το οποίο εντός μερικών λεπτών εκτελεί τα αυτοματοποιημένα unit tests για τον έλεγχο της ποιότητας του κώδικα και δημιουργεί αυτόματα εννέα διαφορετικά πακέτα wheel τα οποία προαιρετικά μπορούν και να δημοσιευθούν ως νέες εκδόσεις της βιβλιοθήκης στο Python Package Index.

Ο χρόνος που απαιτείται για την ολοκλήρωση των jobs, συμπεριλαμβανομένου του build και των unit tests εμφανίζεται στον παρακάτω πίνακα για κάθε λειτουργικό σύστημα:

Λειτουργικό σύστημα	**Χρόνος ολοκλήρωσης**
Linux	8 λεπτά
MacOS X	17 λεπτά
Windows	10 λεπτά

Πίν. 4: Χρόνοι εκτέλεσης του CI pipeline για το python-jgrapht ανά λειτουργικό σύστημα

Λόγω της παράλληλης εκτέλεσης των jobs, ο συνολικός χρόνος που χρειάζεται να ολοκληρωθεί ολόκληρο το pipeline ταυτίζεται με τον χρόνο του πιο αργού job, δηλαδή 17 λεπτά. Ο αθροιστικός χρόνος εκτέλεσης των jobs είναι 35 λεπτά.

Η χρήση του CI/CD pipeline σε συνδυασμό με τα unit tests επιτρέπει στους προγραμματιστές να επικεντρωθούν στον κώδικα, γλιτώνοντας πολύτιμο χρόνο, ενώ οι συνεισφορές από τρίτους μέσω pull requests μπορούν στιγμιαία να επαληθευτούν για την ορθότητά τους και να απορριφθούν πολύ νωρίς σε περίπτωση που αποτυγχάνει το CI pipeline, χωρίς να δαπανηθεί χρόνος για review.

6. Πιθανές βελτιώσεις

Σε αυτή την ενότητα περιγράφονται ενδεικτικά μερικές βελτιώσεις ή προσθήκες χαρακτηριστικών που θα μπορούσαν να γίνουν πέραν των πλαισίων αυτής της εργασίας.

6.1 codecov

Το codecov είναι ένα εργαλείο και υπηρεσία για την εξαγωγή και καταγραφή στατιστικών σε σχέση με τα ποσοστά κάλυψης του κώδικα από unit tests. Σύμφωνα με τη φιλοσοφία του Test Driven Development, πριν από τη συγγραφή κώδικα για την υλοποίηση ενός νέου feature, προηγείται η συγγραφή ενός unit test το οποίο δρα ως έλεγχος για την ορθή υλοποίηση του feature. Για παράδειγμα, για την υλοποίηση μιας συνάρτησης πρόσθεσης δύο αριθμών

6.2 Caching

Σε πολλές περιπτώσεις, για να γίνει build ένα λογισμικό, χρειάζεται να γίνει λήψη άλλων εργαλείων, όπως π.χ. ένας compiler, διάφορες βιβλιοθήκες, κ.α. Εκτός από τη λήψη εξαρτήσεων, ενδέχεται να χρειάζεται ακόμη και το build κάποιων εξ' αυτών. Και οι δύο αυτές διαδικασίες μπορεί να αποβούν ιδιαίτερα χρονοβόρες, και δεδομένου ότι επαναλαμβάνονται σε κάθε commit με τον ίδιο ακριβώς τρόπο και με το ίδιο ακριβώς αποτέλεσμα, είναι σκόπιμο να μπορούν να τοποθετηθούν κατευθείαν τοπικά στο μηχάνημα εκτέλεσης του CI pipeline. Το Travis CI, όπως και τα περισσότερα συστήματα CI, δίνει τη δυνατότητα για τοπική αποθήκευση εξαρτήσεων που χρησιμοποιούνται συχνά σε κάθε επανάληψη του CI pipeline. Αυτό το χαρακτηριστικό ονομάζεται caching.

Στην παρούσα εργασία, αρχικά θα μπορούσε να υλοποιηθεί το Travis CI caching για το compilation του SWIG που εκτελείται στο Linux build job, ενώ ενδεχομένως να παρουσιαζόταν μια μικρή επιτάχυνση του pipeline εάν πραγματοποιούνταν caching και στα Maven dependencies που εγκαθίστανται κατά το build του jgrapht-capi.

6.3 Υποστήριξη arm64 και άλλων αρχιτεκτονικών

Στα πλαίσια της εργασίας αυτής υλοποιήθηκε ένα CI/CD pipeline για τη δημιουργία wheels της βιβλιοθήκης python-jgrapht στην αρχιτεκτονική x86_x64. Υπάρχουν πολλές συσκευές που χρησιμοποιούν άλλες αρχιτεκτονικές, μία εκ των οποίων είναι η arm64. Η δημιουργία wheel για arm64 μπορεί να αποδειχθεί χρήσιμη σε περίπτωση που κάποιος χρήστης επιθυμεί να εκτελέσει τη βιβλιοθήκη python-jgrapht σε συσκευές όπως το Raspberry Pi αλλά σίγουρα απαιτεί επιπλέον βήματα, καθώς θα πρέπει να βρεθούν και όλες οι εξαρτήσεις που χρειάζονται για το build της βιβλιοθήκης στην ίδια αρχιτεκτονική, πράγμα το οποίο κατά τη συγγραφή της εργασίας δεν ήταν εφικτό σε όλες τις περιπτώσεις.

6.4 Εύκολη προσθήκη νέων εκδόσεων της CPython

Οι υποστηριζόμενες εκδόσεις της CPython εμφανίζονται σε πολλαπλά σημεία μέσα στο pipeline καθώς και στα build scripts για κάθε λειτουργικό σύστημα. Αυτό σημαίνει πως αν χρειαστεί να προστεθεί ή να καταργηθεί μία έκδοση της CPython, θα πρέπει να αλλαχθούν όλα τα σχετικά σημεία που αναφέρονται οι επιθυμητές εκδόσεις. Θα ήταν επομένως μια χρήσιμη βελτίωση το να οριστούν οι επιθυμητές εκδόσεις μία φορά σε κάποιο κεντρικό σημείο αναφοράς του CI/CD pipeline και να χρησιμοποιούνται από εκεί σε όλα τα jobs.

Βιβλιογραφία

[1] Michail, D., Kinable, J., Naveh, B., & Sichi, J. V. (2020). JGraphT—A Java Library for Graph Data Structures and Algorithms. *ACM Transactions on Mathematical Software, 46*(2), 1-29. doi:10.1145/3381449

[2] Booch, Grady. *Object-Oriented Analysis and Design: with Applications*. Benjamin Cummings, 1991

[3] Basu, Soumyajit. "Continuous Integration: Its History and Benefits - DZone DevOps." *Dzone.com*, DZone, 24 May 2017, dzone.com/articles/continuous-integration-and-its-whereabouts.

[4] Beck, K. (1999). Embracing change with extreme programming. *Computer, 32*(10), 70-77. doi:10.1109/2.796139

[5] Stolberg, S. (2009). Enabling Agile Testing through Continuous Integration. *2009 Agile Conference*. doi:10.1109/agile.2009.16

[6] Schaefer, A., Reichenbach, M., & Fey, D. (2013). Continuous integration and automation for DevOps. In *IAENG Transactions on Engineering Technologies* (pp. 345-358). Springer, Dordrecht.

[7] Sharma, A. (2018, February 01). A Brief History of DevOps, Part III: Automated Testing and Continuous Integration. Retrieved from https://circleci.com/blog/a-brief-history-of-devops-part-iii-automated-testing-and-continuous-integration/

[8] Native Image (n.d.). Retrieved from https://www.graalvm.org/reference-manual/native-image/

[9] PEP 427 -- The Wheel Binary Package Format 1.0. (n.d.). Retrieved from https://www.python.org/dev/peps/pep-0427/

[10] PEP 571 -- The manylinux2010 Platform Tag. (n.d.). Retrieved from https://www.python.org/dev/peps/pep-0571/

[11] Pypa. (n.d.). Pypa/auditwheel. Retrieved from https://github.com/pypa/auditwheel

[12] Building and Distributing Packages with Setuptools. (n.d.). Retrieved from https://setuptools.readthedocs.io/en/latest/setuptools.html

[13] Travis CI Documentation. (n.d.). Retrieved from https://docs.travis-ci.com/user/for-beginners/#builds-stages-jobs-and-phases

[14] Travis CI Documentation. (n.d.). Retrieved from https://docs.travis-ci.com/user/reference/windows/